IAAS CLOUD SERVICE SELECTION USING CASE-BASED REASONING

By

Sima Soltani

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen’s University
Kingston, Ontario, Canada
February 2016

Copyright © Sima Soltani 2016
Abstract

Cloud computing provides on-demand resources and removes the boundaries of resources' physical locations. It allows vendors to save upfront infrastructure costs and focus on features that discriminate their businesses. Growing number of provided services makes manual selection of the most suitable service time consuming and very hard which requires expertise. The large number of features and properties that the services are characterized by makes automatic service selection challenging.

In this thesis we present QuARAM Service Recommender, a self-adaptive Infrastructure-as-a-Service (IaaS) service selection system that recommends a list of suitable services for cloud application deployment based on an application's requirements and the customer's preferences. The process starts with automatic extraction of an application's features, requirements and preferences and ends with a list of potential services for the application deployment. TOSCA provides a standard way of specifying the cloud application. Defined Normative Types in TOSCA do not cover defining all the requirements, features, and customer's preferences. In this thesis we propose an extension to the TOSCA Normative Types, so our system can extract all the information required for service selection automatically from the specification of the application.

We use case-based reasoning to provide a recommendation of suitable services for application deployment. This method can be beneficial for cloud customers in service selection even when lacking complete knowledge about their application or features offered by cloud services. It can efficiently handle heterogeneous attributes that characterize cloud services and the requirements of cloud applications and is able to integrate the customer's preferences through assigning weights to these attributes. The feedback from both customers and the monitoring system is used to automatically adapt the system behavior and enhance the quality of recommendations. We use MCDM method for cloud service selection when there are not sufficient cases in the system case base and we use clustering to handle the problem of a large search space. We further describe a service consolidation method to improve the resource utilization and reduce
the total service price. Our step-by-step case study demonstrates that an automatic IaaS service selection using a combination of all the proposed approaches is practical and achievable.
Acknowledgment

First and foremost, I thank God, for his mercy and guidance.

I would like to express my sincere gratitude to my supervisor Prof. Patrick Martin for the continuous support of my PhD study and related research, for his patience, motivation, and immense knowledge. I had his support and help along all the way. I could not have imagined having a better supervisor for my PhD study.

I would like to thank my co-supervisor, Dr. Khalid Elgazzar, for his insightful comments and encouragement, his knowledge and guidance, and his patience through finishing this thesis.

Special thanks are due to Wendy Powley for her continuous help, support and encouragement. It is a pleasure to thank my best friend Mastoureh Hassannezhad for her non-stopping support.

I thank all the members of the Queen’s School of computing, especially the members of The Database Lab for their support, collaboration, friendship and insightful feedbacks.

Last but not the least, I would like to thank my family: my parents who are always encouraging and supporting me. I owe them any success I made in my life. I would like to thank my husband, Farid, for his support during this long journey with all the ups and downs. I am also in debt to my son, Kian, for being a part of my PhD. And thanks to my brothers, Soheil and Mohammad, for supporting me spiritually throughout this chapter of my life.
Statement of originality

I hereby certify that this Ph.D. thesis is original and that all ideas and inventions attributed to others have been properly referenced.

Sima Soltani
February 7th, 2016
Contents

Abstract........................................................................................................................................................................... ii
Acknowledgment............................................................................................................................................................ iv
Statement of originality................................................................................................................................................... v
Contents ........................................................................................................................................................................... vi
List of Figures............................................................................................................................................................... ix
List of Tables ................................................................................................................................................................... x
List of Listings ............................................................................................................................................................... xi
List of Acronyms ........................................................................................................................................................... xii

Chapter 1 Introduction................................................................................................................................................. 1
  1.1 Motivation .......................................................................................................................................................... 3
  1.2 Contributions .................................................................................................................................................. 8
  1.3 Thesis organization ......................................................................................................................................... 9

Chapter 2 Background and Literature Review............................................................................................................ 11
  2.1 Cloud Service Selection ............................................................................................................................... 11
  2.2 Case-Based Reasoning ............................................................................................................................... 13
    2.2.1 Knowledge Representation in CBR.................................................................................................... 15
    2.2.2 Case Memory Models (Case Base Organization) .............................................................................. 16
    2.2.3 CBR Cycle ........................................................................................................................................... 26
    2.2.4 Case Based Recommendation ........................................................................................................... 50
  2.3 Open Research Issues .................................................................................................................................... 51

Chapter 3 QuARAM Cloud Application Management Framework................................................................................. 52
  3.1 Introduction ..................................................................................................................................................... 52
  3.2 Example Web Application ............................................................................................................................ 54
  3.3 QuARAM Application Management Framework ........................................................................................ 55
    3.3.1 Deployment Engine .............................................................................................................................. 56
    3.3.2 Recommender ..................................................................................................................................... 58
    3.3.3 Simulator .............................................................................................................................................. 61
    3.3.4 Deployment Manager .......................................................................................................................... 61
    3.3.5 Runtime Manager ............................................................................................................................... 61
  3.4 Summary ......................................................................................................................................................... 63

Chapter 4 Requirements Identification and TOSCA Normative Type Extension......................................................... 64
  4.1 Introduction ....................................................................................................................................................... 64
  4.2 Requirements Determination ........................................................................................................................ 65
  4.3 Topology and Orchestration Specification for Cloud Application (TOSCA)............................................. 70
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>142</td>
</tr>
<tr>
<td>7.2</td>
<td>QuARAM Service Recommender</td>
<td>143</td>
</tr>
<tr>
<td>7.2.1</td>
<td>General Description</td>
<td>144</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Components Description</td>
<td>144</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Recommendation Process</td>
<td>147</td>
</tr>
<tr>
<td>7.3</td>
<td>Similarity Threshold</td>
<td>149</td>
</tr>
<tr>
<td>7.4</td>
<td>Validation of the QuARAM Service Recommender</td>
<td>152</td>
</tr>
<tr>
<td>7.5</td>
<td>Summary</td>
<td>159</td>
</tr>
<tr>
<td>8.1</td>
<td>Summary</td>
<td>163</td>
</tr>
<tr>
<td>8.2</td>
<td>Future works</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>167</td>
</tr>
<tr>
<td>8.1</td>
<td>Appendix A Service Template for MyWebApplication</td>
<td>182</td>
</tr>
<tr>
<td>8.1</td>
<td>Appendix B Explanation of application attributes (features)</td>
<td>188</td>
</tr>
<tr>
<td>8.1</td>
<td>Appendix C Service selection results for applications 2 to 4</td>
<td>190</td>
</tr>
<tr>
<td>8.1</td>
<td>Appendix D Application 1 specification and service selection results</td>
<td>194</td>
</tr>
</tbody>
</table>

Chapter 8 Conclusion

References

Appendix A Service Template for MyWebApplication

Appendix B Explanation of application attributes (features)

Appendix C Service selection results for applications 2 to 4

Appendix D Application 1 specification and service selection results
List of Figures

Figure 1.1. Sample cloud application entities, requirements and customer priorities for cloud deployments ................................................................. 4
Figure 2.1. CBR cycle proposed by Aamodt and Plaza [28] ................................................................................................................................. 15
Figure 2.2. Three-layer structure of a case base (adapted from [52]) ................................................................................................................. 17
Figure 2.3. Structure of case base in dynamic memory model (adapted from [28]) ................................................................. 18
Figure 2.4. A two dimensional search space and how the k-d tree split the space (adapted from [53]) .................................................... 19
Figure 2.5. Case base in Footprint model (adapted from [56]) ......................................................................................................................... 20
Figure 2.6. Example of CRN in travel agency domain (adapted from [57]) .................................................................................................. 21
Figure 2.7. Polyhedral cases and case base as a network of these cases (adapted from [58]) .......................................................... 22
Figure 2.8. Hierarchy of Memory models ......................................................................................................................................................... 23
Figure 2.9. Hierarchy of retrieval methods ..................................................................................................................................................... 28
Figure 2.10. Two stage retrieval process in Footprint retrieval method (adapted from [56]) ................................................................. 30
Figure 2.11. DH-S based method representation of the cases in the case base (adapted from [68]) .......................................................... 32
Figure 2.12. Learning adaptation data from case base (adapted from [36]) .............................................................................................. 40
Figure 3.1. Application Management Tasks ..................................................................................................................................................... 54
Figure 3.2. The high level architecture of the proposed QuARAM framework .................................................................................................. 56
Figure 3.3. Application Service Template for the Web application ............................................................................................................. 57
Figure 3.4. Finding deployment resources .................................................................................................................................................. 59
Figure 4.1. Service Template schema (adapted from [9]) ............................................................................................................................ 71
Figure 4.2. Requirements and Capabilities definition in TOSCA (adapted from [9]) ..................................................................................... 72
Figure 4.3. MyWebapplication topology ...................................................................................................................................................... 73
Figure 5.1. Example of Memory value intervals ........................................................................................................................................ 89
Figure 5.2. Constant asymmetric function ................................................................................................................................................. 91
Figure 5.3. Relative asymmetric function ................................................................................................................................................ 92
Figure 5.4. Steps of our implementation of the learning adaptation knowledge method ............................................................................ 95
Figure 5.5. QuARAMRecommender CBR architecture .................................................................................................................................. 96
Figure 5.6. Upgrade transition graph for Amazon EC2 instances ............................................................................................................. 100
Figure 5.7. Adaptor component in QuARAMRecommender CBR ............................................................................................................... 102
Figure 5.8. Effect of weights on the precision of the recommendation .................................................................................................... 105
Figure 5.9. The impact of adaptation on the precision of recommendations .............................................................................................. 105
Figure 5.10. Effect of reinforcement learning on the accuracy of the results .......................................................................................... 106
Figure 6.1. Comparison of TOSIS and LFF on similarity, price and performance of selected services ...................................................... 130
Figure 6.2. Effect of using clustering in service selection response time .................................................................................................. 132
Figure 6.3. Precision of clustering and non-clustering method in service selection .................................................................................. 133
Figure 7.1. The architecture of QuARAM Service Recommender .......................................................................................................... 143
Figure 7.2. The relation between similarity and precision .......................................................................................................................... 151
Figure 7.3. The EER schema of application case base ............................................................................................................................... 153
Figure 7.4. The process flow between the components of the recommender ........................................................................................... 154
Figure 7.5. The .csv files pertaining to server1 of the application .............................................................................................................. 155
Figure 7.6. Query for Service Search Engine (.csv file) ............................................................................................................................... 156
List of Tables

Table 2.1. Memory models, advantages and disadvantages ................................................................. 24
Table 2.2. Case base reduction methods.................................................................................................. 47
Table 4.1. Top 10 IaaS providers and the price and SLA features. ......................................................... 67
Table 5.1. Attributes of the problem part of cases in the Application Case Base................................. 99
Table 5.2. Similarity functions and QuARAMRecommender CBR case attributes ............................... 101
Table 5.3. Assigned weights for the case’s attributes ............................................................................ 104
Table 6.1. Similarity, price and performance of 5 most similar services to a deployment entity .......... 114
Table 6.2. Information table of TOPSIS .............................................................................................. 115
Table 6.3. System requirements of application 1 .................................................................................. 120
Table 6.4. System requirements of application 2 .................................................................................. 120
Table 6.5. Application 3 system requirements ...................................................................................... 121
Table 6.6. System requirements of application 4 .................................................................................. 121
Table 6.7. Selected services for application 1 using LFF ..................................................................... 123
Table 6.8. Selected services for application 1 using TOPSIS .............................................................. 124
Table 6.9. Similarity and price of different providers for application 1 using LFF ................................. 125
Table 6.10. Similarity and price of different providers for applications 2, 3 and 4 using LFF .......... 126
Table 6.11. Similarity and price of different providers for application 1 using TOSIS ............................ 127
Table 6.12. Similarity and price of different providers for applications 2, 3 and 4 using TOPSIS ......... 127
Table 6.13. Potential providers for applications deployment (using LFF) ............................................. 129
Table 6.14. Potential providers for applications deployment (using TOPSIS) ........................................ 129
Table 6.15. Weights for system requirements ...................................................................................... 132
Table 6.16. A deployment entity requirements of application 2 ........................................................... 134
Table 6.17. Suggested cloud service for the deployment entity in Table 6.16 ......................................... 134
Table 6.18. Percentage of cost reduction in tested applications .......................................................... 138
Table 6.19. Percentage of reduced number of services for tested applications ................................... 139
Table 6.20. Dollar saving per hour for tested applications ................................................................. 139
Table 7.1. Example calculation of precision based on similarity ranges .............................................. 150
Table 7.2. Recommended services for each deployment entity ............................................................. 157
Table 7.3. Results of service consolidation for application 1 ............................................................... 158
List of Listings

Listing 4.1. Extended properties of Node Type "QuARAMServer" .......................................................... 76
Listing 4.2. Extended properties of the Node Type "QuARAMOperatingSystem" .................................. 77
Listing 4.3. Extended properties for Node Type "QuARAMDBMS" ........................................................ 78
Listing 4.4. QuARAMLoadBalancer Node Type definition ................................................................. 78
Listing 4.5. Properties of New Node Type "QuARAMLoadBalancer" .................................................... 79
Listing 4.6. New Policy Types ............................................................................................................. 79
Listing 4.7. Properties of new Policy Types ......................................................................................... 80
Listing 4.8. Properties of new Policy Types (cont’) ........................................................................... 81
Listing 4.9. Example properties of a Service Template ....................................................................... 82
List of Acronyms

ACCM  Accuracy Classification Case Memory
AGR  Adaptive Guided Retrieval
AHP  Analytical Hierarchy Process
AI  Artificial Intelligence
AWS  Amazon Web Services
CBR  Case-Based Reasoning
CP  Constraint Programming
CRN  Case Retrieval Net
CSAR  Cloud Service Archive
CSRS  Cloud Service Recommender System
D-HS  Discretized-Highest Similarity
FCA  Formal Concept Analysis
FD  Footprint Deletion
FPR  Footprint Retrieval
FUD  FootPrint Utility Deletion
GA  Genetic Algorithm
GE  Generalized Episode
IaaS  Infrastructure as a Service
IE  Information Entity
KBS  Knowledge-Based System
k-d tree  k-Dimensional tree
K-NN  K-Nearest Neighbor
LFF  Learning by Failure with Forgetting
LFF  Linear Fitness Function
MBR  Model-Based Reasoning
MCDA  Multi-Criteria Decision Analysis
MCDM  Multi-Criteria Decision Making
NACCM  Negative Accuracy Classification Case Memory
PaaS  Platform as a Service
QoS  Quality of Service
QuARAM  QoS-aware cloud application management
SaaS  Service as a Service
SLA  Service Level Agreement
SLO  Service Level Objective
SO  SortOut Case Memory
SOI  SortOut Internal Case Memory
SOM  SortOut Mean Case Memory
SOMI  SortOut Mean Internal Case Memory
TOPSIS  Technique for Order Preferences by Similarity to an Ideal Solution
TOSCA  Topology and Orchestration Specification of a Cloud Application
VM  Virtual Machine
Chapter 1

Introduction

Cloud computing, which is a paradigm for providing on-demand resources over the network is gaining popularity. It enables convenient, ubiquitous and on-demand access to a shared pool of virtualized resources (e.g., servers, storage, network, applications, and services). These configurable computing resources can be efficiently provisioned with minimal management effort and low service provider interactions [99]. The cloud computing paradigm is basically described by five essential characteristics: on-demand resource provisioning, broad network access, resource pooling, rapid elasticity, and measured service.

Many organizations have started to adopt the cloud as a way of augmenting, or even replacing, their existing IT infrastructure. As a result, they are considering moving their applications and data to a cloud environment in order to take advantage of its flexibility and potential cost savings [24]. A recent survey of major cloud vendors and customers revealed that over 80% of the respondents were considering moving critical applications to the cloud [106]. Elasticity, cost-efficiency, and on-demand resource provisioning are the primary motivations for this migration of enterprise applications to the cloud [173]. The survey also shows that security concerns, vendor lock-in and interoperability are major challenges that hinder the wide adoption of the cloud.

The current cloud computing landscape is both complex and constantly changing. It contains different types of clouds (private, public and hybrid), different usage models (Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS)) and different pricing models (rental, reservation and spot market). It is anticipated that the fluidity of, and the competition within, the cloud computing market will grow as the technology matures, which will force providers to adopt a wider range of mechanisms to attract consumers (e.g., sale prices and incentives). Cloud spot markets, for example, are a recent approach to marketing cloud resources [160]. They use a market-driven approach to resource pricing. The prices are
increased when resource demands are high and decreased when demand is low. This approach makes it appealing to consumers to shift their high demands into periods of lower prices. One might also expect that Quality-of-Service (QoS) requirements of the application execution, such as response time, throughput and availability, will be essential attributes of the Service Level Agreements (SLAs) that are negotiated between consumers and cloud providers. These complexities and changes introduce new demands on the consumer side for technologies that carry out effective evaluation of the provider’s offerings, SLA negotiation, and transparent migration of applications between different providers, if they are to take advantage of the evolving cloud market.

The increasing number of cloud providers’ offerings may offer functionally identical or similar services in the cloud but with different characteristics, which makes it difficult for potential consumers to assess and weigh which option best meets their requirements [150]. There are currently more than 90 IaaS [46] providers, each of them offers a set of various configurations of VMs, storage, and bandwidth in different regions of the world. A recent study by Statista [118] on the growth of the public cloud services industry in the period between 2011-2016, found that IaaS has the highest annual growth rate of 43%.

Cloud providers including Amazon Web Services (AWS), Microsoft Azure, Rackspace, GoGrid and others give customers the choice to deploy their applications over a pool of virtual services with practically no upfront investment and with an operating cost proportional to their actual usage [173]. Although this large pool of virtual services provides customers with the opportunity to focus on their core business, there are many factors and heterogeneous parameters that the customer needs to consider to choose a service. This makes it challenging to choose a suitable service that supports all the functional and non-functional requirements of an application at a low price. Having priorities over different parameters and requirements makes it even more challenging.

In this dissertation, we present a new self-adaptive method of cloud service selection that helps cloud customers in making their choice decision. It adjusts decision-making to changes in customer demands and in cloud providers’ offerings. The objective is to provide customers with a short list of potential services
that are best for their cloud application deployment. We first propose to capture the necessary application structure and requirements by extending Normative Types of TOSCA (Topology and Orchestration Specification for Cloud Applications) [107], an industry standard for specifying cloud applications, to capture the requirements of an application and facilitate effective application deployment. Second, we present techniques to explore and compare services and integrate them to achieve better deployments in the cloud. We also present a comprehensive framework (i.e., the QuARAM\textsuperscript{1} Service Recommendation framework) to demonstrate the usability of our work.

The remainder of this chapter presents the motivation behind our research, summarizes the thesis contributions and outlines the thesis document.

1.1 Motivation

The large number of available services (e.g., virtual machines or cloud servers in IaaS) with different characteristics, prices and features provides cloud customers with a wide range of options for their application deployment. However, there are many similar services from different cloud providers which make the decision on the best option challenging. Customers also typically have preferences over certain features, which pose an additional challenge on the selection algorithm. For example scalability, which means that the application can dynamically scale up and down according to varying demands, is an important feature for many cloud applications. Security and privacy are examples of other features that may be of priority for some applications (e.g., medical cloud applications). In most cases, price is a strong determining factor for cloud service selection and may, in fact, be the sole reason motivating customers to move to the cloud.

As an example, consider a simple application that needs to be deployed on the public cloud. The application has two components (i.e., Apache Web Server [7] and MySQL database Server [154]), which we will call deployment entities, which can be deployed on multiple service instances (e.g., multiple VM instances).

\textsuperscript{1} QoS- Aware Cloud Application Management
Figure 1.1 illustrates the minimum system requirements and QoS requirements of the different deployment entities of this application. The number to the right of each feature represents the customer’s respective preferences, ranging from 0 to 1, where 0 means “no preference” and 1 means “most required”. Given the application requirements, it can be deployed on two VM instances, one for the Apache webserver and one for the MySQL database server. There are more than 20 different IaaS providers who satisfy the requirements and can provision service (VMs in this case) to these two entities [33]. The price varies from $10 to $67 per month for each instance (e.g., Amazon EC2: m1.small+10GB EBS, GoGrid: small instance,
Rackspace: *Performance 1SSD 1G RAM* instance). Another option is to deploy the two entities on a single powerful instance (e.g., Amazon EC2: *m3.medium*+20GB EBS or Rackspace: *Performance 1SSD 2G RAM* instance). The choice of the appropriate service is not solely based on the service price, but also takes into consideration the service performance and the experience of previous tenants.

These challenges require careful analysis and efficient evaluation of available services, discovery of services that better satisfy customers and the integration of potential services. An automatic system that is capable of incorporating these aspects together helps customers to save resources and better focus on their business.

There are three challenges that must be resolved to realize this automatic decision making system:

1) How to describe cloud applications so that the application requirements and customer preferences are automatically identified and extracted. This is to reduce the interactions between the customer and the system and to entrust tasks to the system when possible. A standard specification for cloud applications helps to improve the portability and interoperability of cloud applications. It also assists in resolving the vendor lock-in problem [19, 107].

*Topology and Orchestration Specification for Cloud Applications (TOSCA)* is a standardization effort by OASIS that aims to describe composite applications and their management in a modular and portable format [107]. The defined *Normative Types* in TOSCA [107] contain some of the general features for an application specification. However, these types do not cover all application requirements (such as region, data transfer, operating system type and version, etc.) and the customer preferences. In order to allow customers to specify all of the requirements we need to extend the Normative Types to inclusively incorporate the application requirements and customer preferences in the TOSCA specification of the application.

2) Decision makers are not always experts in cloud services. Lacking the proper level of expertise in the field makes it difficult for them to select the best deployment plan for their applications on the cloud. On the other hand, providing them too many options including standalone and integrated
services makes it even more complex and confusing. The decision must balance between the application’s requirements and the customer’s goals, desires and constraints. Previous studies show that due to the large size of the search space of available services, and the wide range of heterogeneous selection criteria, there is a need for a robust approach that simplifies the cloud service selection. Such an approach may reduce the search complexity by either reducing the search space or the number of comparisons required to select the appropriate service [173]. It must support both qualitative and quantitative criteria.

Typically, there are multiple services (possibly from different providers) that can satisfy the requirements of an application. The service price can be a distinguishing factor but not the only one. There are other factors that can relax the price constraint in favor of better performance. In these cases, it is necessary to define the performance metric that best distinguishes between different services. This metric must be derived from the specifications of the application.

For example, assume we have a computationally-intensive application with the following requirements:

- **4 1.0 GHz 2007Xeon CPU power;**
- **1GB of memory;**
- **Maximum of 1000 concurrent users;**
- **1GB download and 10 GB upload bandwidth;**
- **Maximum cost of $100 per month.**

There are more than 20 services available on public clouds that can satisfy these requirements ranging in price from $10 to $84 per month [33]. Selection based just on price would choose the service with $10/month but taking performance into account it would be better to select a service with a higher price that provides the best performance for this type of application and has the lowest price among the suitable ones with a similar level of performance.

Applications can be deployed on multiple instances, where each instance serves a deployment
entity (e.g., Database servers or Web server) or on a single large instance. Large VMs are typically expensive. If the application could be distributed on different small VMs, it may lead to a better price. For instance, assume we have an application with the requirements of $32 \text{ 1.0GHz 2007 Xeon CPU power}$, $15.5 \text{ GB of memory}$, $1 \text{TB of storage}$, $1 \text{GB download and 10 GB upload bandwidth}$. A single service that can accommodate the entire application may cost at least $\$351$ per month [33]. If we distribute this application among different services, this price can be lowered to $\$137$ per month which is 40\% less than the first case.

Now, splitting the application into independent deployment entities presents an additional challenge, which is the service integration. The final decision on the regions where to deploy components, consolidation of different services to reduce communication costs between entities, increase resource utilization and lower the price, needs careful considerations. Therefore, services must be efficiently consolidated such that all the application requirements are satisfied, resources are efficiently utilized and the price is the lowest possible for the required performance.

Studies have addressed service selection in different type of cloud services (i.e., IaaS, PaaS, and SaaS) [4], [6], [11–18]. While they all are trying to solve the service selection problem, they have shortcomings in terms of complexity, scalability, overlooking customer preference, applicability for limited number of availabilities, missing values, and the restrictions to service selection for applications that can be deployed on single services. All the proposed approaches handle every query for service selection separately without any consideration of similar previous queries.

3) How the system responds to dynamically changing environments is also a major challenge. The service selection system must be self-adaptive and change its behavior according to changes in the customer’s interests or in the parameters of the execution environment. It must pay attention to customer satisfaction as it impacts the decisions of fellow customers who come later to use the system. For example, assume a service provider regularly violates its SLAs so that current customers are not satisfied with the support. The service selection system must incorporate the
feedback reported by customers as well as monitoring systems to provide better recommendations. Among the previously proposed service selection methods, CSRS [59] is the only method that incorporates the customers’ feedback in the process of service selection.

1.2 Contributions

Our research objective is to build an automatic and self-adaptive cloud service selection system that recommends a list of suitable services for cloud application deployment based on the application’s requirements and the customer’s preferences. The main contributions of this thesis are summarized as follows:

1. We extend the TOSCA Normative Types standards for cloud application specification to support a wider range of application requirements, customer preferences and SLAs. The extension facilitates the automatic extraction of the application requirements and customer priorities (Chapter 4).

2. We use case-based reasoning to assist customers in selecting a suitable deployment configuration for their applications on the cloud. Using this machine learning technique, the customer experience and the knowledge of the experts are incorporated into the decision making process. The system uses various machine learning techniques to improve selection performance over time. Our case-based service recommendation system provides dynamic adaptation of retrieved solutions based on runtime conditions and system/application requirements (Chapter 5).

3. We propose a service consolidation method to integrate different deployment entities of an application to improve the resource utilization and reduce the total service price (Chapter 6).

4. We present a comprehensive cloud service recommendation framework that automatically extracts the requirements of an application and recommends a suitable configuration of
services for cloud deployment. This framework encompasses all required functionalities ranging from automatic extraction of requirements and preferences to dynamic adaptation of retrieved solutions to consolidation of services (Chapter 7).

1.3 Thesis organization
The remainder of this thesis is organized as follows. Chapter 2 gives background information and a literature survey on cloud service selection, case-based reasoning and case-based recommendation systems. Chapter 3 presents the QuARAM (QoS-aware cloud application management) framework, the parent of the work presented in this thesis. The QuARAM framework receives the specification of an application, and generates a plan for the application deployment, which involves selecting the appropriate cloud provider(s) and allocating required resources on the provider(s) for the application. QuARAM deploys the application and provides the run-time management which includes monitoring the application to ensure that the SLA is satisfied. If a violation occurs, QuARAM may perform dynamic re-provisioning of resources within the cloud or even migration of the application to a new cloud provider.

Chapter 4 provides a brief background on TOSCA and discusses how to determine a list of possible application requirements and QoS from the specification of cloud services. We investigate the current Normative Types in TOSCA to determine what specifications we need to include and accordingly extend these Normative Types to incorporate additional requirements, customer preferences and QoS in the specification of an application.

Chapter 5 explains the use of case-based reasoning (CBR) in cloud service selection and how the proposed method makes adaptable decisions using the feedback from customers and monitoring systems. We prove the feasibility and usability of the proposed cloud service selection through a proof-of-concept prototype.

Chapter 6 presents our search-based service selection approach when case-based reasoning solutions are not applicable or do not meet the customer’s requirements. We use two different methods for service selection: a linear fitness function and the TOPSIS method [159] for ranking and selecting the most suitable
service for an application (or deployment entity). These two methods are compared to select the one best suited for our service selection. We also present a method to consolidate services in order to increase resource utilization while lowering the price.

Chapter 7 presents a comprehensive platform of the QuARAM Service Recommendation framework that includes all the functionalities of the methods proposed in Chapters 4 to 6. Using this framework, cloud customers enter the specification of their applications and receive a list of recommendations for the best services that the application can be deployed on. We present a proof-of-concept prototype to validate our framework.

Chapter 8 concludes our research and discusses the future directions.
Chapter 2

Background and Literature Review

In this chapter, we provide a literature survey on the proposed approaches for cloud service selection. Then we provide a thorough background on case-based reasoning and recommendation. Lastly, we discuss open research issues in the domain of cloud service selection.

2.1 Cloud Service Selection

Many studies have been carried out to address the selection of cloud providers since 2009. For example, CSRS [59] is a conceptual framework that compares all available cloud services based on the performance of virtual machines, QoS and customer feedback. Cloud providers must register their services with CSRS and the performance of each virtual machine (VM\textunderscore factor) is calculated using a set of benchmarks. The authors define the \textit{S-Rank} metric to compare between different services. S-Rank combines performance, QoS, and the feedback on the service and is calculated by:

\[
S - \text{Rank} = \alpha \cdot e^{VM\textunderscore factor} + \beta \cdot e^{QoS} + \gamma \cdot U_f
\]

(1)

where \(\alpha\), \(\beta\), and \(\gamma\) incorporate the preferences of the customer over different factors, \(\alpha + \beta + \gamma = 1\) and \(U_f\) is the customer feedback. For each query, S-Rank is calculated for all services that have the same specifications as the customer requirements. This framework selects services that satisfy the technical requirements of the application first and then eliminates those that do not accommodate the cost constraints of the customer. The selected set is ranked and presented to the customer.

Zang et al. [171] propose a two-step approach that uses a maximum gain and minimum cost to optimize the service selection. The proposed algorithm selects a list of available services first and then recommends the service based on the customer preferences (e.g., maximum gain or minimum cost).

Rehman et al. [124] introduce a mathematical formulation and method based on a set of abstract criteria for
selecting a cloud service provider. Their approach generates a matrix that contains the specification and performance of services. They form a customer query as a vector and accommodate the preferences of the customer over different criteria in another vector. Similarity measures in the literature (e.g., Pearson’s correlation) are used to select the most similar service to the customer’s query. As the authors indicate, this proposed method is only effective for service selection amongst offerings that have similar specifications but differ with respect to performance.

All these approaches entail high complexity as they compare all available cloud services against all criteria. As a result they do not scale well, which is a drawback since the number of services is increasing dramatically.

Chen et al. [29] use Constraint Programming (CP) to solve the problem of service selection in the cloud. Their proposed framework has three phases: Filter, Allocator, and Solver. In the Filter phase, the potential valid services are identified by removing services which violate an enterprise policy (e.g., price or location policies). The Allocator phase next finds all potentially valid services for every requirement of the customer amongst the Filter results. Finally the Solver phase derives appropriate solutions. It performs this selection by considering the domain constraints and interdependent relations (e.g., location of two VMs for the deployment of the application tier and data tier of a three-tier application). In this proposed method the solution is a set of services, where each of them serves one customer requirement (with respect to all the enterprise policies and the interdependent relations). Although the authors aim to reduce the number of comparisons, the complexity of the algorithm is still high when the numbers of constraints and services are large. The proposed approach does not accommodate the preferences of the customer over different requirements.

Another group of studies defined the service selection problem as a multi-criteria decision making (MCDM) problem [168]. Godse et al. [52] employ the Analytical Hierarchy Process (AHP) method to solve the SaaS cloud service selection problem. The drawback of this method is its limitation on the number of alternatives
with multiple criteria. The CloudRecommender [173] is a proposed cloud service selection system that defines the selection process as a multi-criteria optimization problem. The approach defines the service ontology and QoS to facilitate the discovery of services based on their functionality and QoS parameters. The system solves the problem with a Genetic Algorithm (GA) and uses AHP as the fitness function to handle mixed quantitative and qualitative criteria. Qian et al. [120] propose a heuristic approach to search the solution space. Their approach organizes the selection problem as a graph that encompasses the components of the target application and potential clients (clients of the application). Then, it finds a matching service for each node in the graph based on the service cost, distance to clients, distance to other components of the application, and the reliability of the provider. Sundareswaran et al. [150] tackle the problem of a large search space through indexing the cloud services using iDistance techniques based on a common set of properties. This approach first indexes cloud providers based on their properties in a B+ tree, and then uses the K-Nearest Neighbor (K-NN) algorithm [44] to find the service that fits the requirements of the customer. Since a cloud service customer may have a set of service requirements that cannot be fulfilled by any single service provider, they propose to integrate multiple service providers. The result is presented to the customer if it satisfies all the requirements, otherwise a new query is constructed to satisfy the remaining requirements. The search continues until all requirements are satisfied.

This category of approaches in the literature reduces the number of comparisons by either using heuristic algorithms or indexing the search space. Like all the other previous approaches, approaches in this category handle every query separately without taking into account similar previous queries.

2.2 Case-Based Reasoning

Artificial intelligence (AI) aims to solve complex problems in ways similar to humans. Although AI is still far from reaching human-level intelligence, there have been many successes in different areas such as computer science, medicine, finance, industry, transportation, communication, etc. [53].

The study of knowledge-based systems (KBS) (also known as expert systems) is one of the most successful
branches of AI research. In knowledge-based systems, a model of knowledge must be elicited and implemented, often in the form of rules or objects, irrespective of the depth of the domain knowledge that has to be covered [164]. Although model-based KBSs have been successful in many domains and applications[14–17], several obstacles remain. The major difficulty is the elicitation of knowledge, which is mainly due to the requirement that the expert knowledge must be in the form of rules. Experts do not typically think about their domain problems and solutions in this way. Other problems include the difficulty in KBS implementation for large scale systems, lack of memory, lack of robustness, and the difficulty of maintaining these systems [137, 163, 164]. The problems plaguing KBSs have been reduced by the emergence of case-based reasoning.

Case-based reasoning (CBR) is a problem solving methodology and a theory of reasoning that is based on the way humans think, reason, and solve problems in the real world. People tend to make decisions based on their direct or indirect experience. Similarly, CBR systems are intelligent systems that reason by first retrieving relevant prior cases from their memory of cases and then adapt solutions of these prior cases to solve the new problem. Aamodt and Plaza [2] propose a life cycle for case-based reasoning systems as shown in Figure 2.1. The four steps of CBR, called the four “RE”s, are retrieve, reuse, revise and retain [2, 72]. In the retrieve step, a new problem is compared to cases in the case base and one or more similar cases are retrieved. The solutions of the retrieved cases are reused for the new problem. The suitability of these solutions to the new case is evaluated and recorded. If a suggested solution does not satisfy the new problem, revision is carried out. The revised solution and its problem are retained in the case base for future use [13–15], [20].

There are a large number of applications that use the CBR approach for problem solving in areas involving design [13], [22–24], planning [33–37] and diagnosis [38–42] to interpretive tasks like understanding and justification [164]. Recently, several Web-based applications have been built that use CBR [43–47].
2.2.1 Knowledge Representation in CBR

In order to utilize the previous experiences in the CBR cycle, cases must be represented in a structural manner. Several methods of representation can be used in case base reasoning. The selection of the appropriate representation method depends on the domain that the system is modeling and the types of similarity assessments and retrieval, which are chosen according to the system [14, 137, 163].

The simplest format to represent the cases in the case base is to have simple feature-value vectors, which are good for cases with nominal or numeric attributes. This representation cannot capture the relationship between the individual attributes or cases. Surface similarity-based retrieval is used for finding matching cases from the case base [14, 137, 163].

A simple representation format, however, is insufficient for domains that have complex attributes or relationships between case attributes or between cases in the case base. In such situations, cases can be represented in the form of objects, predicates, semantic networks, scripts, frames, rules and concept maps[14, 137, 163]. Case bases also can be represented in the form of XML documents[35].
2.2.2 Case Memory Models (Case Base Organization)

The case base should be organized in a manageable structure that supports efficient search and retrieval methods. Case memory models can be grouped in three main categories: flat memory models, hierarchical memory models, and network-based memory models.

2.2.2.1 Flat Memory

In a flat memory model [17, 158], all the cases are organized at the same level. The retrieval time of this model is high since all cases in the case base must be compared to the target case for each retrieval request. Thus, this method is not recommended for large case bases [89]. However, this model provides maximum accuracy and easy retention, which justifies its use in many applications.

2.2.2.2 Hierarchical or Shared-Feature Network

This type of case base organization is attained when the cases that have the same or similar features are grouped together. The case memory in the hierarchical memory organization is a network that can consist of categories, semantic relations, cases and index pointers [89].

In general, in shared-feature network memory models it is difficult to maintain an optimal network as the case base expands [89]. A list of hierarchical memory models is as follows:

Three-layer model [174]: Figure 2.2 illustrates a three-layer structure of a case base. The first layer contains the feature-value pairs (features and the assigned values for them, e.g., color-red where color is the feature and the value is red). The second layer contains the problems and the third layer contains the solutions. Separating problems from solutions makes it possible for different problems to share a single solution and for a problem to have alternative solutions. The connections between the first and second layers show the feature-values for each problem. The second set of connections is between the problems and the potential candidate solution for the problems. The weights of the second set of connections represent how important
A potential candidate solution is for a problem. The restriction of the three-layer model is that it does not support complex and continuous attributes as it assumes simple nominal and discrete numeric attributes.

Dynamic memory model [2, 164]: This model (also called Generalized Episodes (GEs)) organizes cases that share similar properties under a more generalized structure (GE). Each GE contains three types of objects: norms, cases and index features that distinguish between cases belonging to the GE. Norms are common features to all cases under a GE. A new GE is created when a feature of a new case matches a feature in an existing case. Indexing the two cases under different indices discriminates these two cases below the generalized episode. Figure 2.3 shows the structure of the dynamic memory model. The disadvantage of this organization is the explosive growth in the number of indexes with an increased number of cases. A proposed solution is to limit the number of permissible indices to a limited vocabulary [164].
Hierarchy based on similarity between cases[167]: This organization is a k-d-tree (k-dimensional tree) that splits the case memory into groups of cases in such a way that each group contains cases that are similar to each other according to a given similarity measure. Figure 2.4 illustrates an example of a two dimensional search space and the corresponding k-d tree. This type of organization provides rapid retrieval; however, additions and deletions to the case base incur high maintenance costs due to the fact that the tree must be re-built for each update.
Formal concept lattice [42]: This organization is based on formal concept analysis (FCA) which provides a way to identify groupings of objects with shared properties. In this method, the cases in the case base are organized as a formal concept lattice using their attributes. With the help of this lattice a set of dependency rules is discovered that can aid retrieval. At the time of retrieval, the description of the target problem is used to find the similar cases in the lattice. Using this method, different cases can have different attributes. An incomplete definition of the target case can also be handled.

Decision tree induction based models [17]: Decision trees partition the case base around nodes composed of single attributes. In making these trees, how much an attribute can discriminate the cases is calculated (i.e., the information gain of the cases) and the attribute with highest discriminative power is located in the top of the tree. The calculation is repeated for the remaining attributes and the tree is made from top to bottom. This type of organization provides rapid retrieval, however, it does not handle complex attributes and suffers from complexity and high cost of maintenance [89].

Object-based model [15]: For representing complex domains, an object-oriented representation is an option. In this model, cases are represented as collection of objects and each object is described by a set of attribute-value pairs. The structure of objects is described by classes and they are arranged in a class hierarchy. There are two types of attributes for objects, simple types like integer or symbol, and relational attributes. The later type holds complete objects of a class. These types of attributes represent a directed
binary relation, like a part-of relation, between the object that defines the relational attribute and the object to which it refers. Complex cases can be represented with an object-oriented model.

Footprint memory model [144]: This memory model is based on competence. For construction of the model, the coverage and reachability of each case is calculated. Using these two measures, competence groups are formed. In each of the competence groups a sub-set of cases that covers all the cases in that group are selected as the footprint set of the group. Figure 2.5 illustrates the formation of competence groups and footprint sets in a case-base. The construction of this model is costly (O(n^2)) but the model’s scalability is good and it is efficient for large case-bases.

![Figure 2.5. Case base in Footprint model (adapted from [56]).](image)

2.2.2.3 Network Based Memory Models

Case retrieval nets [81]: The fundamental item in case retrieval nets is the information entity (IE). IEs represent any basic knowledge item, such as attribute-value pairs. A case consists of these IEs and the case base is a net with nodes for the IEs in the domain and additional nodes which are for cases. IE nodes may be connected by similarity arcs, and relevant arcs connect the case nodes to the IE nodes which make the case. Figure 2.6 illustrates an example of case retrieval net for travel agency. Construction of this organization is expensive, but these nets can handle partially specified queries. Using this kind of organization cases can have different attributes.
Category exemplar model [164]: The case memory in this model is a network structure of categories, semantic relations, cases and index pointers. This organization has three types of indices: feature links which point from problem features to a case or a category, case links that point from a category to its cases and difference links which point from categories to the neighbor cases where the differences to the current category are small. In this organization the categories are interlinked within a semantic network that represents a background of general domain knowledge which supports providing explanations for some CBR tasks.

Fish & shrink polyhedral [130]: This model is proposed for domains in which the similarities between different cases are calculated just according to certain aspects, and they can be considered dissimilar when other aspects are regarded for comparison in another query, so similarities are dynamic. In this model, each case is represented in a polyhedral form, and each face of a polyhedral corresponds to one of the aspects. The case base is a network of these cases in which edges between two cases show the similarities between them from different aspects. The label of each edge has a weight value that depends on the distance of the
connected cases with respect to a certain aspect. Figure 2.7 shows an example of this model.

Figure 2.7. Polyhedral cases and case base as a network of these cases (adapted from [58]).

Figure 2.8 illustrates the hierarchy of different memory models. In Table 2.1, the advantages and disadvantages of the different memory models are summarized.
Figure 2.8. Hierarchy of Memory models.
Table 2.1. Memory models, advantages and disadvantages.

<table>
<thead>
<tr>
<th>Memory model</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat Memory</td>
<td>- Easy retention</td>
<td>Slow retrieval (Not applicable for large case bases)</td>
</tr>
<tr>
<td></td>
<td>- Maximum competence</td>
<td></td>
</tr>
<tr>
<td>3-Layer</td>
<td>- Supports different problem with same solution</td>
<td>Does not support complex attributes</td>
</tr>
<tr>
<td></td>
<td>- Supports a problem with alternative solutions</td>
<td>Does not support attributes with infinite values</td>
</tr>
<tr>
<td></td>
<td>- Easy retention</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Supports nominal attributes</td>
<td></td>
</tr>
<tr>
<td>Dynamic memory model</td>
<td>- Fast retrieval</td>
<td>Explosive growth of number of indexes</td>
</tr>
<tr>
<td>k-d tree</td>
<td>- Rapid retrieval</td>
<td>Does not support incomplete problem description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- High maintenance cost</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Does not support non-ordered attributes</td>
</tr>
<tr>
<td>Formal concept lattice</td>
<td>- Fast retrieval</td>
<td>Supports just binary attributes</td>
</tr>
<tr>
<td></td>
<td>- Different cases can have different attributes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Can handle incomplete definition of target cases</td>
<td></td>
</tr>
<tr>
<td>Decision tree induction based models</td>
<td>- Rapid retrieval</td>
<td>High cost of maintenance</td>
</tr>
<tr>
<td></td>
<td>- Supports non-ordered and nominal attributes</td>
<td>High complexity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Does not handle incomplete definition of target cases</td>
</tr>
<tr>
<td>Case retrieval nets</td>
<td>- Handle incomplete definition of target cases</td>
<td>Costly construction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Benefits</td>
<td>Challenges</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Category exemplar model</td>
<td>- Different cases can have different attributes</td>
<td>- Complex structure</td>
</tr>
<tr>
<td></td>
<td>- Supports non-ordered attributes</td>
<td>- Need domain knowledge</td>
</tr>
<tr>
<td></td>
<td>- Supports complex attributes</td>
<td></td>
</tr>
<tr>
<td>Footprint model</td>
<td>- Fast retrieval</td>
<td>- Costly construction</td>
</tr>
<tr>
<td></td>
<td>- Handle incomplete description of target cases</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Scale well and efficient for large case bases</td>
<td></td>
</tr>
<tr>
<td>Fish and shrink</td>
<td>- Supports similarity from different aspects in different queries (dynamic similarity)</td>
<td>- Costly construction</td>
</tr>
<tr>
<td></td>
<td>- Useful for complex attributes</td>
<td>- Costly weighting of aspects between each two cases</td>
</tr>
<tr>
<td></td>
<td>- Fast retrieval</td>
<td>- Costly maintenance</td>
</tr>
<tr>
<td></td>
<td>- Supports large case bases</td>
<td></td>
</tr>
<tr>
<td>Object-based</td>
<td>- Fast retrieval</td>
<td>- Need domain knowledge</td>
</tr>
<tr>
<td></td>
<td>- Can handle incomplete description of target cases</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Supports complex attributes</td>
<td></td>
</tr>
<tr>
<td>Ontology-based</td>
<td>- Fast retrieval</td>
<td>- Need domain knowledge</td>
</tr>
<tr>
<td></td>
<td>- Supports complex attributes</td>
<td></td>
</tr>
</tbody>
</table>
2.2.3 CBR Cycle

In 1994, Aamodt and Plaza [2] proposed a life cycle for CBR systems which is used by other CBR researchers as a framework. This cycle consists of four main parts; retrieve, reuse, revise and retain. Each of these parts includes a set of tasks and different methods have been proposed for each of them. In this section we give an overview on the tasks and the methods.

2.2.3.1 Retrieval

An important step in case-based reasoning is the retrieval of previous cases that can be used to solve the target problem [158]. The input to the retrieval task is the problem description and the output is the cases that most closely match the new problem [2]. Among problem descriptors, more valuable features for retrieval have to be filtered. This filtering is done in a feature selection step. The cases in the case base are stored using these features and at the time of retrieval, these features are compared.

For retrieval of cases from the case base in order to solve the new case, different retrieval techniques have been proposed which are described in this section.

Feature selection

Like other feature-based systems, in case-based reasoning one area of research focus has been on how to select important features among all the features of the problem specification and assign weights to them to make the cases or to facilitate the retrieval. Different feature selection and weight assignment methods have been proposed in the literature. Guo et.al [55] propose using Rough Set theory for reducing the number of the features in cases. Hsu and Huang [62] extract the features by evaluating the relevance between features and classes with a fuzzy measurement. They evaluate feature correlation, data appearance and gain ratio of the features to decide on the most useful ones. This type of feature selection is useful in classification tasks. Another method proposed by Smyth and Kean [141] and Craw [37] uses Genetic algorithms for learning both the features and their weights. Other feature selection methods used in case-based reasoning applications, but not specifically for CBR, are statistical methods like Fisher’s

Retrieval techniques

Given a description of a problem, a retrieval algorithm, using the indices in the case-memory, should retrieve the cases most similar to the current problem or situation [163, 164]. Every retrieval method is a combination of a similarity assessment procedure, which determines the similarity between a target case and a case in the case base, and a procedure for searching the case memory to find the most similar cases [144]. Factors that play major roles in determining the performance of a CBR system are the complexity and the accuracy of the case retrieval phase [89]. A classification of the different retrieval methods in the literature based on the similarity assessment [87] is shown in Figure 2.9.
Figure 2.9. Hierarchy of retrieval methods.
Similarity based retrieval
In this group of retrieval methods, the target is compared to the cases in the case base and relevant cases are found according to the similarity in the features or the structure of the cases. The features that are compared could be the surface features that are provided as a part of the description of the cases, or derived features that can be obtained from the surface features and some inference from the domain knowledge [87].

Surface similarity based retrieval

i. Simple flat memory k-Nearest Neighbor retrieval (K-NN)

In this approach, the assessment of similarity is based on a weighted sum of features. Below is the typical equation for calculating the match between two cases:

$$\frac{\sum_{i=1}^{n} w_i \times Sim (f_i^l, f_i^R)}{\sum_{i=1}^{n} w_i}$$

(2)

where $w_i$ is the weight of feature $i$ and the $Sim (f_i^l, f_i^R)$ function returns the similarity between the value of feature $i$ of the input case and the retrieved case from the case base [164].

The problem with this method is the retrieval time, which is $O(n)$, where $n$ is the number of cases in the case base, so this method is not suitable for large case bases [164, 167]. However, the simplicity of this method leads to its use in other methods that reduce the size of the case base before starting retrieval. The main issue involved in searching in reduced case bases is the risk of missing the optimal cases since not every case is examined during retrieval [87].

ii. Induction methods

Induction methods are based on decision trees in machine learning (e.g., ID3 and C4.5 [101]). These algorithms determine the features that are best to discriminate between cases. Based on these features, a tree structure is formed to organize the cases in memory [164]. This group of methods cannot handle missing attribute values and are not suitable for case bases where the relevant importance of the individual
case features change [144]. For retrieval, features of the target case are compared with nodes in the tree, until it gets to one of the leaves that contain similar cases.

iii. K-d tree based retrieval

K-d tree is a method for partitioning the data source using hyperplanes. Every node in a k-d tree represents a subset of cases of the case base and the root of the tree contains all the cases. The partitioning attributes for building the tree are selected in a way that divides the case base into two equal size parts. Search for similar cases with a k-d tree is done using recursive tree search. The average costs for retrieving the most similar case for this method are $\log_2 n$, if the tree is optimally organized. This method cannot handle missing attribute values [167].

iv. Footprint retrieval

Figure 2.10. Two stage retrieval process in Footprint retrieval method (adapted from [56]).
In this method proposed by Smyth and McKenna [144], first a competence model of the case base is created. A set of cases which cover all the cases in the case base are selected and called footprint cases or reference set. The search for the best case is done in two steps; first in the footprint cases and then, after finding the best case in the reference set, in the subset of case base which this reference case covers. For finding the best match, this retrieval uses the nearest neighbor approach. Figure 2.10 illustrates the two-step retrieval in Footprint retrieval method.

In this method the number of cases that must be compared is similar to K-NN in flat memory while the quality (i.e., a function of distance between the target and the retrieved case) of the Footprint retrieval (FPR) is almost equal to K-NN. This method can handle missing attribute values.

v. Fish & shrink retrieval

This method of retrieval is used where the similarity of cases is dependent upon the aspect of interest. For instance, two cases could be considered similar with respect to a certain aspect, yet be considered dissimilar in another query based on a different aspect (Dynamic similarity). Also this method is useful where the cases have complex attributes. An example of using this method is in the FABEL project [49]. FABEL’s goal is to investigate integration of CBR with heuristic and model-based reasoning (MBR) for use in architectural design. In FABEL, a developed case-based reasoner offers different tools to retrieve cases, in which each tool retrieves under “certain conditions”. Each of these “certain conditions” are regarded as “aspects” and the fish and shrink algorithm is used as the retrieval method.

In the method proposed by Schaaf [130], cases are linked according to specific aspect similarities. It is based on the assumption that if a case does not fit a query, then its neighbors also do not match, thus, leading to the elimination of many cases during retrieval.

This method supports large case bases and since similarity is evaluated according to some aspects, missing attribute values are not an issue. As mentioned in the paper, calculation of the similarities between cases for different aspects (weights between each aspect of two cases) is costly, although it is done at the time of case base construction, and can be viewed as pre-processing cost.
vi. Validated retrieval

Simoudis and Miller [136] propose validated retrieval as a combination of simple retrieval with domain validation of the retrieved cases. In this method after a first stage of retrieval using simple retrievals (based on surface similarity based retrieval), further comparison is done between the target and the retrieved cases in order to reduce the final set of retrieved cases.

The problem with this retrieval method is the need for domain knowledge to build the validation model for the case base. Although the retrieved cases are the more accurate ones, the retrieval time for this method is more than just using the surface similarity based retrieval methods, rendering this method unacceptable for large scale case bases. Based on the retrieval method used for step one, it could be tolerant or not tolerant to missing values. This method is a good choice when the number of retrieved cases is more important than the retrieval time.

vii. D-HS (Discretized- Highest Similarity) retrieval methods

All the D-HS based methods use the cases in the case base as a training set to create a matrix where each cell $M(i,j)$ contains a list of cases whose normalized value $x$ for attribute $i$ lies in the $j$th interval of the attribute. Figure 2.11 has an example of this representation for a case base with 3 features.

![Figure 2.11. DH-S based method representation of the cases in the case base (adapted from [68]).](image)

For continuous attribute values, they discretize the values into intervals and for nominal attributes, the intervals are different values for that attribute. In retrieval, cases which have the most matching attribute
values to the target case are retrieved. In the D-HS method [112], using one of the difference calculation methods (e.g., Euclidean distance), the nearest cases among the retrieved ones in the first step are returned to the user. In D-HS-PSR (Discretised – Highest Similarity with Pattern Solution Reuse) [112] the retrieved sets are kept as a pattern tree in which each node has the attribute-value of one of the attributes and a pointer to the next attribute–value of the pattern. The leaves form the retrieval set. For large case-bases, there is a high probability that a number of different target cases have the same pattern tree.

Galushka and Patterson [51] focus on the issue of uneven distribution and propose D-HS which discretizes the attribute–values based on the entropy and the density of the values for each interval. Stephane et.al [147] focus on the retrieval of the cases in discretized case bases using a query sphere algorithm in which the neighborhood problem query consists of finding the relevant cases within a given distance from a given center location of the target problem. This method works better than previous methods for target problems that are near interval boundaries.

In general all the discretized retrieval methods work significantly faster than K-NN for large case-bases and they are domain independent [147]. They are also tolerant to missing values.

Structural similarity based retrieval

Although structural retrieval is computationally expensive because of the use of domain knowledge in formulating the structure, retrieval may find more relevant cases in comparison to similarity-based retrievals using surface features [87].

Several retrieval methods are proposed to retrieve relevant cases according to the structure of the case base:

i. Structural similarity as guidance [20]

One of the first works on structural assessment is by Borner, where retrieval is done in two steps. In the first step relevant cases are retrieved from the case base using surface similarity assessment and then using a transformation function defined by a domain expert, the system creates a structural format of the target case and searches for the cases with similar structures among the retrieved cases in previous step.
ii. Object-oriented based retrieval

One way to represent cases is in the form of objects where each of the attributes could be of a simple type, like integer or string, or of type object. This forms a hierarchy of the object structure within which cases in the same classes of the hierarchy can be compared. The issue with this type of structure is when the target case and the case in the case base are not objects of the same class [87]. Using this type of retrieval, not all the cases are compared to the target case, so it is faster than K-NN. Also this method is tolerant to missing attributes. If values are missing for the target case, the higher part of the hierarchy is searched, resulting in more retrieved cases.

iii. Spreading activation method

In this method, the case base is organized as an interconnected network of nodes which represents the case attribute-value combinations [144]. The spreading activation method is proposed by Lenz [81] and then customized by Aamodt [1]. The network representing the case base consists of feature-value nodes and case nodes which are interconnected to each other and the weight on each edge between nodes shows the relevance of two different nodes. For retrieval from this network, the features of the target case activate a set of the nodes in the network which in turn activate another set of nodes. If the activation has strength above a defined threshold, the activation spreads until some of the case nodes are activated. The strength of the activations depends on the weights that are assigned to the edges between nodes, which can be learned automatically, or can be assigned by the experts. The problem with this method is the cost of construction of the network and weighting of the edges in the network, which is difficult and time consuming. This method is efficient and flexible enough to handle incomplete case descriptions [87]. Parallel activation and spread of the activation signals in the network make the retrieval faster than k-NN.

iv. Retrieval of generalized cases

Generalized cases can be viewed as the implicit representation of a set of closely related point cases. Mougouï and Bergmann [103] define the similarity assessment as an optimization problem and do the
retrieval of the cases by ranking the general cases so there is no need to compare the target problem with all the cases. This work focuses only on attributes with real values and Taratakovski et al. [152] continue the work over mixed, discrete and continuous attributes. This approach is applied to a real world application [151]. The main issue with this method is that generating the index structure can be time consuming, however, this is only need to be done once [152].

v. Graph-oriented retrieval

Graphs are commonly used for representing complex domains like planning and design. These graphs could be attribute graphs, semantic nets or conceptual graphs [114]. Different approaches to retrieval from graph structures have been proposed [62, 87]. Petrovic et al. [114] propose a two stage retrieval that uses a heuristic search called Tabu search. First a simple Tabu search is used to rank all the cases in the case base by estimating the similarity degree. A subset of possible similar cases are then presented to an advanced Tabu search and the most similar cases are retrieved. The results of their experiments show that in contrast to previous approaches, this retrieval method works for large case bases in which the graph structure has several hundred vertices and there are several hundred cases in the case base. They used domain specific knowledge for a retrieval.

One of the forms used to represent graph structures is feature terms. Arcas and Mantaras [9] propose a method named “Prospective” which does the retrieval by matching a partial description of the target problem with the patterns in a lattice of feature terms, which represents the case base.

vi. Ontology-based retrieval

Ontologies can be used to form the case base where the cases are the instances of the ontology. Assali et al. [3] propose a similarity computation that has two components: a concept base similarity which is dependent upon the location of concepts in the ontology and a slot based similarity which calculates the similarity of two objects based on the common attributes between them. They define a notion of similarity regions, which is a sub-branch of the ontology where concepts and instances can be compared. This
eliminates the need to compare the target case with all the cases in the case base, therefore, making the method faster than K-NN retrieval.

Adaptive guided retrieval (AGR)
The effectiveness of a retrieval method is not just in finding the similar cases, but in identifying the useful cases [87]. In some applications, similar cases are not the ones that can be used in the reuse stage of the system, often because they are not adaptable for the target problem while certain cases with less similarity can be adapted so their solution can be used for the new problem. This issue in retrieval and its relation to reuse leads to studies on how to include the knowledge of adaptation in retrieval [139, 141]. In Adapted Guided Retrieval (AGR), at the time of retrieval, matches between the specification features of the target case and a case in the case base are constructed if adaptation knowledge shows that the match can be supported during the adaptation stage. Also the similar cases are ranked according to their overall adaptation cost. Experiments show that using this type of retrieval leads to less adaptation failure and less adaptation effort in the reuse stage [87, 141]. This method has higher retrieval cost than simple K-NN, but is tolerant of missing values.

Diversity conscious retrieval
In some systems like recommendation systems, the retrieval of cases deemed similar restricts the user’s choices. In these systems, diversity plays an important role in the satisfaction of the customers of the system. To address this issue, Diversity –Conscious retrieval has been proposed [22, 97, 102, 142]. In this group of retrieval methods, the problem is how to make the tradeoff between similarity and diversity. Some examples of the algorithms that try to make this trade-off are the greedy and bounded greedy algorithm [142]. Experiments show that the retrieval cost is the problem with these algorithms.

Compromise-driven retrieval [98] and coverage optimized retrieval [96]
In recommendation systems, the preferences of the user must be used at retrieval time. This is the reason that using K-NN cannot always return back the cases that satisfy the user. McSherry [96, 98] examined
how to increase the satisfaction of a recommender system’s user by defining some preference criteria and retrieval of the cases in the case base that have the coverage over all the cases that could satisfy the user. The work includes an assumption, called a compromised assumption, that is “if a given case C1 is more similar to the target query than C2, and differs from the target query in a subset of the attributes in which C2 differs from the target query, then C1 is more acceptable than C2.” Although this method of retrieval increases the retrieval time, the retrieved cases are more acceptable from the user’s point of view.

Explanation–oriented retrieval

Explanation-oriented retrieval explains how a question can discriminate between competing cases in recommendation systems. It can be used to explain the predicted outcome in classification and diagnosis systems, which can help in teaching the user about the domain. In planning, the explanation can be used to explain the plan failures in the system and to re-plan [57]. Some of the explanation based retrieval systems are reviewed elsewhere [12, 43, 57, 87]. These systems need domain knowledge in order to make an explanation model (e.g., explanation tree) for the retrieval.

Although for most of the retrieval methods the type of application and needs of that application are the criteria used to select a retrieval method, when more than one option for retrieval is available, the following aspects can help to decide [81, 87]:

- Efficiency of the method in both the speed and the efforts for searching in the case-base
- Quality of the solution with respect to measures like precision, recall and the overall length of the dialog with user, and how the method deals with the problems like noise, missing values or cases with different attributes.

2.2.3.2 Reuse

The second step in the case-based reasoning cycle is reuse. After finding similar cases to the target problem, the system needs to reason according to the retrieved cases to find a reasonable and accurate solution for the problem. The reuse of the solution can be done in two ways. One is just copying the solution of the
retrieved case as the solution for the target case (null adaptation) [164]. This is applicable to classification applications. However, in most applications, a retrieved solution cannot be used directly as the solution of the target case and some adaptation is necessary [2, 87].

**Adaptation**

Adaptation is particularly useful in constructive problem-solving tasks like design, planning and configuration. In these types of tasks, we do not have all the possible solutions in the case base, so by retrieving similar cases we find similar solutions and use the difference between the retrieved cases and the target case to modify the retrieved solution for the target problem [87].

Adaptation methods can be grouped as follows, according to how the changes on the retrieved solution can be achieved:

1. **Transformational / structural adaptation:** In this type of adaptation there exists domain-dependent knowledge in the form of a transformational operator \{T\} such that, applied to the old solution, it transforms this solution into a solution for the new case [2, 164]. Besides this knowledge, a control system is required to organize the operator [164]. Examples of this type of adaptation are parameter adjustment, abstraction and specialization, reinstantiation, model-based adaptation [87, 164] and the adaptation method proposed in the work by Fuchs et al. [50].

2. **Substitution adaptation:** In this type of adaptation, the values appropriate for the new target problem are substituted from values in the old solutions [72]. Reinstantiation is an example of this kind of adaptation [164]. Craw et.al [38] also propose a substitution adaptation in their work.

3. **Compositional adaptation:** In this type of adaptation, the adaptation takes parts of the solution from different cases that match corresponding parts of the user’s input problem requirements. This adaptation can be guided by rules that consist of preconditions that check for equivalent
parts of the problem description before copying parts of the desired solution [72]. An example of this method is proposed by Hanney and Keane [60] where the system searches for the rules that have all the differences between target and retrieved case. If it doesn’t find the matching rule, it tries to divide the differences into smaller parts and find matching rules for those parts.

4. Derivational adaptation: This type of adaptation, which is also called *derivational replay* or *generative adaptation*, looks at how the problems in the retrieved cases are solved. The cases hold the information about the method used for solving the retrieved problem including a justification of the operators used, sub-goals considered, alternatives generated, failed search path, etc. [2, 72, 87].

5. Special purpose adaptation and repair: This adaptation method is for domain-specific and structure-modification that is not covered by transformational and substitution methods [72].

Extracting adaptation knowledge is a complex research issue. One method of learning adaptation knowledge is to make a training set from the case base by leave-one-out testing [38]. When one of the cases is removed from the case base, the other cases in the case base can be used to find the solution for the removed problem. The adaptation knowledge is saved as a case in the adaptation case base. Figure 2.12 shows a framework for learning adaptation knowledge.

Aquín et al. [39], propose inductive learning to extract adaptation knowledge. In this method, a similarity path between the target and a retrieved case is calculated in the form of a similarity path and then for each step in the path, adaptation knowledge is extracted from domain knowledge and is kept in the adaptation knowledge base.

Finding adaptation rules by comparing each pair of cases and the differences in their solution is another method which is proposed by Li et al. [83]. Case base mining is also used to extract adaptation knowledge from a case base [40, 60].
2.2.3.3 Revision

After choosing to reuse a solution from the retrieved cases for a new problem, it may be discovered that this solution is, in fact, incorrect, thus providing an opportunity to learn from failure. In this phase, which is called revision, the case solution is evaluated and if the solution is incorrect, then domain specific knowledge is required to repair it [56]. CHEF [56] is one of the CBR systems that includes revision. In CHEF, causal knowledge is used to generate an explanation as to why the solution does not fit the goals of the system. These explanations are used to modify the solution [137].

Another method of correcting the proposed solution used in CASEY [158] and the proposed system by Portinale et al. [116], is to use a model-based system (i.e., a domain theory implemented with rules) that at the time of failure finds the correct solution for the problem. Model-based reasoning (MBR) should have a complete model of the system. This method is proposed for systems where complete information about the domain is accessible for creating the model for the domain. Using a case-based reasoning system renders the task faster and allows it to work online.

In general, revision can be viewed as two tasks: diagnosis of failure and solution repair [56]. For diagnosis,
one of the following ways can be used:

1) Execution of the solution and evaluating the outcome

2) Using a simulation model of the real world and evaluating the solution using the model. This solution is safer and more cost effective.

3) Experts also can help in diagnosing the failure in solutions. The expert evaluates the solution using his/her experiences.

4) Use the case base itself to identify the failure. In this case, to assist in problem diagnosis, in addition to the specification of the problem and the solution for each case, knowledge about the conditions under which the failure may occur must also be stored in the case base [146].

2.2.3.4 Retention

In a case-based reasoning system learning is done in the retention step. In this step, a new case is added to the case base according to some policies in the system. Retention includes adding knowledge and new cases to the case base, all of which needs to be indexed, as well as deleting cases from the case base in order to restrict its growth. Having new information about the cases in the case base and the knowledge system obtained in the previous steps in the cycle, indexing of the case base and other knowledge would be changed in this step.

The different retention (maintenance) strategies fall into one of two groups: maintenance of the content of the case base and maintenance of the organization of the case base. Research on maintaining the content involves work on the reduction of the case base and the deletion and addition policies. Maintenance of the organization is related to indexing the case base in order to make the case retrieval faster and more efficient.

Case base Size Reduction

The maintenance of the case base content is important for two main reasons: 1) to control the size of the case base and reduce the retrieval time. 2) To eliminate useless cases and irrelevant and redundant instances
that render the case base inconsistent in order to increase the accuracy of the CBR system [127].

In early knowledge based systems, there was a belief that more knowledge is a good thing. As knowledge based systems become more practical, it has become obvious that there is some “harmful” knowledge that can degrade the performance of the system. In case based reasoning, too many cases stored in the case base can lead to expensive searches. This is termed the *Swamping Problem*. As the number of cases in the case base grows, the expense for searching for similar cases increases [87, 140].

In early works on case base maintenance, the deletion policies were random deletion or selective deletion according to the performance benefits of the cases in the case base [140]. Smyth and Keane [140] define two concepts, coverage and reachability. Coverage of a case is the set of cases that it can be used to solve. Reachability of a case is the set of cases that can be used to solve the case. Based on these measures, they group the cases accordingly into four groups: Pivotal, Auxiliary, Spanning and Support. Smyth and Keane propose a competence model that is used in their deletion policy. First they propose the Footprint deletion (FD) policy where deletion is done according to the competence. The problem with this method is the possibility of preserving low utility cases while deleting high utility cases. To solve this problem, they propose Footprint Utility Deletion (FUD) where the decision to delete a case is based on both performance and competence. They next propose a case selection method based on the coverage of the cases in the case base [143]. Continuing the work of Smyth and Keane, Lu et al. [88] propose a new competence model and define steps to be taken after the deletion of a case to preserve the efficiency of the model for retrieval. Haouchine et al. [61] expand the deletion policy proposed by Smyth and Keane by defining two types of spanning cases (inter-spanning and intra-spanning). In their work, they keep all the Inter-class spanning cases and remove all the Intra-class spanning cases except the cases which have coverage less than a predefined threshold. Their experiments show positive results in terms of case base reduction, accuracy and competence.

Leake et al. [77] argue that case selection based on coverage as proposed by Smyth and Keane [143] is not a good criteria and as an alternative, they propose performance-based metrics for case selection. The
performance is calculated according to the adaptability of the cases in the case base. For each case they calculate a metric, called relative adaptation performance, which is the percent savings the case provides compared to the worst alternative case that solves the problem. Experimentally they prove that their method works better than previous methods, especially in non-uniform case distribution environments when some regions in the case base are used more than other regions.

Zhu and Yang [175] prove that using FD and FUD policies, a case base may suffer reduced competency after deletion. They propose an additional policy for case base reduction where a new case base is made from the original case base by selecting K cases with the highest coverage (K is the defined size for the new case base). The problem with this policy and FD and FUD is the time complexity of maintenance ($O(n^2)$) which is a high cost for the CBR system [75]. Another case reduction technique based on addition is JUST [110], which selects the cases from the original case base and adds them to the new case base using some justification criteria such as the size of the new case base and the minimum accuracy to terminate the addition of cases to the new case base. This system is just for classification tasks. Ni et al. [105] also propose an addition technique using outlier mining and a sieving strategy to formulate a new case base from the most valuable cases. The values of the cases are calculated using a goodness measure which is based on coverage of the cases. Their algorithm also has a complexity of $O(n^2)$ which is costly for the CBR system.

Lawanna and Daengdej [75] proposed a method called DRCBM which does the deletion in such a way that it maintains the maximum competence level of the case base. In their algorithm, they index on features that have maximal coverage and minimal reachability between cases. In their evaluation they compare their method with FD & FUD and case addition [175] and prove that their method is more efficient since it achieves a better case reduction rate with a finer competence reduction. Also they show that their method has a higher reduction rate in comparison to the other two methods.

Another work on case base retention is Adaptive Case-based reasoning [128]. In this system a case base is formed by retaining and forgetting cases. Different retention and forgetting strategies are used in the system and a measure called “goodness measure” is used to decide on the cases to be added or forgotten. The
goodness measure is calculated using reinforcement learning. When a retrieved case has the correct solution for the new problem the goodness of the case increases and when it has the wrong solution the goodness decreases. A simple calculation for goodness is shown to not affect the efficiency of the system and their system generates a more compact case base in comparison to other CBR maintenance methods.

The deletion policy used by Romdhane and Lamontagne [125] is based on the usage of the cases in case retrieval and the reinforcement value of each case. According to their experiments case usage is a good criterion on which to base the decision to delete cases, but reinforcement value only contributes when it is combined with other criteria.

Rough set is another method used in deletion policies. Salamo and Golobardes [126, 127] propose different deletion policies using the foundation of rough set theory. The proposed methods include Accuracy-Classification Case Memory (ACCM), Negative Accuracy-Classification Case Memory (NACCM), SortOut case memory (SO), SortOut Internal Case Memory (SOI), SortOut Mean Case Memory (SOM) and SortOut Mean Internal Case Memory (SOMI). They propose a new definition of coverage and reachability of the cases using rough set theory. ACCM keeps all the cases that are near the outliers and maintain all the internal cases in a way that covers all the internal cases. In NACCM, selecting cases starts from internal cases and then continues to outlier ones. SortOut case memory policies are based on grouping the cases in coverage groups and their difference is on the number of cases that have to be deleted from each coverage group. The case in the coverage group which has the maximum coverage is called the master case. In the SO method, for each of the coverage groups just a case with maximum coverage is kept. SOI deletes all the cases except master case if it can solve all the cases in the coverage group correctly, otherwise all the cases are kept in the case base. The next two methods (SOM and SOMI) try to make a new case base from the original one instead of deleting the cases from the case base. The reduction of cases obtained from these methods was not as large as with previous algorithms [110].

Neural networks and fuzzy logic techniques are also used in reducing the case base. In Shiu et.al [135] the system uses a neural network as a classification tool to divide a case base into various classes and each of
the cases has a fuzzy membership to each of the classes. After classifying the cases, the coverage of all the cases is computed and the cases with highest coverage are selected for addition to the new case base. The problem with this method, like other addition methods (making a new case base from the original case base) lies with adding a new case to the case base. After a period of using the system, if just the new case base is used (and new cases are not added to the case base) then it loses its efficiency. Also, rebuilding the case base is costly. How to overcome this problem is an open problem in maintenance. Yang and Wu [170] use the same clustering method, however, retrieval is based on the information gain of the features. The features are presented to the user, and based on the values the users assigned to each feature one of the clusters is selected and the cases contained in the cluster are returned to the user.

Harmful cases need to be eliminated from a case base. One type of harmful case are noise cases (cases that contain errors in values used to represent the case) which can decrease the efficiency of a CBR system by returning incorrect solutions for the target problem [92]. Another type of harmful case are boundary cases (especially for classification applications), that is, those cases located near the boundary of a class. In Massie et al. [92], a ratio is calculated which can provide the potential harmfulness of a case in the case base. The ratio gives an indicator of the positioning of a case in relation to the cases which have the same classes and the ones with different classes within the case’s local neighbourhood. According to this ratio, they find noisy and harmful cases. The deletion of the cases is based on a threshold specified for the ratio, and this threshold is domain specific. The proposed policy, Threshold Error Reduction, increases the accuracy in many applications.

Inconsistency in a case base is another reason for the need for maintenance of the case base. Racine and Yang [122] propose the use of a rule-based system for finding inconsistencies in the case base. The main problem with this method is the knowledge acquisition for the rule-based system.

Portinale et al. [115] propose a case memory management schema with the idea that when a case represents for learning (a new case is considered to be retained in a case base), the cases in the case base that cover the same portion of the problem space are consider to be replaced by the new one. In their later work [116]
they propose a failure-driven deletion method which is called learning by failure with forgetting (LFF). The main idea of their policy is to find the false positive cases during use of the case base and delete them. Also, their system distinguishes another group of cases which are old cases. In a specified time interval, the system detects the old cases and deletes them.

The following table summarized the different methods of case base size reduction, and their advantages and disadvantages.
<table>
<thead>
<tr>
<th>Method</th>
<th>Criteria</th>
<th>Pros and cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Footprint Deletion</td>
<td>Competence</td>
<td>High time complexity</td>
</tr>
<tr>
<td>Footprint Utility Deletion</td>
<td>Performance and Competence</td>
<td>High time complexity</td>
</tr>
<tr>
<td>Hauchine [61]</td>
<td>Competence</td>
<td>High time complexity</td>
</tr>
<tr>
<td>(expands smith and Keane work)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leake et al. [77]</td>
<td>Performance (performance according to adaptability)</td>
<td>Good for non-uniform case distribution</td>
</tr>
<tr>
<td>Zhu and Yang [175]</td>
<td>Competence</td>
<td>High time complexity</td>
</tr>
<tr>
<td>JUST [110]</td>
<td>• Size of new Case base</td>
<td>Only for classification tasks</td>
</tr>
<tr>
<td></td>
<td>• Minimum accuracy for termination</td>
<td></td>
</tr>
<tr>
<td>NI et al. [105]</td>
<td>• Outlier cases</td>
<td>High time complexity</td>
</tr>
<tr>
<td></td>
<td>• Goodness (coverage)</td>
<td></td>
</tr>
<tr>
<td>DRCBM [75]</td>
<td>Competence</td>
<td>Better reduction rate in comparison to FD &amp;FUD</td>
</tr>
<tr>
<td>Adaptive CBR [128]</td>
<td>Goodness measure (reinforcement learning)</td>
<td>• Low time complexity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• More compact CB with same efficiency</td>
</tr>
<tr>
<td>Salamo and Golobardes [126, 127]</td>
<td>Competence using rough set theory</td>
<td></td>
</tr>
<tr>
<td>(ACCM, NACCM, SO, SOI, SOIM)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shiu et.al [135]</td>
<td>Coverage (using fuzzy logic and neural networks)</td>
<td>• Problem of adding new cases</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Costly rebuilding case base</td>
</tr>
<tr>
<td>Threshold Error Reduction [92]</td>
<td>Harmfulness ration</td>
<td>• Elimination of harmful cases</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Only for classification applications</td>
</tr>
<tr>
<td>Recine and Yang [122]</td>
<td>Consistency (intra-case and inter – case)</td>
<td>• Elimination of inconsistent cases</td>
</tr>
<tr>
<td>Failure-driven deletion (LFF)</td>
<td>• Oldness of cases</td>
<td>• Elimination of harmful cases</td>
</tr>
<tr>
<td></td>
<td>• False positive cases</td>
<td></td>
</tr>
</tbody>
</table>
**Indexing**

A part of the retention step is indexing, and it constitutes one of the main issues for efficient retrieval of cases [89]. Indexing helps to reduce the search time and increases the efficiency of identifying a possible solution by means of making only a selective portion of the case-base available [111].

According to Watson [163, 164] indices should: 1) be predictive, 2) address the purpose for which the cases are used, 3) be abstract enough to allow for widening the future use of the case base and, 4) be concrete enough to be recognized in the future. We can find different types of indexing in the literature which can be categorized into the following groups:

1. *Difference-based techniques:* This type of indexing selects features that differentiate a case from other cases. A sample of using this technique is CYRUS [164].

2. *Inductive learning methods:* These methods, which are based on inductive learning in machine learning, identify predictive features and use them as indices. Different types of tree-base inductive learning methods fall into this group [135, 164]. Genetic algorithm-based methods and neural network based methods are also included in this group [135]. These methods require a certain number of training examples [12, 101] and they perform poorly when insufficient data is available. Another problem with this group of indexing techniques is that the learning phase is complicated because of the complex architecture which is used. Also maintenance in these techniques is difficult because after adding new cases or deleting some cases from the case base, the indices need to be recalculated [89].

3. *Explanation–based techniques:* These indexing methods try to identify the kind of situations in which the case can provide useful guidance and assign indexes to distinguished combination of features that, if seen together, describe a situation in which the case is useful. [12, 45]. These methods need a complete domain knowledge for indexing and explanations of the situations. Ontology-based techniques [161] can also be classified in this group. In these techniques, domain knowledge (the causal model) is required [12,
4. **Similarity-based generalization**: In these techniques, indices are created in two levels, one level is for the abstract cases, which are the cases that share some common features, to differentiate between different abstract cases, and unshared features are used as indices for original cases [71, 85, 89, 113, 144, 164].

5. **Discretized-based techniques**: These methods discretize the feature space and the indexing space is the discretized feature space [59–61], [107], [113].

6. **Dynamic indexing**: In this method, indexes will be made online at the time of a new input to the system. This dynamic indexing is based on the weighting of the features which is set by the domain expert. The level rank and the query for retrieval is made according to these indices [41]. Disadvantages of dynamic indexing are: 1) it requires specific domain knowledge from an expert. 2) dynamic indexing cannot cope with the addition of new cases to the case library. This is due to the fact that the risk level of the attribute–value has to be recalculated each time we want to add a new case [111].

7. **Bitwise indexing**: This method of indexing works on discrete features with finite values. According to feature-value pairs for each case, a bit string shows the case and the comparison are between these bit strings at the time of retrieval [30].

8. **Introspective indexing**: In this method, introspective learning is used to permit the CBR system to detect the features that are implicit in the original problem and not explicit in its initial indexing schema and to set them as indices in order to direct retrieval towards cases that can easily be adapted [48].
2.2.4 Case Based Recommendation

There are two groups of recommendation techniques, namely collaborative filtering and content-based techniques. Collaborative filtering approaches are based on the user-rating information and make suggestions for a target user based on the items that similar users liked in the past. It does not rely on any information about the items themselves. Content-based techniques rely on the descriptions of the items. Recommendations are generated from items that are similar to those the target user has liked in the past without relying on other users’ preferences[138].

Case-based recommenders are in the content-based class of recommender systems. In this group of recommendation systems, item descriptions are very important. Using these descriptions, the recommendation system generates a set of recommendations for a target user by retrieving items whose descriptions best match the user’s query[21][138].

Recommender systems are either reactive or proactive. In reactive recommenders, the user provides an explicit query and the system reacts with a recommendation response. In proactive recommender systems, the recommendations are made without the need for an explicit query[138],[21].

In terms of the dialog between the user and the recommender system, there are single-shot, and conversational recommenders. In single-shot recommender systems, the system returns a single set of suggestions to the user in a given session. In conversational recommenders, they adopt an iterative approach to recommendation. Users augment their requirements as part of an extended recommendation dialog [21, 138].

Conversational recommenders themselves can be grouped in two different classes. If the recommender ask users a series of questions regarding their requirements, it is called navigation-by-asking. If the system shows the users particular products and obtain requirements in the form of feedback on the proposed products it is called navigation-by-proposing [21, 138].
2.3 Open Research Issues

Based on the literature survey on cloud service selection methods we note some of the open research problems in this field:

- In all the approaches the requirements of the application are entered manually into the system. Automation of the requirements extraction could help to speed up the selection process and achieve more accurate results.

- Different users can have similar requirements or similar preferences over requirements. Previous contributions in IaaS service selection isolate each query and try to find suitable services for the query, regardless of previous deployments and queries. In some of these contributions, the only factor from previous deployments taken into consideration is the customers’ feedback on services. While this can help in better service selection, using previous queries and their similarities to the new one can provide a faster and more efficient service selection.

- Many contributions in IaaS service selection focus on applications or a set of requirements that can be satisfied with a single service. As regards multiple deployment entities for many medium and large size applications, it is required to select multiple services for the applications deployment. Some of the contributions in IaaS service selection, take account of this aspect of large applications, but the consolidation of selected services is not considered in these contributions. Service consolidation can help to yield a better price for application deployment while affecting the application performance in a positive way. IaaS service selection and consolidation and service management in federated cloud is still an ongoing open problem in cloud computing.

- Cloud customers are required to read the specifications of the cloud services in order to find the services that fulfill their requirements. Automatic service identification and representation is required in order to have a reliable service description.
3.1 Introduction

Cloud application management is the set of tools and processes that ensure the correct and efficient operation of applications in a cloud. It involves tasks such as provisioning cloud resources, deploying the application on the resources, monitoring the execution of the application, and dealing with performance problems and errors that may arise. In order to meet the demands of consumers in the dynamic and competitive cloud market of the future, we believe that cloud application management must evolve in several dimensions. First, it must become provider-independent and be able to work across multiple clouds. Second, it must become more autonomic so that it can quickly and automatically react to changes in the market, in the availability of resources, and in the application demand. Third, it must become QoS-aware and be able to take QoS requirements into account when making management decisions.

The process of deploying an application on the cloud should be simple and efficient. Unfortunately this is not necessarily the case today. Suppose, for example, a cloud customer wishes to launch the Web application introduced in Chapter 1 on a public cloud and offer it to clients over the Internet. A business analysis of the application has forecasted the number of clients, the level of service clients will expect and the price point the cloud customer will need from a cloud provider if they are to garner a profit from the application. The customer must go through the following steps to deploy and maintain the application:

- Choose the best cloud provider to host their application with respect to both service and cost. This can involve considerable research.
- Given a provider, provision the appropriate set of resources for the application, (i.e., choose the correct type of virtual machine (VM), the type and amount of storage required) and determine the most appropriate location of the VMs, if applicable. Customers must understand VMs and have
some level of understanding of the hardware needs of their application.

- Configure the cloud resources.
- Load data into the cloud storage.
- Create VM images including specific software required for the application.
- Initiate the application components.
- Monitor and manage the application.

Research on the topic of application provisioning and deployment tends to focus on different aspects of the problem including service description [19], resource provisioning [25], service deployment [69], and SLAs [70].

We propose a framework for autonomic QoS-aware cloud application management (QuARAM) [91]. It supports cloud application developers through the full or partial automation of the set of tasks described above and which are shown in Figure 3.1. QuARAM addresses all aspects of application provisioning and deployment in a unified QoS-aware infrastructure.

A vendor-independent description of an application is first provided to the system specifying the type of the application, the application components, the inter-connections among the components, and the Service Level Objectives (SLOs) for the application ({q1, q2, …, qn}). For example, for the Web application described in Chapter 1, a specification could state that a Load Balancer (LB) forwards requests to one of the several Application Servers, (AS), which access data via one or more Data Servers (DS). A Cache Server (CS) is used to improve data access performance. The QoS measures may include availability requirements, (e.g., the application must be available 99.9% of the time) and/or performance requirements (e.g., the average response time of requests must not exceed 5 seconds).

Our QuARAM framework generates a plan for the application deployment, which involves selecting the appropriate cloud provider(s) and determining a resource allocation on the provider(s) for the application. The execution of the plan entails provisioning the required resources on the cloud provider(s) and then deploying the application on those resources. The run-time management of the application includes
monitoring the application to ensure its SLAs are satisfied and, if not, possibly performing dynamic re-provisioning of resources within the cloud or even migration of the application to a new cloud provider.

![Application Management Tasks](image)

Figure 3.1. Application Management Tasks.

The remainder of the chapter is structured as follows. Section 2 outlines the requirements of a sample application. In Section 3 we present the details of the QuARAM framework. Lastly we summarize the chapter in Section 4.

### 3.2 Example Web Application

The Web application introduced in Chapter 1 is used throughout the remainder of this chapter to help illustrate how the QuARAM framework is used to deploy and manage cloud applications. It is assumed that the application is a three-tier architecture with multiple concurrent http requests handled by a load balancer that, in turn, delegates the requests to one of several application servers. The application’s servers query and update back-end data sources. Replication is used to scale the application when necessary.

The customer wishes to deploy this application, called “MyWebApplication”, on a public cloud and is searching for a provider that satisfies the following requirements:

- Supports a Linux Ubuntu platform;
- Supports MySQL;
- Provision resources for the application with maximum load of 1000 concurrent users;
- Provides 100% availability;
- Guarantees average response times of 5 seconds per Web interaction;
- Hosts the application for a total cost of less than $70 per month;
- Provides high security;
- Preferably in US-West region.

3.3 QuARAM Application Management Framework

Figure 3.2 shows the architecture of the QoS aware cloud application management (QuARAM) framework. It takes the requirements from the customer, recommends a provider (or set of providers) and a configuration to meet the customer’s requirements, provisions the appropriate resources provided by selected cloud provider(s), and deploys and manages the application.

The Deployment Engine coordinates the interaction between the customer, the Recommender and the Deployment Manager. The Recommender receives the application requirements and, recommends appropriate configurations based on similar past application deployments, if possible, or searches for services that can satisfy the requirements of the application. The Deployment Manager provisions the resources on the selected cloud provider(s) using the provided “effectors” (E in Figure 2) and deploys the application. Once deployed, the Runtime Manager monitors the application (data is provided via the “sensors” denoted by S in Figure 3.2), and, if necessary, the resource configuration is adapted to ensure that the application requirements are met.

Each of these components is described in detail in the following subsections with reference to the deployment of our sample application. For this deployment scenario, we assume that several multi-tier Web applications have been deployed by QuARAM in the past and that descriptions of the deployments of these applications are stored in the Deployment Repository. We also assume that several cloud providers are registered with QuARAM. Some of these providers are hosting applications deployed by QuARAM
and some are not. In this thesis, a part of Deployment engine component for extracting the application information from its specification (Chapter 4), and the recommender component for finding suitable configuration for application deployment (Chapters 5-7) is investigated and implemented.

![Diagram of the high level architecture of the proposed QuARAM framework.](image)

**Figure 3.2.** The high level architecture of the proposed QuARAM framework.

### 3.3.1 Deployment Engine

To deploy a new application, a customer submits the application specifications expressed as a TOSCA (Topology and Orchestration Specification of Cloud Application) Service Template (explain in Chapter 4) in to QuARAM. The Deployment Engine stores the TOSCA specification in the Application Template Repository and parses it to extract pertinent information used by the search engine to query the Recommender to find a suitable provider and configuration for the application. The configurations are adapted as necessary. The Deployment Engine communicates the Recommender’s suggestions to the customer, and formulates a plan for deployment.
Figure 3.3. Application Service Template for the Web application.
The TOSCA Service Template includes the application’s topology (i.e., what components need to be launched on what resources as well as the relationships between them), files required by the application, QoS requirements, and specifications for deployment, termination, and management of the service. The Service Template is stored in the Application Template Repository to be used during deployment.

Figure 3.3 shows a diagrammatic version of a TOSCA Service Template for the example application. The Topology shows the overall organization of our sample Web application including the application’s components and the relationships among them. A specification of the QoS requirements is also included for the application in defined policies in the application’s Service Template (explanation in Chapter 4).

When a recommendation is selected, the Deployment Engine creates a deployment plan for the application on the recommended cloud provider(s) using the recommended resource allocation. The deployment plan is a workflow constructed using the plans provided in the application’s TOSCA Service Template. An example of a plan is shown in Figure 3.3. The interactions with the cloud provider are specified through a uniform interface provided by the “effectors” assigned to each provider. The “effectors” in turn map the calls to the particular provider’s interface. The deployment plan is passed to the Deployment Manager for execution.

### 3.3.2 Recommender

The QuARAM Recommender helps the customers to determine an appropriate cloud provider and resource allocation for their application. It uses a combination of collaborative and content-based filtering [138]. It takes a description of an application and its requirements and supplies a ranked list of cloud providers and resource allocations for the application. Recommendations are based on previous deployments as well as requirements from the customer and QoS measures from the provider. The Recommender can be implemented using a rule-based system, however, systems of this type have two main disadvantages. First, rule-based systems require a large amounts of knowledge, usually from domain experts in order to build the system. Second, eliciting and modeling the required knowledge can be difficult and time consuming.
Case-based reasoning systems are built on knowledge that is collected as the system grows so they are simpler to implement and maintain than rule-based systems [137, 158]. The challenges with a CBR system are modeling the cases and designing the case base (CB) to support effective retrieval of cases.

The Deployment Repository provides the information used by the Recommender. It is composed of the Provider Knowledgebase and the Case Base. The Provider Knowledgebase (PKB) maintains information about each cloud provider registered with QuARAM such as the type and cost of offered resources, the offered services (e.g., monitoring, account management, Map/Reduce) and the supported features (e.g., location options and alternative pricing strategies). The Case Base (CB) stores the descriptions of previous application deployments. The features of a case include properties of the application and its deployment, the required QoS and the preferences of the application developer on the demands (e.g., the region is more important for the deployment than the price of application deployment). It also includes the “credit” of the case on further decisions based on the monitoring and feedback from customers regarding the services (as explain in Chapter 5).

![Figure 3.4. Finding deployment resources.](image-url)
The Recommender follows the process shown in Figure 3.4 to determine a provider and resource allocation for an application. Consider a customer called CustomerA. The customer is looking for a cloud service that guarantees a 99% general availability, a monitoring service, and the option of locating the application on sites in North America and Asia. The steps in the process are as follows:

**Step1**: A list of services and configurations is generated based on cases containing similar applications and requirements to those of CustomerA. The CB is searched and based on similarity measures, the cases deemed most similar are returned. More details on the retrieval process are given in Chapter 5. If the similarity of any of the retrieved cases is larger than a specified threshold then the process proceeds to step 2 otherwise it goes to step 2*.

**Step2**: While some of the retrieved configurations are similar to the customer’s application and preferences, it is highly unlikely that any are an exact match. Therefore, the selected cases are adapted to match the requirements. The adaptation knowledge consists of a set of rules that are applied to the feature values of a retrieved case to make a better fit with the new case requirements. For example, if the new application requires a faster response time than a similar retrieved case, and the gap is more than a given threshold, then the case could be adapted by upgrading to a larger VM type (e.g., from medium to large). These rules are manually defined by an expert.

**Step2**: A list of suitable cloud providers are selected from the PKB based on the providers’ features and the customer’s requirements. Given that this match is based solely on features offered by the provider, and not on previous customer experience, there is a chance for new providers who have not yet built up a deployment history with QuARAM to be selected given that their offerings can meet the customer’s requirements.

**Step 3**: The final configurations are returned to the Deployment Engine which presents the options to the customer who selects the provider(s) and a suitable configuration and approves the deployment. The selected configuration is stored in the case base for future use.
3.3.3 Simulator
The Simulator in QuARAM is used to provide estimates of how applications would perform on a cloud provider when there are no relevant cases in the Deployment Repository. The results are retained in Deployment Repository. The Simulator can be developed using an existing cloud simulation package such as CloudSim [26]. Generating new cases using the Simulator makes it possible for new providers and new services to be considered in future recommendation processes.

3.3.4 Deployment Manager
The Deployment Manager receives a deployment plan from the Deployment Engine along with the application’s Service Template from the Template Repository to provision the required resources on the selected cloud provider(s). Effectors (E in Figure 3.2) provide a provisioning interface for each cloud provider. Through this interface, virtual machines are launched and configured, storage space is set up, applications are deployed and activated on the virtual machines, and data is transferred to the cloud and replicated as required. Tools like CHEF [28], Puppet [119], and Ansible [5] are examples of many deployment tools which can handle system and application deployment on cloud.

In our sample deployment, virtual machines (running Linux) are provisioned on the selected provider to host the load balancer, the application server, the cache server and the database server. The data storage is provisioned and linked to the virtual machine hosting the database server. The application components (the selected Web server and application server as well as the MySQL database) are deployed, configured, and launched on the instances. The database schema (contained in the Service Template) is used to set up a MySQL database to host the application data and the database software is configured as specified by the customer.

3.3.5 Runtime Manager
Once an application is deployed, the Runtime Manager begins to monitor the application hosted by the cloud provider. The cloud Sensors (S in Figure 3.2), provide a uniform monitoring interface across
providers to the Runtime Manager. Sensors use the monitoring tools of the providers to collect performance data and emit a stream of events containing the performance measures [90]. When an application is deployed on a provider the Runtime Manager subscribes to the relevant sensors on that provider in order to monitor and evaluate the QoS received by the application. A possible way to implement the SLA monitoring is proposed by Moustafa [104].

The Analyzer component of the Runtime Manager analyzes the information collected from the Sensors to determine whether the specified service level objectives are being satisfied. If violations are detected, adjustments to the deployment can be made by the Adapt component of the Runtime Manager. For instance, if the QoS requirements of our example (e.g., 5 seconds response time) are not met, another application server may be launched. Modifications are made via the Effectors (E) provisioning interface. If minor adjustments are insufficient to satisfy the application requirements, or if there are significant external changes to the current cloud provider or competing providers, the application can be returned to the Deployment Engine to generate a new plan and trigger the redeployment of the application on new resources.

The Runtime Manager updates the Deployment Repository on related cases and providers with relevant statistics such as QoS information, for instance, the average QoS statistics, the frequency of QoS violations, the responsiveness of the cloud provider to violations, etc. This information is useful in assisting future customers in choosing a suitable provider.
3.4 Summary

The growing popularity of cloud computing means that many organizations are beginning to see clouds as an economical way of augmenting, or even replacing, their existing IT infrastructure. This trend contributes to the increasing demands of consumers for technologies that effectively evaluate different offerings, SLA negotiation, application deployment on the cloud, and efficient transparent migration of applications between providers.

Cloud application management is the set of tools and processes that ensure the correct and efficient operation of applications in a cloud. It involves tasks such as provisioning cloud resources, deploying the application on the resources, monitoring the execution of the application, and dealing with performance challenges and errors that may arise.

In this chapter we described the QuARAM framework for autonomic QoS-aware cloud application management. QuARAM facilitates selecting an appropriate cloud provider, provisioning resources on that provider, deploying the application on those resources and managing the execution of the application. QuARAM implements the autonomic MAPE (Monitor, Analyze, Plan and Execute) loop [34] to ensure continuous compliance with the application’s specified QoS requirements.

In the current thesis, the focus is on Deployment Engine and Recommender components of QuARAM framework. The Deployment Engine is responsible for parsing and extracting information from the specification of applications. In Chapter 4 we discuss this part of the Deployment Engine. The architecture and implementation of the Recommender component is discussed in Chapter 5 to 7.
Chapter 4

Requirements Identification and TOSCA Normative Type Extension

4.1 Introduction

A challenge in automating cloud service selection is how to specify cloud applications to easily identify and extract the requirements and customer preferences from the specification. Autonomic systems require low interactions between the customer and the system. TOSCA [107] (Topology and Orchestration Specification of Cloud Applications) is a standard specification of cloud applications that can help with integration of the application’s requirements (e.g., hardware requirements and QoS), customer preferences and characterizations of the application (e.g., application type) into the specification. Having this standard specification, brokers can identify and extract the requirements and characterizations automatically from the specification and select suitable services, provision the service instances, configure and deploy the application on the cloud.

To automate the extraction process, we need to identify a set of application requirements and then extend the TOSCA Normative Types to incorporate the requirements and preferences into the specification of the application. In this chapter we explain how we determine the requirements, determine which of them are covered by current definitions in TOSCA and how we extend the current defined specifications to cover all of application requirements, QoS requirements and preferences of the customer (i.e., application provider).

In first part of the chapter we focus on how to determine a list of possible application requirements and QoS from the specification of cloud services. In Section 3 we have an overview on TOSCA and the different elements used to specify an application. In Section 4 we explain the extension of TOSCA Normative Types to incorporate the proposed set of application requirements, QoS and the customer preferences into specification of the application. We explain how we extract these requirements and preferences automatically in Section 5. In Section 6 we summarize the chapter.
4.2 Requirements Determination

While most of public clouds have similar service offerings, their price and performance are the facets that lead customers to favor one over the other. Therefore, we explore the features that impact the service price of the 10 most popular IaaS providers. These top 10 providers are selected based on different blogs that compare various IaaS providers from different points of views [31, 153, 155]. For each provider, we explore the service specification features advertised on their websites. Table 4.1 shows the price and SLA features of these providers. We identified 12 functional and non-functional features (common features between IaaSs we examine) that have the most influence on the service price. We also identified 4 other features that impact the performance of an application on the cloud (e.g., SLOs). All these features are used in the recommender subsystem which is a service selection system based on case-based reasoning (Chapter 5).

We add Applications tiers (i.e., the number of tiers of the application, in multi-tier applications), Application type (I/O intensive, compute intensive, etc.), application servers (e.g. Tomcat, Jetty, Zend Server, etc.), and maximum number of concurrent users to our list of application features to be able to compare different applications in our recommender subsystem.

The maximum number of concurrent users is an important feature in service selection since based on this feature the size of a selected service may vary. For example, if for the application we mentioned in Chapter 1, the developer defines the maximum number of concurrent users as 2000 instead of 1000, the size of the services the recommender subsystem selects for the deployment of the application might be larger than the ones that are selected for the application with 1000 users.

In total, we determined 20 features to include in the TOSCA specification of an application as follows:

- Memory
- CPU power (Xeon 2007 1.0-1.2GHZ, EC2, etc.)
- Number of CPU cores
- Relational DBMS
- NoSQL datastore
- Storage
- Number of Instances
- Number of load balancers
- Region
- Security
- OS (Type, Platform, Version)
- Bandwidth (download, upload)
- Application type
- Application tiers
- Maximum number of concurrent users

- Application servers
- Availability
- Response time (range)
- I/O performance
- Maximum latency

Furthermore, the preferences on different features need to be included in the application’s deployment specification. An application may have restrictions or demands on some or all these features which we want to incorporate with the specification of the application. An application may need to deploy on more than one service instance on the cloud (depends on the size of the application or if the application needs multiple instances for reliability or performance reasons). For each of the application’s deployment entities, it is required to keep the system requirements (CPU, Memory, Storage, OS,…) and number of instances for that entity separately.

Furthermore, the preferences on different features (as is explained in Chapter 1) need to be included in the application’s specification.
Table 4.1. Top 10 IaaS providers and the price and SLA features.

<table>
<thead>
<tr>
<th>Name</th>
<th>Price features</th>
<th>SLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoGrid</td>
<td>- RAM (Memory)</td>
<td>- Availability</td>
</tr>
<tr>
<td></td>
<td>- Number of CPU cores</td>
<td>- Persistent storage</td>
</tr>
<tr>
<td></td>
<td>- Storage</td>
<td>- Internal /external network</td>
</tr>
<tr>
<td></td>
<td>- OS</td>
<td>- o Packet loss</td>
</tr>
<tr>
<td></td>
<td>- Bandwidth</td>
<td>- o Latency</td>
</tr>
<tr>
<td></td>
<td>- Block storage</td>
<td>- o Jitter</td>
</tr>
<tr>
<td></td>
<td>- Application server</td>
<td>- Cloud storage uptime and latency</td>
</tr>
<tr>
<td></td>
<td>- Relational DBMS</td>
<td>- Server reboot time</td>
</tr>
<tr>
<td></td>
<td>- Region</td>
<td></td>
</tr>
<tr>
<td>Amazon</td>
<td>- RAM (Memory)</td>
<td>- Availability</td>
</tr>
<tr>
<td></td>
<td>- Number of CPU cores</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Region</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Pay-as-you-go / prepaid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- OS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Bandwidth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Database server</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Elastic IP address</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Amazon cloudwatch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Elastic Loadbalancer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- EPS storage</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Cloud storage</td>
<td></td>
</tr>
<tr>
<td>Windows Azure</td>
<td>- Bandwidth</td>
<td>- Availability</td>
</tr>
<tr>
<td></td>
<td>- Region</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- RAM (Memory)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Computation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>o CPU Power</td>
<td></td>
</tr>
<tr>
<td></td>
<td>o Number of CPU cores</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- OS</td>
<td></td>
</tr>
<tr>
<td>Cloud Service</td>
<td>Features</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>RackSpace</td>
<td>SQL server (Relational DBMS) and size, Biztalk server, Bandwidth, Support, Pay-as-you-go/prepaid, Cloud storage, Backup, Memory, Number of CPU cores, OS, Storage, Pay-as-you-go/prepaid, Cloud monitoring, Load balancers, Cloud backup, Bandwidth, Database server, VM storage, Provider could management, Region, Availability</td>
<td></td>
</tr>
<tr>
<td>CloudSigma</td>
<td>Number of CPU cores, RAM (Memory), Storage, Bandwidth, Region, Availability, Network latency</td>
<td></td>
</tr>
<tr>
<td>IBM</td>
<td>Pay-as-you-go/prepaid, Number of CPU cores, Memory, Storage, Availability</td>
<td></td>
</tr>
<tr>
<td>Joyent</td>
<td>OS, Bandwidth, Availability</td>
<td></td>
</tr>
<tr>
<td>Cloud Provider</td>
<td>Features</td>
<td>Note</td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
<td>------</td>
</tr>
</tbody>
</table>
| Atlantic.net   | - RAM (Memory)  
                - Number of CPU cores  
                - Storage  
                - Bandwidth  
                - Pay-as-you-go/prepaid  
                - OS  
                - Monitoring  
                - Number of IP addresses  
                - Backup  
                - Region |
| eAPPs          | - OS  
                - RAM (Memory)  
                - Number of CPU cores  
                - Storage  
                - Region  
                - Availability |
| e24Cloud       | - Number of CPU cores  
                - RAM (Memory)  
                - Storage  
                - Bandwidth  
                - Region  
                - OS  
                - Number of IP addresses  
                - Backup |
4.3 Topology and Orchestration Specification for Cloud Application (TOSCA)

Topology and Orchestration Specification for Cloud Applications (TOSCA) is a standardization effort by OASIS that aims to describe composite applications and their management in a modular and portable format [107]. An application is defined using a Service Template. A Service Template contains a cloud service’s (i.e., application) topology and a set of plans [177]. A topology is a graph with nodes and relationships (e.g. an application is hosted on an application server, which is in turn hosted on an operating system). A plan is a workflow that describes the service’s operational aspects such as how to deploy, terminate, and manage this service. TOSCA enables interoperable descriptions of applications and cloud infrastructure services, the interrelationships between parts of the service, and the operational behavior of these services, independent of the supplier creating the service, and any particular cloud provider or hosting technology. It also supports a higher-level operational behavior to be associated with the cloud infrastructure management. Service Templates are interpreted by a TOSCA-compliant environment, which operates the cloud services and manages their instances [177].

4.3.1 Service Templates

A Service Template represents the cloud application in two parts: Topology Template which describes the structure of the application, and Plans which define the manageability behavior of the cloud application [107]. Figure 4.1 illustrates the schema of the Service Template.

The Topology Template of the cloud application describes the involved software and hardware (nodes) and the relationship between them. Each node and relationship has a type. These types offer management functionality, which is provided in Node Types and Relationship Types’ implementations. Concrete implementations (in form of scripts or WAR\(^1\) implementations) making them reusable.

Management Plans (which are typically modeled as BPMN\(^2\) or BPEL\(^3\) workflows), encapsulate knowledge

---

1 Web application ARchive [165]
2 Business Process Model and Notation [23]
3 Business Process Execution Language [178]
to deploy and manage the application on cloud.

The topology, management plans, and all the required artifacts (including installables, business logics, management logics) are encapsulated in the application’s package called a TOSCA CSAR (Cloud Service Archive) [73].

![Service Template](image)

Figure 4.1. Service Template schema (adapted from [107])

### 4.3.2 Node Types and Node Templates

Node Templates are descriptions of the ingredients of an application (components of a service). Node Types are reusable entities that each defines the type of one or more Node Templates. Node Types have interfaces that describe the operations made by the Node Templates for managing its instances (e.g. start and stop of the instance). The properties of the components of a service are defined as properties in the Node Type. Each Node Type can have a set of capabilities and requirements [82, 107].
4.3.3 **Requirements and Capabilities** [107]

Requirements and capabilities of components of a service can be expressed in TOSCA. For example to demonstrate that a component has certain requirements in preparation for deployment on the hosting environment (e.g. allocation of certain resources). They are also defined to express that one component requires a feature provided by another component of the service.

![Diagram of Requirements and Capabilities](image)

Figure 4.2. Requirements and Capabilities definition in TOSCA (adapted from [107]).

The requirements and capabilities are modeled as a part of Node Type. To make them reusable, Requirement Types and Capability Types are defined separately and so they can be used in the context of several Node Types. A Requirement Type is a reusable entity that expresses a kind of requirement that a Node Type can announce to expose (e.g. Database connection requirement). A Capability Type is a reusable entity that expresses a kind of capability that a Node Type can announce to expose (e.g. Database connection capability). Figure 4.2 illustrates the requirements and capabilities and how they defined in TOSCA.

4.3.4 **Relationship Type and Relationship Template**

A Relationship Template specifies the development of a relationship between nodes in a Topology Template. Each Relationship Template is relevant to a Relationship Type that defines the semantics and properties of the relationship. These types are defined separately to make them reusable.
Requirements and capabilities of Node Templates can be connected via Relationship Templates to show that the specific requirements of one node are delivered by a specific capability of another node [107]. Assume the example of the cloud application we introduced in Chapter I. The topology of the application consists of a PHP module, an Apache Webserver, a MySQL database, two operating systems (one for Webserver and one for the database server), and two virtual machines as illustrated in Figure 4.3. Three different Relationship Templates are used in this application’s topology template, hosted on, connects to, and depends on. The bold name in each box is the name of the Node Template and the names in the parenthesis below them are the Node Types for those Node Templates.

![Figure 4.3. MyWebapplication topology.](image)

### 4.3.5 Plans

Plans that are defined in a Service Template describe the management feature of the instances of the service (e.g. creation, and termination). They are defined as process models usually using BPEL or BPMN. A plan can contain tasks that refer to the operation of interfaces of Node Templates, operations of interfaces of Relationship Templates, or any other interface [107]. Such a task passes an input to an operation, which triggers actions on an instance of the Node Template or Relationship Template. On completion of the operation, the output is passed to the task. This data is available for further tasks of the plan [82].
4.3.6 Policy Types and Templates
Policies in TOSCA are used to define non-functional behavior or Quality-of-Service (e.g. monitoring behavior, payment conditions, scalability or continuous availability). A Node Type can be accompanied with a set of policies to express the non-functional behavior or QoS that each instance of the Node Template will introduce.

Each policy specifies the actual properties of the non-functional behavior of the individual instance of a Node Template. These properties are defined in Policy Types. The types can be defined in hierarchies to reflect the structure of non-functional behavior or QoS. Policy Types are reusable entities [107].

4.4 Extension of TOSCA Normative Types
According to TOSCA documentation V1.0 (November 2013) [107], there are two collections of Normative Types defined in TOSCA as basetypes and specificetypes. basetypes (Node Types, Requirement Types, Capability Types, Relationship Types and Artifact Types) define the base structure of the TOSCA type system and it is independent from any specific solution (e.g. “server”, “webserver”, “DBMS”, “host on”). specificetypes define types of specific solution (e.g. “Apachewebserver”, “MySQLDBMS”) and they are derived from the basetypes.

These defined Normative Types in the TOSCA contain some of the features (i.e., requirements and QoS) we need to incorporate in TOSCA specification of an application, but they are insufficient to capture all of the features and preferences we mentioned in Section 2. With the defined Normative Types in TOSCA we can incorporate Memory and vCPU for each of the servers. We can have the information on webserver and MySQL DBMS but other requirements, characterizations of the application and preferences of the customers can not be incorporated. Therefore, we propose an extension to the defined Normative Types to be able to encapsulate all the application requirements, characterizations and preferences of the customer. We incorporate the requirements information into TOSCA Service Templates by extending the defined Node Types, defining new Node Types, and new Policies. In order to do so, we first investigate the Node Types we need to extend
and the information we would include in each of them.

4.4.1 Extension of Existing Node Types

To integrate the application requirements into the Node Types we add the following properties to each of the TOSCA basetypes Node Types.

The first Node Type that we extend is the “server”. The defined properties for this Node Type are CPU and Memory. Along with these properties we add “NumberofInstances”, “IOPerformance”, “Data Transfer”, “Storage”, and “CPUPower”. Listing 4.1. Extended properties of Node Type "QuARAMServer". We named this new Node Type as “QuARAMServer”. The properties are defined as a complex type (i.e., “tQuARAMServerProperties”). Prefix t indicates that it is a type.

“OperatingSystem” Node Type is the next base type that we extended. The defined Node Type just includes the requirements and capabilities. In the original TOSCA definition it does not have the type or the version of the operating system as the properties of the Node Type. We extend this Node Type by adding “Type”, “Version” and “Platform” as new properties. Listing 4.2 illustrates the extended properties of the “operatingSystem” Node Type. We named the extended Node Type as “QuARAMOperatingSystem”. “tQuARAMOSProperties” is the complex type defined for the properties of the extended Node Type.
Listing 4.1. Extended properties of Node Type "QuARAMServer".

```xml
<xs:complexType name="QuARAMServerProperties">
  <xs:sequence>
    <xs:element name="NumberOfInstances">
      <xs:complexType>
        <xs:sequence>
          <xs:element default="1" name="numberOfInstances" type="xs:integer"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Weight" type="tns:tWeight"/>
  </xs:sequence>
  <xs:element name="CPUType">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="cpuType" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
    <xs:annotation>
      <xs:documentation xml:lang="en">Type of CPU</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="DataTransfer">
    <xs:complexType>
      <xs:sequence>
        <xs:element default="0" name="download" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
    <xs:annotation>
      <xs:documentation xml:lang="en">Download requirement for the server in GB</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="Storage">
    <xs:complexType>
      <xs:sequence>
        <xs:element default="0" name="storage" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
    <xs:annotation>
      <xs:documentation xml:lang="en">Storage for the server in GB</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="IOLatency">
    <xs:complexType>
      <xs:sequence>
        <xs:element default="Low" name="IOLatency" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
    <xs:annotation>
      <xs:documentation xml:lang="en">Required I/O performance</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:complexType>
```
The Node Type “DBMS” in `basetypes` contains the general requirements and capabilities of a DBMS. In original TOSCA definitions there is a collection of Node Types which are derived from the Node Types in `basetypes` to include specific properties and interfaces for those Node Types. This collection is called `specificTypes`. Definition of specific types of DBMSs (e.g. MySQL, Oracle, etc.) is in `specificTypes`. We extend the Node Type “DBMS” so it includes the type of the DBMS (relational or NoSQL) and the name of DBMS (e.g. MySQL, Oracle, Apache Cassandra, etc.) as its properties. The specific DBMSs then are
derived from this new Node Type (i.e., “QuARAMDBMS”). Listing 4.3 shows the definition of “QuARAMDBMS” properties.

Listing 4.3. Extended properties for Node Type "QuARAMDBMS".

```xml
<xsd:complexType name="tQuARAMDBMSProperties">
  <xsd:sequence>
    <xsd:element name="DBMSType">
      <xsd:annotation>
        <xsd:documentation xml:lang="en">Type of DBMS</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="DBMSName" type="xsd:string">
      <xsd:annotation>
        <xsd:documentation xml:lang="en">Name of DBMS</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="Weight" type="tns:tWeight"/>
  </xsd:sequence>
</xsd:complexType>
```

Listing 4.3. Extended properties for Node Type "QuARAMDBMS".

### 4.4.2 Definition of New Node Types

While some of the new applications’ requirements could be added to the application Service Template through extending the existing Node Types, other requirements had to be defined in a new Node Type. This new Node Type is “Loadbalancer”. Listing 4.4 shows the definition of this Node Type. We also propose a new “requirement” element for the “QuARAMServer” Node Type to interrelate servers and load balancers.

Listing 4.5 illustrates the definition of the properties of the new Node Type.

```xml
<NodeType name="QuARAMLoadBalancer">
  <PropertiesDefinition element="tns:QuARAMLoadBalancerProperties"/>
  <CapabilityDefinitions>
    <CapabilityDefinition
      capabilityType="Server-loadbalancerContainerCapability"
      lowerBound="0" name="server" upperBound="unbounded"/>
  </CapabilityDefinitions>
</NodeType>
```

Listing 4.4. QuARAMLoadBalancer Node Type definition.
4.4.3 New Policy Types

To specify QoS and non-functional requirements of an application in its TOSCA Service Template we need to define a new set of Policy Types. Therefore, we propose the following Policy Types: “ResponseTime”, “Security”, “Availability”, “Price”, “Region”, “Bandwidth”, and “Provider”. If the customer has any preferences towards a specific provider, the “provider” Policy Template captures these preferences in the application Service Template. The definitions of the Policy Types are shown in Listing 4.6. The properties of these Policy Types are illustrated in Listing 4.7 and 4.8.

Listing 4.5. Properties of New Node Type "QuARAMLoadBalancer".

Listing 4.6. New Policy Types.
Listing 4.7. Properties of new Policy Types.
4.4.4 Definition of New Properties for Service Templates

Applications have some properties which can help us in finding similar applications in our case-based recommender (Chapter 5). They are ApplicationType, Applicationtiers, and Maximumusers. These properties can be incorporated into the application Service Template as properties of the Service Template. These are xml fragments defined in <BoundaryDefinitions> of a Service Template. As an example the properties of a Service Template can be like the one in Listing 4.9.
4.4.5 Incorporating Customer Preferences into TOSCA Service Template

In addition to the properties and requirements of the application we also need to incorporate the preferences of customers on different requirements and properties into the Service Template. For example, a customer may have a higher preference for a specific region rather than the price. Therefore, we express the customer preferences related to each property through an associated weight that captures how important a property is to the customer (as is illustrated in Listing 1-8).

The Service Template of the “MyWebapplication” (Figure 4.3) application which used the extended types is illustrated in Appendix A.

4.5 Automatic Identification of Application’s Requirements and Preferences

We use the defined Node Types, Policy Types and the Service Template properties to incorporate the application’s new requirements, properties, customer preferences and required QoS into the application Service Template. These specifications can be automatically extracted by parsing the Service Template during the selection process. We use JAXB [117] to map the Node Templates into java objects that can be used in our recommendation system. These Java objects then are used in the recommender system to find the suitable services for the deployment of application with regard to all the requirements, application characterizations, required QoS and the preferences of the customer.

Listing 4.9. Example properties of a Service Template

```xml
<Properties>
  <ApplicationType>I/O Intensive</ApplicationType>
  <ApplicationTiers>3</ApplicationTiers>
  <MaximumUsers>1000</MaximumUsers>
</Properties>
```
4.7 Summary

Automatic extraction of cloud applications’ requirements and required QoS makes the service selection more effective. Cloud application providers can incorporate the preferred deployment requirements of their application into the application’s TOSCA Service Template. The Node Types, policies and properties in basetypes and specifictypes collections defined by OASIS cover some of the requirements (e.g., CPU, memory) but they can not incorporate all the deployment requirements (e.g., data transfer, I/O performance, number of instances, etc.), so we need to extend existing types or define new ones. To extend the reusability of these changes, we define new reusable and general Node Types with possible deployment requirements as their properties. We derived the proposed set of possible deployment properties by investigating the cloud service specifications advertised by various providers. We identified a set of features that have the most impact on the service price and performance or that characterize the application. We integrate user preferences by adding weights to the properties of Node Types that express the application’s requirements. In addition, we define new policies to incorporate QoS requirements into the application’s Service Template. The extraction of application requirements from the Service Templates is performed automatically using a general-purpose XML parser.
5.1 Introduction

Case-based reasoning (CBR) has been used in decision making systems in different areas since it was proposed in the 1980s. Cloud computing is no exception. CBR is used in different areas of cloud computing for automatic enactment and on-demand resource allocation [95], personalized recommendation service [63], management of autonomic services [16], and resource configuration and cloud infrastructure management [94, 100],[132].

CBR can be beneficial for cloud customers in choosing the cloud platform that best fits their application requirements, even when lacking complete knowledge about their application or features offered by cloud services. Considering all providers and their offerings, and the possible combinations of offerings, the search space for cloud customers is large and, hence, the decision making complex. Having a set of cases of previously deployed applications on the cloud and their requirements, customers’ preferences, and their feedback on the quality of the provided services, can reduce the search space.

CBR can efficiently handle heterogeneous attributes (i.e., quantitative and qualitative) that characterize cloud services and the requirements of cloud applications. It can also integrate customer’s preferences through assigning weights to these attributes. CBR has the ability to find matching services based on both system requirements, and the quality of the previous deployments. CBR also derives benefit from using the previous experiences and the knowledge of the experts reflected in the decisions. We use case-based reasoning for service selection to derive a benefit from its capabilities and strengths.

The remainder of this chapter is organized as follows. We explain the methods we use in our proposed case-based recommendation system in Section 2. Section 3 describes a prototype implementation of our proposed approach. In Section 4 we discuss the experiments we conducted using our prototype. Finally, we
summarize the chapter in Section 5.

5.2 Methodology

Previous studies show that there is a need for an efficient approach that can reduce the complexity of the search by reducing the search space or the number of comparisons required to select the appropriate service [173]. The approach should be able to handle a large number of attributes and support both qualitative and quantitative distinguishing criteria. It also should be able to adapt to a customer’s varying requirements, which can give rise to the problem of missing or unexpected values encountered in other approaches proposed in the literature [52, 120, 149, 150, 172].

Having many applications already deployed on the cloud, can help new cloud customers in making better decisions for their application deployment. There are many applications with similar requirements and features and even if the applications themselves are not similar, the deployment components they use (e.g., application servers, database servers, etc.) can be similar. Such similarity is valuable in finding an appropriate set of services and a configuration for new applications.

The information about the previous application deployments can be stored in knowledge bases and extracted when needed. CBR is a machine learning method that can efficiently utilize this knowledge to assist new customers in finding the appropriate services for their applications. CBR can search this knowledge base and find similar cases to the new case and retrieve potential solutions. The adaptation process in CBR systems provides the ability to adapt existing solutions to new problems presented to the system. New cases can then be incorporated into the case base, which, extends the effectiveness of the system. CBR can also handle the missing and unexpected inputs because it searches for similar cases instead of exact matches. Also since each case is a discrete, and independent knowledge unit, cases can be dynamically added or removed from the case base.

Case organization, similarity measures and adaptation methods are important aspects to the performance of a case-based reasoning system. While the organization of cases can affect the response time and complexity
of the system, similarity measures and adaptation impact the precision of the recommendations. In the following, we explore each of these aspects in detail and justify our choice on methods in each one.

5.2.1 Case organization

Organization of the cases in the case base has a direct effect on the complexity and response time of the case-based recommendation system. There are different case organizations (a.k.a. memory models) proposed in the literature. With the fact that the market of cloud services is rapidly growing, which implies a fast growth in the case base, we need a case organization that can support efficient retrieval from large case bases.

The flat memory model [18, 158] is the simplest one as it organizes all the cases in the same level. It is a good choice when the number of cases in the case base is relatively small, since during retrieval the CBR engine compares the problem case with each of the cases in the case base. Despite this disadvantage, this model provides maximum accuracy, easy maintenance and easy retention, which explains its wide use in many applications.

We adopt the flat memory model for our recommender. It has easy retention and maximum competence compared to other memory models. At the first glance, it seems that this model is not suitable for our cloud recommender because of the potential large case bases in our system. However, there are methods that distribute the case-based reasoning system (computational intensive jobs) among a set of VMs, which keeps the model simple while maintaining reasonable performance. An example of these methods is using Map-Reduce for case retrieval proposed by Hu et al. [63].

Each case in our case base is composed of a problem part and a solution part. The problem part includes the application requirements as described in Chapter 4. This includes: Application type, Memory, CPU power, Number of CPU cores (vCPU), Relational DBMS, NoSQL datastore, Application tiers, Application servers, Storage, Maximum number of concurrent users, Number of Instances, Number of load balancers, Region,
Security, OS (Type, Platform, Version), Bandwidth (Transfer In, Transfer Out), Availability, Response time (range), I/O performance, Maximum latency and Preferences. The solution part contains the cloud providers, the services (VMs), SLOs and configurations.

5.2.2 Similarity functions

Finding the best local similarity measure for attributes is domain-dependent and hard to generalize. Below, we define a list of similarity functions for the attributes in our case-based reasoning system. For each of the similarity functions we indicate, in parentheses, an example of the features to which this function applies. In these similarity functions $q$ represents an attribute of the target case and $c$ represents an attribute of the potential similar case in the case base. Both $c$ and $q$ could be a single attribute or of an array type. $\text{Sim}(c,q)$ is the similarity measure between the $q$ and the $c$ and the value is in the range of $[0,1]$. The list of defined functions is as follows:

1. Equal: For some attributes, for example OS, $q$ and $c$ are only similar if they have the same value (Windows and Linux OSs are not similar at all). For these attributes, we use Equal similarity function. This function returns 1 if the value of both attributes are equal, otherwise returns 0. The attributes can have numeric or string values. This function is used for attributes that similarity means identical values (e.g., OS).

\[
\text{Sim}(c,q) = \begin{cases} 1 & \text{if } c = q \\ 0 & \text{if } c \neq q \end{cases}
\]

(1)

2. Max String: Some attributes like region can be specified by the customer in general (e.g., US) or be more specific (e.g., US-West). For these type of attributes we use Max String function. This function returns the similarity of attributes of a string type. It returns a similarity value depending on the biggest substring that belongs to both strings (e.g., region, ApplicationType, DBMS, NoSQLstorage). This function yields the same results as the first function “1” if the
two strings are identical. Otherwise, it returns values between 0-1 where 0 means no match.

\[ Sim(c, q) = \frac{MaxSubString(c, q)}{MaxLength(c, q)} \]  

(2)

3. Interval: For attributes that can have a value in a range, the distance of \( q \) and \( c \) values can be the indicator of the similarity of them. We use *Interval* function for these attributes. The function returns the similarity of two numbers inside an interval (loadbalancer, Applicationtiers, computationalUnit).

\[ Sim(c, q) = 1 - \frac{|c - q|}{\text{interval}} \]  

(3)

Where *interval* is the length of the interval over which the values can be defined (e.g., the number of load balancers can be defined in the range [0, 10]).

4. Large Interval: The function finds similarity between two attributes when the range of acceptable values is large and the similarity across the total range is non-uniform mostly because of the properties of the underlying infrastructure (memory, vCPU, storage, bandwidth). For example, consider the memory attribute for VMs from a particular provider. Even though the numerical differences between the value pairs (600M, 1.5G) and (3.1G, 4G) are the same the similarities in the two cases are different if the move is from the lower value to the higher value. In the first pair, it requires switching to a larger VM type while the move in the second pair is accomplished within a single VM type. Therefore, this function splits the whole range of the attribute into subintervals (sInt) according to the specification of VMs, where each subinterval represents the range of the specific attribute in various VM types. Figure 5.1 illustrates an example of this interval and subintervals. In this Figure the attribute of the target case (\( q \)) and the attribute of the case in the case base (\( c \)) have values in two different subintervals. The VM labels (VM1, VM2, …) shows the specified memory for the VM.
If the two compared values belong to the same subinterval, the similarity is calculated as a regular interval similarity using the upper part of function (4), but within the subinterval to which \( c \) and \( q \) belong. When both \( q \) and \( c \) are in the same interval, it means that while they are not the same they can be covered by the same VM. So, the similarity of these two are calculated in that subinterval instead of the whole interval.

If they are in two different subintervals, they cannot be covered by the same VM but they still can be similar to some extent. We want to incorporate the difference of the subintervals of \( q \) and the \( c \) into the calculation of similarity. So, the difference between \( q \) and \( c \) is calculated in the whole defined interval for the attribute (e.g., the interval is 16000 (16GB) for the Memory as shown in Figure 5.1). Then to show the effect of being in two different subintervals into the similarity calculation, we divide the differences between \( q \) and \( c \) into the differences of the subintervals they belong to (as shown by the lower part of the function).

\[
Sim(c,q) = \begin{cases} 
1 - \left( \frac{|c - q|}{SInterval} \right), & sInt_c = sInt_q \\
1 - \left( \frac{|c - q|}{Interval} \right), & sInt_c \neq sInt_q 
\end{cases}
\]

(4)

where \( SInterval \) is the length of the subinterval to which \( c \) and \( q \) belong, \( sInt_c \) and \( sInt_q \) are the numbers assigned to the subintervals to which \( c \) and \( q \) belong, respectively. The \( Interval \) is the length of the interval in which the values can be defined.
5. Array Similarity: This function finds the similarity between the contents of two arrays of strings (e.g., `applicationServers`). The function compares every pair of strings (m,n) from q and c arrays. Strings are compared using the MaxString function (2). We then calculate the average of the similarities of the strings in q to strings in c. So, the overall string array similarities are aggregated and divided by the size of q. While this provides the average similarity of array q to array c, the size of the arrays also have an effect on the total similarity of q to c (e.g., the similarity of q and c when they both have 10 strings is different from when one has 5 and one has 10 strings). Therefore, we multiply the average similarity by the proportion of c and q.

\[
Sim(c, q) = \frac{\sum_{m \in c} \frac{\text{MaxSubString}(m,n)}{\text{minlength}(m,n)}, \forall m \in c}{\text{size}(q)} \times \frac{\min(\text{size}(c), \text{size}(q))}{\max(\text{size}(c), \text{size}(q))}
\] (5)

6. Constant asymmetric function: We use an asymmetric function for attributes where similarity is not equal for their lower and higher values (e.g., in price where lower values are more acceptable than higher ones) (e.g., `security`, `numberofInstances`, `availability`). The slope, coefficient k and the constant α show the tolerance to the changes of that attribute. The slope is the gradient of the line (function) where the values are less acceptable in terms of similarity. α can get values -1 and 1 which indicates whether the higher values are more acceptable (α=-1) or lower values (α=1). The coefficient k changes the slope of the side where the values are more acceptable in terms of similarity. Figure 5.2 illustrates the similarity function and the role of k and α in calculating the similarities. The coefficients k and α can be defined by the customer or a domain expert.
\[
Sim(c, q) = \begin{cases} 
0, & \alpha(c - q) \geq \frac{1}{\text{slope}^k} \\
\text{slope}^k \times (c - q) + 1, & 0 < \alpha(c - q) < \frac{1}{\text{slope}^k} \\
1, & c - q = 0 \\
\text{slope} \times (c - q) + 1, & -\frac{1}{\text{slope}} < \alpha(c - q) < 0
\end{cases}
\] (6)

If \( k=1 \), both side of the similarity function (lower values and higher values) behave the same and the function can be shortened to:

\[
Sim(c, q) = \begin{cases} 
\text{slope} \times (c - q) + 1, & |c - q| < \frac{1}{\text{slope}} \\
0, & \text{otherwise}
\end{cases}
\] (7)

7. Relative asymmetric function: This is similar to the previous similarity function except that the \textit{slope} is calculated based on the \( q \) value (e.g., maxConcurrentUsers, maxLatency, responseTime, price). The \( Rvalue \) is the percentage of the tolerance of that attribute. For example, if the price of \( c \) is 20% more than the price of \( q \), then the similarity is 0. We use a relative asymmetric function for calculating the similarity of the prices. Figure 5.3 provides a graphical representation of this function. As in function (6), \( \alpha \) indicates whether lower or higher values are

![Graphical representation of the relative asymmetric function](image-url)

Figure 5.2. Constant asymmetric function.

a. \( \alpha=1 \): cases with higher values are more acceptable

b. \( \alpha=-1 \): cases with lower values are more acceptable

7. Relative asymmetric function: This is similar to the previous similarity function except that the \textit{slope} is calculated based on the \( q \) value (e.g., maxConcurrentUsers, maxLatency, responseTime, price). The \textit{Rvalue} is the percentage of the tolerance of that attribute. For example, if the price of \( c \) is 20% more than the price of \( q \), then the similarity is 0. We use a relative asymmetric function for calculating the similarity of the prices. Figure 5.3 provides a graphical representation of this function. As in function (6), \( \alpha \) indicates whether lower or higher values are
more acceptable and \( k \) represents the tolerance to the more acceptable side of the function.

\[
\text{slope} = \frac{1}{R\text{value} \times q} \tag{8}
\]

Using the \( \text{slope} \), \( k \), and \( \alpha \), the similarity function is the same as the constant asymmetric function (6).

\[
\text{Sim}(c, q) = \begin{cases} 
0, & \text{for } 0 \leq \alpha(c - q) \leq (R\text{value} \times q)^k \\
\frac{1}{R\text{value} \times q} \times (c - q) + 1, & \text{for } 0 < \alpha(c - q) < (R\text{value} \times q)^k \\
1, & \text{for } c - q = 0 \\
\frac{1}{R\text{value} \times q} \times (c - q) + 1, & \text{for } -(R\text{value} \times q) < (c - q) < 0
\end{cases} \tag{9}
\]

Figure 5.3. Relative asymmetric function.

5.2.3 Maintaining the customer’s priorities

Including the priorities of the customer is also an important aspect of the recommendation system. There are two different aspects of priorities that we maintain in our recommendation system. The first one is the priorities of a new customer for different features of available services (e.g., computation is more important than memory). We tackle this issue by assigning weights to different attributes to reflect their importance in the global similarity function. The global similarity function combines the computed similarities of all
the attributes into a numeric value. A weighted sum of attributes [164] (function (10)) is used as the global similarity function in our case-based recommender.

\[ SIM (Case, Target) = \frac{\sum_{i=1}^{n} w_i \times Sim (a_i^e, a_i^t)}{\sum_{i=1}^{n} w_i} \]  \hspace{1cm} (10)

where \( w_i \) is the weight of feature \( i \) and the \( Sim (a_i^e, a_i^t) \) function returns the similarity between the value of feature \( i \) of the input case \( (a_i^t) \) and the retrieved case from case base \( (a_i^e) \). \( n \) is the number of attributes.

The second aspect is to maintain the priorities for previously deployed applications, so that the similarity between priorities also is used in calculating the overall similarity between the target case and the cases in the case base. To address this aspect we add an attribute ("priority") to the case metadata in the case base. This attribute holds the normalized values for the priorities of that application over requirements. The similarity function for this new attribute is as follows:

\[ Sim(c, q) = \frac{\sum_{i=1}^{n} Sim (c_i, q_i)}{n} \]  \hspace{1cm} (11)

Where \( Sim (c_i, q_i) \) is the similarity of the "priority" for \( i^{th} \) requirement and is calculated using the Interval function described in function (3).

5.2.4 Adaptation

Adaptation is an important aspect of CBR systems to adjust the solution of a similar previous case so that it is a better fit to the query.

A case-based reasoning system is responsible for adapting the retrieved solution to the requirements of a new application using the adaptation knowledge.

There are many adaptation methods proposed in the literature. We believe that substitution adaptation [72] fits our proposed recommendation system. In this type of adaptation, the appropriate values for the new target's solution are substituted from values in the old solution (i.e., retrieved from the case base). Selection of this adaptation method for our system is based on the observation that the solution of the retrieved case typically partially covers the requirements of the target problem. Changes to the attribute values of the
retrieved solution renders the solution to be more accommodating to all the requirements of the new application (e.g., selecting a larger or smaller VM, changing the region of a VM, or changing the provider). These changes are applied as actions on the retrieved solution (e.g., upgrade or downgrade a VM type).

The major challenge in adaptation is how to collect adaptation knowledge. The adaptation knowledge can be provided by a domain expert or can be extracted from the case base itself, as described in Chapter 2. We adopt the method proposed by Craw et al. [38] to extract the adaptation knowledge. This knowledge is learned using the cases in the case base. This method learns the knowledge using the “leave-one-out” cross validation approach [10]. In this approach a single case from the case base is used as the validation data and the remaining cases in the case base as training data. This is repeated until all the cases in the case base are used once as a validation data. The extracted adaptation knowledge is in the form of cases and is stored in a new case base (Adaptation case base). Figure 5.4 illustrates the steps of this method.

5.2.5 Maintenance and Reinforcement

The last aspect of the case-based recommendation system is to make a decision whether to retain new cases and for how long. As described in Chapter 2 the maintenance of the case base content is important for two main reasons: 1) to control the size of the case base to minimize the retrieval time; 2) to remove useless, irrelevant, and redundant cases that may make the case base inconsistent [127]. There are many maintenance approaches for CBR systems in the literature. The selection of certain cases to be removed could be based on performance [143], coverage and reachability of cases [76], consistency of cases [122], age and number of false positive retrieval of cases (i.e., the number of times that a case is incorrectly retrieved as a solution) [115, 116], harmfulness of the cases [92], or goodness of cases (reinforcement learning) [128]. Methods that make a decision based on the competency of cases in the case base have a high time complexity, while methods that adopt reinforcement learning have a low complexity and do not negatively impact the system efficiency. Also, these methods generate a more compact case base in comparison to other methods. Therefore, we adopt reinforcement learning in our system to enhance the
recommendation precision and improve the overall system performance.

This adaptation is based on the feedback from both the customer and monitoring systems on previous deployments or suggested services.

![Diagram of steps in implementation of learning adaptation knowledge method]

Figure 5.4. Steps of our implementation of the learning adaptation knowledge method.

5.2.6 Architecture of Cloud Service Recommender

Figure 5.5 illustrates the architecture of our proposed QuARAMRecommender case-based reasoning system (QuARAMRecommender CBR).

The customer submits the application requirements, constraints and preferences (a.k.a. target case) to QuARAMRecommender CBR through a user interface (UI). QuARAMRecommender CBR queries the
CBR engine with these attributes to retrieve similar cases from the *Application Case Base (CB)*. The engine then sends the retrieved cases and the submitted case to the *Adapter* where the retrieved solution(s) are adapted to the requirements of the customer’s application. The Adapter performs adaptation using the *Adaptation Case Base (CBA)* and the knowledge about available services (*Providers’ KnowledgeBase*). The customer selects one of the adapted solutions and the system retains the selected solution in the target case for future reference. The *Monitoring* service is used in our framework to update the knowledge in the CBA and CB.

![Figure 5.5. QuARAMRecommender CBR architecture.](image)

### 5.2.6.1 Knowledge Bases

As illustrated in our proposed architecture, QuARAMRecommender CBR has three different knowledge bases: *Application Case Base (CB)*, *Adaptation Case Base (CBA)* and *service Providers KnowledgeBase (PKB)*.
A. Application Case Base
Our system maintains a case base that contains cases of applications and their associated solution (i.e., deployment information). This case base includes the information of application requirements and the customer preferences as the problem part and the provider, VMs, configuration and the SLA as the solution part of the case.

B. Adaptation Case Base
This case base contains the knowledge about the adaptation of retrieved solutions. The problem part of the cases in this case base consists of all the application description attributes, the difference values for each of the attributes and a retrieved solution for that application description. The solution part of adaptation cases contains the action that must be performed on the retrieved solution [38]. More details on this knowledgebase construction is provided in subsection 5.3.4.

C. Providers KnowledgeBase
This knowledge base maintains a list of VM instances offered by different providers. It also contains the knowledge related to possible transitions from one type of instance to another for each provider.

The Application and Adaptation case bases are regularly updated. Adding new cases to the CB triggers the update of the Adaptation case base. New target cases can be solved using recently retained cases in the CB. So, the adaptation knowledge must be updated to encompass the recently retained cases (using the method we explained earlier in 5.2.4). Monitoring data is also another trigger for updating both the CB and CBA. If the monitoring system detects a SLA violation for a currently deployed application, the monitoring data is used to update the CB so the case related to that application loses its credit in future retrieval references. This SLA violation means that the selected solution for the deployed application is not a good solution. So the case which represents this deployed application is not a good case for future decisions. The update decreases the credit of the case in future retrievals. Following this update in the Application case base, the Adaptation Case Base is automatically updated to reflect the changes. The same process is performed if the customer provides unsatisfactory feedback about the received service.
5.3 Proof-of-Concept Implementation

We developed a prototype of our proposed QuARAMRecommender CBR to demonstrate the feasibility and usability of the system. This sub section describes the implementation details.

In this prototype, customers enter the requirements of their application and their preferences through a Web form to find appropriate IaaS cloud services (a representative set of requirements is provided in Table 5.1). In our implementation, we assume that the system responds with an appropriate VM that meets the application’s requirements.

We use Jcolibri 2.0 [65] to implement the prototype. Jcolibri is a Java-based object oriented framework that provides a platform for developing CBR applications. It is extensible and reusable across different domains and CBR families.

We adopt the flat memory model implemented in JColibri for our prototype due to the accuracy and simplicity of its construction and maintenance. We use the K-Nearest Neighbor (K-NN) as the retrieval method (as explained in Chapter 2) and a relational DMBS to construct our case bases and providers’ knowledgebase. Case bases in our prototype are relatively small and just for evaluation of our proposed approach. For large case bases, the knowledge bases can be constructed using Hbase [6] and relational database layer on top of a NoSQL storage such as Apache Phoenix [8], which can connect the case-based reasoning engine to the knowledge bases. A distributed case-based reasoning method like the one proposed in Betterlife [63] then be used.

The QuARAMRecommender CBR matches the list of requirements with similar cases in the case base and returns the potential provider and the appropriate VM type.

5.3.1 Data

In machine learning systems, in general, a set of data is required to initially train the system. Data are required for the proper functioning of our system whether for testing or running. For our proof of concept, we collected 260 cases from different available AMIs (Amazon Machine Images). Each AMI consists of one server (deployment entity). Some of the required attributes for these AMIs were missing, so we had to
fill in missing attributes to fit in our system, such as the maximum number of concurrent users, region, price for associated VM, etc.

### 5.3.2 Knowledge bases

The MySQL database engine [154] is used to create and maintain the knowledge bases. We briefly explain the content of our knowledge bases in this section.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Type</td>
<td>String</td>
<td>{I/O intensive, CPU intensive, …}</td>
</tr>
<tr>
<td>Memory</td>
<td>Integer</td>
<td>$n \geq 0$</td>
</tr>
<tr>
<td>CPU power</td>
<td>Integer</td>
<td>enum</td>
</tr>
<tr>
<td>Number of CPU cores</td>
<td>Integer</td>
<td>$n \geq 0$</td>
</tr>
<tr>
<td>OS</td>
<td>Set of three strings (Type, Platform, Version)</td>
<td>{windows, 64, 7.0}, {Linux, Ubuntu}, …</td>
</tr>
<tr>
<td>DBMS</td>
<td>String</td>
<td>{MySQL, DB2, …}</td>
</tr>
<tr>
<td>NoSQL datastore</td>
<td>String</td>
<td>{Casandra, Hadoop/HBase,…}</td>
</tr>
<tr>
<td>Application tiers</td>
<td>Integer</td>
<td>$n \geq 1$</td>
</tr>
<tr>
<td>Application servers</td>
<td>Array of strings</td>
<td>{Apache Tomcat, Jetty, Zend, …}</td>
</tr>
<tr>
<td>Storage</td>
<td>Integer</td>
<td>$n \geq 0$</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>Set of two integers</td>
<td>{[1, 10], [40,20], …}</td>
</tr>
<tr>
<td>Maximum number of users</td>
<td>Integer</td>
<td>$n \geq 0$</td>
</tr>
<tr>
<td>Number of instances</td>
<td>Integer</td>
<td>$n \geq 1$</td>
</tr>
<tr>
<td>Number of load balancers</td>
<td>Integer</td>
<td>$n \geq 0$</td>
</tr>
<tr>
<td>Region</td>
<td>String</td>
<td>{US East, Asia, …}</td>
</tr>
<tr>
<td>Response time</td>
<td>Set of two integers</td>
<td>{[20, 40], [10, 30], …}</td>
</tr>
<tr>
<td>Security</td>
<td>Integer</td>
<td>[1-10]</td>
</tr>
<tr>
<td>Availability</td>
<td>Integer</td>
<td>[0-100]</td>
</tr>
<tr>
<td>I/O performance</td>
<td>Enumerated type</td>
<td>{low, medium, high}</td>
</tr>
<tr>
<td>Maximum Latency</td>
<td>Integer</td>
<td>$n \geq 0$</td>
</tr>
<tr>
<td>price</td>
<td>Integer</td>
<td>$n \geq 0$</td>
</tr>
<tr>
<td>Priority</td>
<td>Array of doubles</td>
<td>$\forall n \in array, 1 \leq n \leq 10$</td>
</tr>
</tbody>
</table>

### A. Application Case Base

The attributes that constitute the problem part of the cases in our prototype are illustrated in Table 5.1. We explained in Chapter 4 how we decided on the features that affect the service selection. We used the same features as the attributes of a case’s problem part. We explain the features in more detail in Appendix B.
Solutions are represented by the suitable VM(s) for the application.

B. Adaptation Case Base
The adaptation actions in our prototype are “upgrade” and “downgrade” of the retrieved VM and the levels of the transition, where one level means selecting the next higher (or lower) VM. For example, the upgrade of an \textit{m1.small} instance one level is \textit{m1.medium} and two levels is \textit{m1.large}.

The extracted adaptation knowledge is in the form of cases and is stored in the CBA. We construct our CBA with 2036 cases.

C. Providers Knowledge Base
Providers Knowledge Base contains the transitions from one type of instance to another. Figure 5.6 shows an example of how the system can upgrade from a certain Amazon EC2 instance to the next higher level (in general and with no preferences over different features of the instances). The downgrade transition can be inferred from the graph. This knowledge can be modified to reflect the relation between the various instances regarding other features such as memory, I/O performance, CPU power, etc.

![Upgrade transition graph for Amazon EC2 instances.](image)

\textbf{5.3.3 Similarity Measures}
To determine the most similar case (s) to the target, the attributes of the target case are compared with the attributes of the stored cases using local similarity measures and a global similarity function that calculates
the aggregate similarity scores. For compound features like OS and Bandwidth (i.e., have sub-features) the
similarity function returns the average value of all sub-features. Table 5.2 illustrates the similarity functions
we use for the various attributes of our cases in the prototype.

Table 5.2. Similarity functions and QuARAMRecommender CBR case attributes.

<table>
<thead>
<tr>
<th>Function</th>
<th>Application attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max String</td>
<td>Application type, DBMS, NoSQL storage, Region</td>
</tr>
<tr>
<td>Interval</td>
<td>ComputationalUnit, LoadBalancer, Application tiers</td>
</tr>
<tr>
<td>Equal</td>
<td>OS (type, platform, version)</td>
</tr>
<tr>
<td>Large Interval</td>
<td>Memory, Storage, Number of Cores, Bandwidth</td>
</tr>
<tr>
<td>Array Similarity</td>
<td>Application servers</td>
</tr>
<tr>
<td>Constant asymmetric function</td>
<td>Security, Number of instances, Availability</td>
</tr>
<tr>
<td>Relative asymmetric function</td>
<td>Max concurrent users, Max latency, Response time, Price</td>
</tr>
</tbody>
</table>

In calculation of total similarity of the target case to the case from the case base, some of the features need
to be there or the cases similarity is zero. For example, if the OS of the case in the case base is not equal to
the required OS by the target case, then the similarity of the cases is set to zero.

To reflect the significance of the various attributes to the overall similarity of a case, we assign weights for
each attribute. These weights could be determined based on the importance of the attribute or according to
the user preferences. It is also possible to determine these weights based on the system history using
machine learning techniques. The default weight value for each of the attributes is 1. To show the impact
of weights on the system performance, we test the prototype with equal and weighted attributes. We assign
the weights to various features based on inputs from a domain expert.

5.3.4 Adaptation

Figure 5.7 illustrates the architecture of the Adaptor component in our proposed system. The adaption works
as follows. After QuARAMRecommender CBR retrieves a similar case \( r \) for the target case \( P_t \), they are given to the Adapter. The adaptation engine makes a query \( P_A \) using \( P_t \) and \( r \) and sends it to the Adaptation CBR. The similar case \( r \) encompasses both the problem part \( P_r \) and solution part \( S_r \). The query \( P_A \) represents the requirements of the application \( P_t \), differences between the problem parts \( D_{t,r} \) and the retrieved solution part \( S_r \). Equation 12 illustrates the procedure of forming the adaptation query.

\[
D_{t,r} = \{ att_{i,t} - att_{i,r} : att_{i,t} \in P_t, att_{i,r} \in P_r, att_{i,t} \neq null \}
\]

\[
P_A = P_t + D_{t,r} + S_r
\]  

(12)

The Adaptation CBR returns the action \( A \) that must be applied on \( S_r \). In our prototype, these actions are either upgrade or downgrade transitions. It also indicates how many levels the transition must be applied (e.g., upgrade for 2 levels means that the retrieved VM must be upgraded two configurations). The Adaptation engine applies the action using the knowledge in the Providers KnowledgeBase (PKB) and sends back the recommended solution \( S_t \). \( att_{i,t} \) and \( att_{i,r} \) represent the value for \( i \)th attribute of target case and retrieved case, respectively.

![Figure 5.7. Adaptor component in QuARAMRecommender CBR.](image-url)
As an example, assume our target case is the example described earlier in Chapter 1. Assume that the retrieved case has the following values for the problem part:

- OS: Linux
- DBMS: MySQL
- Region: US
- Maximum number of users: 1200
- Memory: 2GB
- CPU: 8 vCPU

And the solution for this case is \textit{m1.2xLarge}. The Adapter receives the retrieved case and the target case. $D_{t,r}$ is calculated by the Adaptation engine as set containing the differences between query and the retrieve case for each of the problem attributes. The adaptation query ($P_A$) is made using the target case, the differences and the retrieved case solution (i.e., \textit{m1.2xlarge}). Using this adaptation query, Adaptation CBR returns the action “\textit{downgrade}” to the Adaptation engine. Executing the action on the retrieved solution provides \textit{m1.xLarge} as the recommended solution.

### 5.4 Validation of CB Recommendation

We conduct experiments using our prototype to validate our case-based recommendation approach. A set of cases is generated based on the AMIs and the VMs on which they can be deployed. We validate the CBR method using the “\textit{leave-one-out}” cross-validation technique [10].

Each experiment focuses on a different aspect of our approach, namely the suitability of CBR for cloud service selection, the impact of adaptation on the recommendations, and the impact of reinforcement learning on the recommendations. The results are evaluated based on the precision of the recommended solutions. Precision (P) here is defined by Equation (13).

\[
P = \frac{t}{T},
\]

where $t$ is the number of cases for which the recommender suggests the correct solution and $T$ is the total
number of cases in the test set.

We query the QuRAMRecommender CBR with the left-out case. The retrieved solution is “most relevant” when it is similar to the original solution part of the left-out case. If the system recommends a larger type of VM than actually required (just one level larger, e.g., recommending *m1.Large* rather than *m1.medium*), then we say that the retrieved solution is “overestimated”.

The first experiment evaluates the suitability of case-based reasoning as a method for recommending IaaS services for cloud applications. It also examines the effect of involving weights (priorities of the requirements) in decisions. In our experiments we assume that the weights are assigned based on an expert’s opinion on the importance of attributes. There are other approaches that can be used to find weights for case features. An approach is learning feature weights from the cases in the case base [36][54]. Another method is to learn it from users’ feedback [68].

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Weight</th>
<th>Attribute</th>
<th>Weight</th>
<th>Attribute</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Type</td>
<td>0.1</td>
<td>Application tiers</td>
<td>0.1</td>
<td>Number of load balancers</td>
<td>0.1</td>
</tr>
<tr>
<td>Memory</td>
<td>0.5</td>
<td>Application servers</td>
<td>0.1</td>
<td>Region</td>
<td>0.3</td>
</tr>
<tr>
<td>CPU power</td>
<td>0.5</td>
<td>Storage</td>
<td>0.5</td>
<td>Response time</td>
<td>0.5</td>
</tr>
<tr>
<td>Number of CPU cores</td>
<td>0.5</td>
<td>Bandwidth</td>
<td>0.5</td>
<td>Security</td>
<td>0.5</td>
</tr>
<tr>
<td>OS</td>
<td>0.3</td>
<td>Maximum number of users</td>
<td>0.9</td>
<td>Availability</td>
<td>0.3</td>
</tr>
<tr>
<td>DBMS</td>
<td>0.1</td>
<td>number of instances</td>
<td>0.3</td>
<td>I/O performance</td>
<td>0.3</td>
</tr>
<tr>
<td>Price</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We first study the performance of the proposed system without applying weights (i.e., all attributes are equally important) and then study the significance of attributes by applying different weights that reflect their relative importance to the case. Values in the range of [0, 1] for these weights are used for comparison and stored in the case base for retention. An example of assigned weights is shown in Table 5.3.

Figure 5.8 shows the precision of both retrieved solutions (most relevant and overestimated) for equal and weighted attributes. Results reveal that the system achieves a precision of 32% with equally weighted attributes and 57.5% for varying weights. The weights present 26% improvements to the overall precision. We observed, however, that this precision is too low to be practical.
We then study the impact of adaptation on the overall precision of recommendations. Figure 5.9 shows the results of using adaptation in the system. The total precision has improved to 71% with both “most relevant” and “overestimated” solutions.

Maintenance of the case bases and adapting knowledge based on the feedback boosts further the performance of the system. We conduct another experiment to evaluate the effect of the selected maintenance method. Every case in the case base is assigned an attribute named “credit”. To simulate the
feedback of customers over the selected services for deployment on the cloud, we use the “leave-one-out” method again. If the recommended service for the left-out case is not similar to the actual registered service in the case base, we assume that the customer is not satisfied with the recommended service. Therefore, the reinforcement learning algorithm increases the “credit” of the retrieved case if the recommended solution is correct. Otherwise the “credit” is decreased. Based on this method, we set the K parameter in the K-NN retrieval method from 1 to 3. Hence, 3 similar cases are retrieved for a target case. These cases are ranked based on their associated credit attribute. The solution with the highest credit is presented as the recommended solution to the customer. We repeat the experiments for 3 rounds on the all cases in the Application Case Base to show that as the system is used and the “credit” of cases are adjusted based on the decisions they involved in, the precision of recommendations improves.

![Figure 5.10. Effect of reinforcement learning on the accuracy of the results.](image)

The reinforcement learning algorithm improves the overall precision to 89.6% after three rounds, as shown in Figure 5.10. The first round represents the system with the adaptation phase, while all cases in the case base have the same initial value of the “credit” attribute (i.e., “1”). After this round, the “credit” of the cases in the case base changes based on how frequently the case is retrieved as a potential solution. The next two rounds take into account the credit factor in the recommendation decision. We observe that, after three rounds the precision changes are insignificant.
5.5 Summary

Cloud computing provides a cost efficient platform for many applications, but due to the number of available options and fast growth of this industry, the selection of a suitable service or a set of services for an application becomes a challenge for cloud customers. Case-based reasoning is a viable option to assist cloud customers in finding the best service for their applications based on previous experiences of other customers and experts.

In this chapter we propose QuARAMRecommender CBR, a case-based recommendation system for cloud service selection based on the application’s requirements and the preferences and priorities of the customers. We investigate different aspects of the case-based reasoning system to cover all dimensions of the service selection decision.

We implemented a proof-of-concept prototype to show the feasibility of our approach. We define a set of similarity measures to calculate the similarity of the new application to previously deployed applications on cloud. We accommodate the customer’s preferences as a set of weights that are used in the similarity calculations. Adaptation is used in the proposed system to adapt previous deployments to better fit the requirements of the new application. We argue that the more data we have the higher the accuracy we can obtain from the system. This means that the system requires a reasonable number of cases to offer good results. New providers and service offerings need to be continuously updated in our Providers Knowledge Base. This growth may cause inconsistency in the case base which can decrease the precision of the recommendations. We use reinforcement learning to keep the precision high and the case base more consistent.

The final results of 89.6% precision shows that using case-based reasoning is a reasonable method for cloud service selection.
Chapter 6

Service Identification and Consolidation

6.1 Introduction

In the previous chapter, we talk about finding a suitable service for an application using the case-based reasoning approach. This approach requires cases in the case base that are similar to the target case. Sometimes there is no similar experience in the case base. In this situation, we need to find providers that can supply the required functionality for the entire application without the case-based recommender system. Typically, a cloud application consists of a set of deployment entities. We describe an example of the Webapplication topology in Chapter 4 and its TOSCA Service Template in Appendix A. As described in Chapter 4, the “server” entities in the Service Template of an application represent deployment entities that are deployed on the cloud.

For some applications (especially those that have large system requirements), it is more suitable to deploy different components of the application (i.e., deployment entities) on different services. Large VMs are expensive and if the application can be distributed over multiple VMs, it may lead to a better price. For example, let’s assume an application with 7 deployment entities (one of which requires two instances) and the following system requirements for the entire application:\(^5\): 24 vCPU, 59 GB of memory, 201GB of storage, 1GB download and 10 GB upload bandwidth. If we select Microsoft Azure as the service provider to deploy this application on a minimum number of services (just 3 VMs), the lowest possible deployment cost would be $4653.02 per month (based on a manual calculation). If we distribute this application on 8 smaller VMs on Microsoft Azure, the deployment cost can be reduced to $2047.09 per month, which is 56% less than the first deployment.

For each of the deployment entities, the search for the suitable service can be performed using the case-

---

\(^5\) This is one of the applications which we use for our experiments in this chapter
based recommender explained in Chapter 5. If no similar deployment entity is found in the case base (i.e.,
the similarity to the most similar case is less than a specified threshold), then the recommendation system
searches for a suitable service among all available services that are offered by the cloud providers.
After finding suitable services for each deployment entity of the application, we integrate all these services
into a deployment plan. This plan includes the regions, that is the data centers where entities must be
deployed, and consolidation of services to either lower the overall price or achieve a higher performance or
both. Service consolidation aims to reduce the number of cloud services used by the application (i.e., the
number of VM instances in our case). The choices made for the integration deployment plan are important
to the application performance and customer satisfaction.

The remainder of this chapter is organized as follows. In Section 2 we explain in detail the service selection
methods we use for deployment entities and the experiments we conducted to validate the selected methods.
Section 3 shows our methodology for consolidating the deployment entities for deployment entities to
improve the utilization of the services while reducing the total cost of the deployment of an application.
Lastly, we summarize the chapter in Section 4.

6.2 Service Selection for Deployment Entities
The numbers of available cloud services and providers are growing rapidly. The large number of services
makes the search process for a suitable service slow. Our ultimate objective is to decrease the number of
comparisons as well as to distinguish between similar service offerings that satisfy an application’s
requirements. The distinguishing parameters that compare between similar services can be defined based
on the customer’s preferences for service features or performance of the services for specific application
types (i.e., computation intensive, memory intensive, and bandwidth intensive). Customers expect to
receive recommendations for services that satisfy all the system requirements of their application. They
also expect to pay as little as possible for these services with no impact on the performance. Our objective
is also to minimize the overall cost for applications with multiple deployment entities.
Each entity can have some or all of the following system requirements: vCPU, memory, storage, I/O performance, data transfer (download/ upload), number of instances and CPU power. The customer assigns weights for each of the system requirements to reflect their relative importance and their impact on the service selection decision.

The customer may have another set of requirements for the entire application such as: region, availability, security and price with varying preferences. Security requirements are considered later in the “deployment engine” and the “deployment manager” when planning and provisioning the application. Therefore, the following set of requirements for each deployment entity are used for service selection: vCPU, memory, storage, I/O performance, data transfer, CPU power, region, and availability. Our objective is to find a set of services that have the lowest deployment price and highest performance while satisfying the other requirements of the application.

Some of the application deployment entities may need multiple instances for performance, security or fault tolerance purposes. In this case, our service consolidation places the entities on multiple separate services for fault tolerance. In the process of service consolidation we use this information as is described later in Section 3.

6.2.1 Methodology

The number of available services for deployment of entities is large and growing very fast. We use clustering to decrease this search space and improve the overall response time, while maintaining high precision. We use the k-means clustering algorithm [58] to make a model to cluster all available services. The clustering model then is used to find the cluster to which a deployment entity belongs. The search engine aims to find the service in that cluster from different providers that best satisfies the entity’s requirements. Within the cluster, we use the K-Nearest Neighbor (K-NN) algorithm [44] to find the most similar service to the target entity. Customer preferences over different requirements are integrated into the similarity measure as weights (Equation (1)).
Sim(service, deployment entity) = \frac{\sum_{i=1}^{n} w_i \times Sim(f_i^s, f_i^d)}{\sum_{i=1}^{n} w_i}

where \(w_i\) is the weight of the \(i^{th}\) requirement of the application entity, \(f_i^d\) is the application’s required value for the \(i^{th}\) requirement and \(f_i^s\) is the service value of the \(i^{th}\) requirement.

Federated clouds (a.k.a., cloud federation) refers to a group of cloud providers that share their resources and collaborate to improve each other’s services [133]. Studies show that deploying cloud applications on federated clouds can bring several benefits including cost effectiveness, scalability, fault tolerance, and reliability [148, 156],[133]. Therefore, we perform an initial search for the most similar service for the entire application from the available service list of all providers, given that there is no customer preference for providers. Although federated clouds may decrease the total price of the deployment entities, applications with multiple deployment entities may suffer from performance degradation and price increases due to communication and data transfer costs between entities deployed on different cloud providers. Inter-cloud communication and interoperability issues remain challenging for cloud providers, which makes deployment over federated clouds a less desirable option for some applications [156]. There are also other challenges related to federated clouds environments like security, management and monitoring [133, 148]. However, federated clouds could be a much cheaper option for applications that do not require a high amount of communication between their different deployment entities.

In order to recommend a suitable provider for the application when there is no similar application in the case base, we first search for the most suitable service for each deployment entity separately. Price is always an important factor in making decisions. The importance of the service price attribute to the customer is represented as a weight for this feature by the customer. In addition to comparing available services based on the application’s and deployment entities’ requirements, services are also compared based on performance. Services have varying performance based on their resources and the type of deployed
The performance information on services can be obtained by an independent third-party service like CloudHarmony [32] or the service proposed by Acs et al. [4].

The objective performance measures for most IaaS services are available in CloudHarmony. It provides independent and objective analysis on cloud services using various benchmarks to compare cloud providers. Acs et al. [4] use a hierarchical fuzzy system to reduce the complexity of the performance comparison and provide a comparable and readable performance analysis of IaaS providers. The performance objective can be based on individual resources such as CPU, memory, and disk or the overall service performance.

Cloud providers provision VMs in different categories with various configurations in terms of CPU, storage, memory and networking capacity. These categories are optimized to offer better performance in specific applications (such as computation-intensive or memory-intensive).

The most suitable service is selected based on the similarity of the service to the system requirements of the deployment entity, the service price, and the performance of the service based on the deployment entity’s “category”. For example, if the “category” of a deployment entity is of type “CPU-Optimized” the service price and the CPU-relevant performance of potential services are used for comparison.

A service with the highest similarity, lowest price and highest performance is the most suitable service for the deployment entity. In view of the fact that this combination is not always possible for a service, the similarity, price and performance of the potential services are compared to the maximum and minimum values for these parameters amongst the top n most similar services to the deployment entity. We incorporate the customer preferences on each of these parameters by adding suggestive coefficients to each of these parameters. We define a linear fitness function (LFF) which can be used to rank the potential services.

\[
\text{fitness}(\text{service}) = \alpha \Delta \text{Similarity}(\text{service}, \text{server}) + \beta \Delta \text{Price}_{\text{service}} + \\
\gamma \Delta \text{Performance}_{\text{service}}(\text{type})
\]  

where \(\alpha + \beta + \gamma = 1\) and using min-max normalization [58] \(\Delta \text{Similarity}, \Delta \text{Performance}, \) and \(\Delta \text{Price}\)
are calculated as follows:

\[ \Delta \text{Similarity} (\text{service, server}) = 1 - \frac{\text{Similarity}_{\text{max}} - \text{Similarity}_{\text{service}}}{\text{Similarity}_{\text{max}} - \text{Similarity}_{\text{min}}} \]  

(3)

\[ \Delta \text{performance}_{\text{service}} (\text{type}) = 1 - \frac{\text{Performance}_{\text{max}} - \text{Performance}_{\text{service}}}{\text{Performance}_{\text{max}} - \text{Performance}_{\text{min}}} \]  

(4)

and

\[ \Delta \text{Price}_{\text{service}} = \frac{\text{Price}_{\text{max}} - \text{Price}_{\text{service}}}{\text{Price}_{\text{max}} - \text{Price}_{\text{min}}} \]  

(5)

We use \( \alpha, \beta, \) and \( \gamma \) as the weights for similarity, price and performance criteria respectively, where \( \alpha, \beta \) and \( \gamma \) are calculated using the weights that the customer assigns for each of the requirements as follows:

\[ \alpha = \frac{\text{Avg} (W^{-})}{\text{total_weight}}, \quad \beta = \frac{W_{\text{price}}}{\text{total_weight}}, \quad \text{and} \quad \gamma = \frac{W_{\text{perf}}}{\text{total_weight}} \]  

(6)

where \( W \) is the set of customer-assigned weights to the requirements of the deployment entity, \( W^{-} \subseteq W = W - \{ W_{\text{price}} \} \), \( W_{\text{price}} \) is the price weight and \( W_{\text{perf}} \) is the performance weight. This performance weight is determined by a domain expert. The \text{total_weight} \ is defined as follows.

\[ \text{total_weight} = \text{Avg} (W^{-}) + W_{\text{price}} + W_{\text{perf}}, \]  

(7)

where \( \text{Avg} (W^{-}) \) is the average of all the customer-assigned weights except the price weight.

For example, assume that a deployment entity has the following customer-assigned weights to the requirements: \( W_{\text{CPU}}=6, \ W_{\text{Memory}}=6, \ W_{\text{Storage}}=4, \ W_{\text{Region}}=5, \ W_{\text{Price}}=10, \ W_{\text{OS}}=10, \ W_{\text{Availability}}=6, \ W_{\text{I/O Performance}}=10, \ W_{\text{DataTransfer}}=4 \). The expert set the performance weight as \( W_{\text{perf}}=5 \).

\[ \text{Avg}(W^{-}) = \frac{6 + 6 + 4 + 5 + 10 + 6 + 10 + 4}{8} = 6.375 \]

\[ \text{total_weight} = \text{Avg}(W^{-}) + W_{\text{price}} + W_{\text{perf}} = 6.375 + 10 + 10 = 26.375 \]
\[ w_{\text{similarity}} = \alpha = \frac{\text{Avg}(W^-)}{\text{total_weight}} = \frac{6.375}{26.375} = 0.242 \]

\[ w_{\text{price}} = \beta = \frac{W_{\text{price}}}{\text{total_weight}} = \frac{10}{26.375} = 0.379 \]

\[ w_{\text{performance}} = \gamma = \frac{W_{\text{perf}}}{\text{total_weight}} = \frac{10}{26.375} = 0.379 \]

The top \( n \) services are ranked based on their fitness score. The service with the highest score is selected for the deployment entity.

For example consider the services with similarities, performances and prices for a deployment entity as shown in Table 6.1.

<table>
<thead>
<tr>
<th>Service number</th>
<th>Similarity</th>
<th>Price</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.95</td>
<td>1.18</td>
<td>87</td>
</tr>
<tr>
<td>2</td>
<td>0.94</td>
<td>1.10</td>
<td>87</td>
</tr>
<tr>
<td>3</td>
<td>0.94</td>
<td>1.08</td>
<td>85</td>
</tr>
<tr>
<td>4</td>
<td>0.90</td>
<td>1.00</td>
<td>90</td>
</tr>
<tr>
<td>5</td>
<td>0.98</td>
<td>1.28</td>
<td>88</td>
</tr>
</tbody>
</table>

The fitness values for these services using the coefficients (\( \alpha, \beta \) and \( \gamma \)) above are as follows: service1= 0.44, service2= 0.52, service3= 0.39, service4= 0.76, service5= 0.47. The ranking of these services is: 4,2,5,1, and 3. As the result, service 4 is selected for the deployment entity.

With these conflicting criteria for selecting suitable service for deployment entities, we can also use and benefit from multi-criteria decision making (MCDM) approaches [159] to solve this problem. MCDM (a.k.a., multi-criteria decision analysis (MCDA)) is a sub-discipline of operations research, which aims to design mathematical and computational tools for selecting the best alternative among several choices with respect to several criteria [168]. Rehman et.al., [123] had a comparative study on different methods of
MCDM for IaaS service selection based on performance measurements made by CloudHarmony[32]. While their study shows MCDM techniques are effective and can be used for IaaS service selection, it also reveals that TOPSIS (Technique for Order of Preferences by Similarity to Ideal Solution) is the most suitable method for service selection when the number of available services are very large. Based on the findings in this study we use the TOPSIS method for ranking and selecting the most suitable service for a deployment entity.

TOPSIS was proposed by Hwang and Yoon in 1981 [159]. The main idea of this method is to select an alternative that is closest to the positive ideal solution and simultaneously farthest from the negative ideal solution (anti-ideal solution). The distance of alternatives from positive and negative ideals solution are calculated based on the Euclidean distance [159]. The optimal solution should have the shortest distance from the ideal solution and the farthest from the anti-ideal one [168].

The procedures of the TOPSIS method in our IaaS service selection are described as follows. Given a set of available services, \( A = \{A_k \mid k = 1, \ldots, n\} \), a set of criteria (i.e., similarity, price and performance in our service selection), \( C = \{C_j \mid j=1,\ldots,m\} \), a set of performance ratings (i.e., values for each criterion), \( X = \{x_{kj} \mid k=1,\ldots,n; j=1,\ldots,m\} \), and a set of weights, \( W = \{w_j \mid j=1,\ldots,m\} \), the information table \( I=(A, C, X, W) \) can represented as Table 6.2.

Table 6.2. Information table of TOPSIS

<table>
<thead>
<tr>
<th>Available services</th>
<th>C₁</th>
<th>C₂</th>
<th>…</th>
<th>Cₘ</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₁</td>
<td>( x_{11} )</td>
<td>( x_{12} )</td>
<td>…</td>
<td>( x_{1m} )</td>
</tr>
<tr>
<td>A₂</td>
<td>( x_{21} )</td>
<td>…</td>
<td>( x_{2m} )</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>Aₙ</td>
<td>( x_{n1} )</td>
<td>( x_{n2} )</td>
<td>…</td>
<td>( x_{nm} )</td>
</tr>
<tr>
<td>W</td>
<td>( w_1 )</td>
<td>( w_2 )</td>
<td>…</td>
<td>( w_m )</td>
</tr>
</tbody>
</table>
The first step is to calculate the normalized ratings using Equation 8.

\[ r_{kj}(x) = \frac{x_{kj}}{\sqrt{\sum_{k=1}^{K} x_{kj}^2}}, \quad k = 1, ..., n; j = 1, ..., m. \] (8)

Then the weighted normalized ratings are calculated as follow:

\[ v_{kj}(x) = w_j r_{kj}(x), \quad k = 1, ..., n; j = 1, ..., m. \] (9)

The next steps is to find the positive (PIS) and negative (NIS) ideal solutions. These solutions are derived as:

\[ PIS = \{v_1^+(x), v_2^+(x), ..., v_j^+(x), ..., v_m^+(x)\} = [(\max_k v_{kj}(x)|j \in J_1), (\min_k v_{kj}(x)|j \in J_2)|k = 1, ..., n], \] (10)

\[ NIS = \{v_1^-(x), v_2^-(x), ..., v_j^-(x), ..., v_m^-(x)\} = [(\min_k v_{kj}(x)|j \in J_1), (\max_k v_{kj}(x)|j \in J_2)|k = 1, ..., n]. \] (11)

Where \( J_1 \) are the benefit attributers (larger is better, e.g., similarity and performance) and \( J_2 \) are cost attributes (smaller is better, e.g., price).

Next step is to calculate the distance of each available service from positive and negative ideal solutions (i.e., \( D_h \) and \( D_l \) respectively) using Euclidean distance,

\[ D_h = \sqrt{\sum_{j=1}^{m} [v_{kj}(x) - v_j^+(x)]^2}, \quad k = 1, ..., n \] (12)

and

\[ D_l = \sqrt{\sum_{j=1}^{m} [v_{kj}(x) - v_j^-(x)]^2}, \quad k = 1, ..., n. \] (13)

The similarity of the available services to PIS are then calculated as:

\[ Similarity \ index = \frac{D_l}{D_h + D_l} \] (14)

Finally the available services are ranked by their similarity index and the service with highest similarity
index is selected as the best solution.

Considering the same example of services in Table 6-1, using TOPSIS and the weights \((\alpha, \beta \text{ and } \gamma)\) above, the similarity index of the services are as follows: service1= 0.37, service2= 0.62, service3= 0.65, service4= 0.82, service5= 0.20. The ranking of these services is: 4,3,2,1, and 5. As the result, service 4 is selected for the deployment entity.

While the ranking of the services using LFF and TOSIS are different, the final selected service is the same. We compare these two ranking methods in this chapter to see which one is better for our service selection based on price, performance and similarity.

Finding the best service for each deployment entity gives us a list of providers with the most suitable service that satisfies the minimum system requirements, price and the required performance. Typically, federated clouds provide the best price while satisfying almost all the requirements of the customer (the minimum system requirements had to be satisfied but some features such as the region, availability and I/O performance may or may not be entirely satisfied according to the weights assigned by the customer). Given the difficulty with federated clouds, while federated clouds solutions are considered, we currently favor a list of services on the same provider for performance purposes. Comparing the total price, the average similarity and performance of the services for the whole application, we can rank potential providers and recommend the best match.

The following steps summarize our service selection algorithm:

1) For all the deployment entities of the application:
   a. Classify the deployment entity in one of the clusters using the clustering model
   b. Find the most suitable service for the deployment entity in that cluster with respect to the entity’s requirements, service price and performance

2) For each “distinct” provider that is suggested for the deployment entities (results of step 1), redo step 1 for potential services on that provider.

3) Rank potential providers based on the total price, average performance and average similarity of the
services to the deployment entities

For example if in step 1 the suggested services for the deployment entities are on “Amazon” and “eApps”
then in step 2, we find the suitable services for the deployment entities once on “Amazon” and once on
“eApps”. The result is two sets of services on these providers that then can be compared based on the overall
price, performance, and similarity.

6.2.2 Experiments and Results

We conduct a set of experiments to validate our proposed method. The first set of experiments studies the
service selection for each of the deployment entities of a set of applications and then selecting single
providers for the applications deployment. In this chapter we show the detailed results of our experiments
on one of the applications and the rest are reported in Appendix C. We compare LFF and TOPSIS using
the results of these experiments.

We also use clustering to decrease the search space and as a result decreasing the response time in service
selection. The next set of experiments in this section show the impact of the use of clustering on the selection
process.

We gathered the specifications (the provided resources, “category”, availability, and service price) of 1120
services from 8 different providers in our Providers Knowledge base. Those providers are in 9 different
regions across the globe. Our knowledge base also includes 110 block storage services from various
providers.

6.2.2.1 Service Selection

We conduct experiments using 4 Web applications with different requirements and deployment entities.
The system requirements for these applications are: vCPU, memory, storage, region, price and OS. The
weights for the requirements of the application entities are defined separately (i.e., two deployment entities
of an application may have different weights for their requirements). Tables 6-3 to 6-6 list the deployment
entity requirements and the assigned weights for each of the applications’ deployment entities. Assigned
weights range from 0 to 1, where 1 means “most important” and 0 is “least important”.

For these applications there are no preferences over the providers. The “0” value in any column indicates that the deployment entity has no specific requirement for this feature. We set the region to “US” for all applications across all experiments. This means that the customer prefers to deploy the application on one of the “US” regions (e.g. “US-West”, “US-East”, etc.). Some applications run multiple instances of the same deployment entity.

While the customer may specify the type of application in the specification, it’s not necessarily that this type applies to the deployment entities. For example an application may be of type “CPU intensive” but it does not mean that all its deployment entities are “CPU intensive”. It may have a database server deployment entity which is “memory intensive”. To compare the performance of potential services, we need to identify the “category” of deployment entities.

Most cloud providers classify their services into the following “categories” based on VM configurations: “Compute optimized”, “Memory optimized”, “Storage optimized” and “General purpose”. Using the configuration information (i.e., memory, CPU, and storage) we train a classifier and use it to categorize the application deployment entities based on their system requirements. We use the WEKA data mining tool [166] and c4.5 algorithm [101] (implemented with the name J48 in WEKA) to train our classifier model.

Table 6.3 shows the requirements of application 1. It includes 8 deployment entities and requires two instances of entity 3. Table 6.4 shows the system requirements of application 2, which includes 13 deployment entities and requires two instances of entity 7. An empty cell in the OS attribute indicates that any operating system is acceptable. Table 6.5 shows the system requirements and weights of application 3. This application has 10 deployment entities, where entity 2 requires two instances and entity 6 requires 3 instances. Table 6.6 shows the system requirements and associated weights for application 4, which has 3 deployment entities each with a single instance.

For all the applications, the weight of the region is set to 0.6, the weight of the OS is 1 if the OS is specified, and the weight of the price is 1 (which means the price is very important).
Table 6.3. System requirements of application 1.

<table>
<thead>
<tr>
<th>Deployment entity</th>
<th>vCPU/weight</th>
<th>Memory/weight</th>
<th>Storage/Weight</th>
<th>category</th>
<th>price</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4/0.6</td>
<td>6/0.6</td>
<td>0/0</td>
<td>'CPU Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>2</td>
<td>2/0.6</td>
<td>8/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>3</td>
<td>2/0.6</td>
<td>1/0.6</td>
<td>20/1</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>3</td>
<td>2/0.6</td>
<td>1/0.6</td>
<td>20/1</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>4</td>
<td>0/0</td>
<td>0/0</td>
<td>111/1</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>5</td>
<td>2/0.6</td>
<td>3/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>6</td>
<td>8/0.8</td>
<td>32/0.8</td>
<td>50/0.6</td>
<td>'Memory optimized'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>7</td>
<td>4/0.8</td>
<td>8/0.8</td>
<td>0/0</td>
<td>'CPU optimized'</td>
<td>0</td>
<td>Windows</td>
</tr>
</tbody>
</table>

Table 6.4. System requirements of application 2.

<table>
<thead>
<tr>
<th>Deployment entity</th>
<th>vCPU/weight</th>
<th>Memory/weight</th>
<th>Storage/Weight</th>
<th>category</th>
<th>price</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2/0.6</td>
<td>3/0.6</td>
<td>20/0.4</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>2</td>
<td>1/0.6</td>
<td>2/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>3</td>
<td>1/0.6</td>
<td>2/0.6</td>
<td>20/0.4</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>4</td>
<td>2/0.6</td>
<td>8/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>5</td>
<td>2/0.6</td>
<td>8/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>6</td>
<td>2/0.6</td>
<td>8/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>7</td>
<td>2/0.6</td>
<td>1/0.6</td>
<td>100/0.8</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>7</td>
<td>2/0.6</td>
<td>1/0.6</td>
<td>100/0.8</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>8</td>
<td>2/0.6</td>
<td>4/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>9</td>
<td>0/0</td>
<td>0/0</td>
<td>520/1</td>
<td>Storage Optimized'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>10</td>
<td>2/0.6</td>
<td>3/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>11</td>
<td>12/0.8</td>
<td>65/0.6</td>
<td>250/0.8</td>
<td>'Memory optimized'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>12</td>
<td>8/0.6</td>
<td>8/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
</tbody>
</table>

To recommend a set of suitable services for each application, the system searches the providers’ knowledge base (1120 services) for candidate services for each deployment entity. The service performance attributes (CPU, memory, disk and total performance) are initially populated with random values based on the type of the services. Acs et.al. [4] propose to aggregate the values of each of these attributes into a single value between 0-100, rather than having a vector of performance metrics for each attribute. Based on this proposal, we assume that the performance values of services are provided by a third party in the form of values between 0 and 100.
Table 6.5. Application 3 system requirements.

<table>
<thead>
<tr>
<th>Deployment entity</th>
<th>vCPU/weight</th>
<th>Memory/weight</th>
<th>Storage/Weight</th>
<th>category</th>
<th>price</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2/0.6</td>
<td>3/0.6</td>
<td>50/0.6</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>2</td>
<td>1/0.6</td>
<td>2/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>2</td>
<td>1/0.6</td>
<td>2/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>3</td>
<td>1/0.6</td>
<td>4/0.6</td>
<td>50/0.6</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>4</td>
<td>2/0.6</td>
<td>6/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>5</td>
<td>2/0.6</td>
<td>6/0.6</td>
<td>0/0</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>6</td>
<td>2/0.4</td>
<td>1/0.6</td>
<td>200/0.8</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>6</td>
<td>2/0.4</td>
<td>1/0.6</td>
<td>200/0.8</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>6</td>
<td>2/0.4</td>
<td>1/0.6</td>
<td>200/0.8</td>
<td>'General Purpose'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>7</td>
<td>0/0</td>
<td>0/0</td>
<td>2000/1</td>
<td>'General Purpose'</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.6. System requirements of application 4.

<table>
<thead>
<tr>
<th>Deployment entity</th>
<th>vCPU/weight</th>
<th>Memory/weight</th>
<th>Storage/Weight</th>
<th>category</th>
<th>price</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4/0.6</td>
<td>8/0.6</td>
<td>0/0</td>
<td>'CPU optimized'</td>
<td>0</td>
<td>Windows</td>
</tr>
<tr>
<td>2</td>
<td>3/0.6</td>
<td>16/0.8</td>
<td>50/0.8</td>
<td>'Memory optimized'</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2/0.6</td>
<td>16/0.8</td>
<td>0/0</td>
<td>'Memory optimized'</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Similarity measures are the same as the ones described in Chapter 5 for each of the application’s requirements. Computational units with high storage are expensive. However, if a service with lower-than-required storage can satisfy all other requirements, an extra storage block can be added to the VM at a much lower price. For example, for deployment entity 6 in application 3, the requirements are: 2 vCPU, 1 GB memory, 200 GB of storage and US region. A service that satisfies all these requirements would cost a minimum price of 0.24$ per hour on Windows Azure. But if we search for a service that satisfies the vCPU, memory and region and then add the required extra storage to the VM, the price would be 0.104$ per hour for both VM and extra storage on the same provider. This is almost half of the large VM price. Based on this observation we set the storage weight to the lowest value to decrease its impact on the similarity calculation.

In each experiment, similarity of the services to the deployment entity requirements is calculated using the
K-NN algorithm. The similarity of the services that cannot satisfy all minimum system requirements of the deployment entity is set as 0.

In the first phase of the experiments, we find the most suitable services for each of the deployment entities separately, regardless of recommended services for other deployment entities of the same application. For each deployment entity, we use LFF and TOPSIS to find the most suitable service. Table 6.7 and 6.8 illustrate the result of the selected services for application 1 using LFF and TOPSIS, respectively.
Table 6.7. Selected services for application 1 using LFF.

<table>
<thead>
<tr>
<th>Deployment entity</th>
<th>Similarity</th>
<th>Provider</th>
<th>name</th>
<th>vCPU</th>
<th>Mem (GB)</th>
<th>Storage (GB)</th>
<th>Region</th>
<th>Category</th>
<th>Price ($/hr)</th>
<th>OS</th>
<th>Performance</th>
<th>extra storage price ($/hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>86.02</td>
<td>Microsoft Azure</td>
<td>Large(A3)</td>
<td>4</td>
<td>7</td>
<td>120</td>
<td>US-North</td>
<td>'General Purpose'</td>
<td>0.308</td>
<td>Windows</td>
<td>99</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>80.96</td>
<td>Microsoft Azure</td>
<td>A5</td>
<td>2</td>
<td>14</td>
<td>135</td>
<td>Asia</td>
<td>'Memory Optimized'</td>
<td>0.31</td>
<td>Windows</td>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>86.55</td>
<td>Microsoft Azure</td>
<td>Medium (A2)</td>
<td>2</td>
<td>3.5</td>
<td>60</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.154</td>
<td>Windows</td>
<td>87</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>76.19</td>
<td>Amazon</td>
<td>m3.xlarge</td>
<td>4</td>
<td>15</td>
<td>80</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.28</td>
<td>Linux</td>
<td>99</td>
<td>0.0042 47</td>
</tr>
<tr>
<td>5</td>
<td>86.72</td>
<td>Microsoft Azure</td>
<td>Medium (A2)</td>
<td>2</td>
<td>3.5</td>
<td>60</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.154</td>
<td>Windows</td>
<td>87</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>86.06</td>
<td>Microsoft Azure</td>
<td>A7</td>
<td>8</td>
<td>56</td>
<td>605</td>
<td>US-East</td>
<td>'Memory Optimized'</td>
<td>1.32</td>
<td>Windows</td>
<td>10 0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>87.77</td>
<td>Joyent</td>
<td>Standard-5</td>
<td>4</td>
<td>15</td>
<td>1467</td>
<td>US-East</td>
<td>'General Purpose'</td>
<td>0.54</td>
<td>Windows</td>
<td>99</td>
<td>0</td>
</tr>
</tbody>
</table>

Average Similarity= 84.61
Average performance= 93.25
Total Price = 3.2242
Table 6.8. Selected services for application 1 using TOPSIS.

<table>
<thead>
<tr>
<th>Deployment entity</th>
<th>Similarity</th>
<th>Provider</th>
<th>name</th>
<th>vCPU</th>
<th>Mem (GB)</th>
<th>Storage (GB)</th>
<th>Region</th>
<th>Category</th>
<th>Price ($/hr)</th>
<th>OS</th>
<th>Performance</th>
<th>extra storage price ($/hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>86.59</td>
<td>eApps</td>
<td>?</td>
<td>4</td>
<td>8</td>
<td>75</td>
<td>US-West</td>
<td>'CPU Optimized'</td>
<td>0.182</td>
<td>Windows</td>
<td>86</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>80.96</td>
<td>Microsoft Azure</td>
<td>A5</td>
<td>2</td>
<td>14</td>
<td>135</td>
<td>Asia</td>
<td>'Memory Optimized'</td>
<td>0.31</td>
<td>Windows</td>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>86.55</td>
<td>Microsoft Azure</td>
<td>Medium (A2)</td>
<td>2</td>
<td>3.5</td>
<td>60</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.154</td>
<td>Windows</td>
<td>87</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>76.19</td>
<td>Microsoft Azure</td>
<td>X-Small (A0)</td>
<td>1</td>
<td>0.77</td>
<td>20</td>
<td>US-East</td>
<td>'General Purpose'</td>
<td>0.02</td>
<td>Linux</td>
<td>88</td>
<td>0.0062 33</td>
</tr>
<tr>
<td>5</td>
<td>81.39</td>
<td>Amazon</td>
<td>t2.mediu m</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>Australia</td>
<td>'General Purpose'</td>
<td>0.1</td>
<td>Windows</td>
<td>89</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>85.88</td>
<td>Joyent</td>
<td>High Storage</td>
<td>8</td>
<td>32</td>
<td>7680</td>
<td>Us-South</td>
<td>'Storage Optimized'</td>
<td>0.923</td>
<td>Windows</td>
<td>72</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>88.47</td>
<td>eApps</td>
<td>?</td>
<td>4</td>
<td>8</td>
<td>75</td>
<td>US-West</td>
<td>'CPU Optimized'</td>
<td>0.182</td>
<td>Windows</td>
<td>86</td>
<td>0</td>
</tr>
</tbody>
</table>

Average Similarity = 84.07

Average performance = 85.375

Total Price = 2.031
As Table 6.7 and 6.8 suggest, the best combination of price, performance and similarity to requirements of the deployment entities for an application can be achieved by deploying the application on federated clouds (suggested configuration with best price, performance and similarity is the deployment of the application on 3 and 4 providers respectively). This means that if we wish to deploy the application on a single provider, it may have lower similarity, lower performance, higher price or all of the above. To validate this observation, we repeat the experiment with a restriction on the provider. Therefore, each application must be deployed on only one provider. In this case, we select one service provider for each application.

**Service selection using IFF**

Table 6.9 shows the average similarity, total price, average performance, and fitness values for each provider for application 1, where “N/A” indicates that some providers are unable to satisfy all the application requirements. Hence, those providers cannot provide services to the application. We rank different configurations (federated and single providers) with IFF and the fitness values show that a federated cloud has the best combination of price, performance and similarity for the deployment of the application. Table 6.10 shows the same results for the other three applications.

**Table 6.9. Similarity and price of different providers for application 1 using LFF.**

<table>
<thead>
<tr>
<th>Providers</th>
<th>Federated</th>
<th>Amazon</th>
<th>Microsoft</th>
<th>Azure</th>
<th>Rackspace</th>
<th>GoGrid</th>
<th>eApps</th>
<th>Atlantic.net</th>
<th>Joyent</th>
<th>e24Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average Similarity</strong></td>
<td>84.61</td>
<td>83.15</td>
<td>83.53</td>
<td>79.21</td>
<td>51.68</td>
<td>N/A</td>
<td>N/A</td>
<td>83.11</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td><strong>Total Price</strong></td>
<td>3.224</td>
<td>2.899</td>
<td>3.248</td>
<td>5.31</td>
<td>4.765</td>
<td>N/A</td>
<td>N/A</td>
<td>3.435</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td><strong>Average Perf</strong></td>
<td>93.25</td>
<td>89.125</td>
<td>92.75</td>
<td>89.87</td>
<td>85.62</td>
<td>N/A</td>
<td>N/A</td>
<td>84.12</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td><strong>Fitness value</strong></td>
<td>0.937</td>
<td>0.882</td>
<td>0.917</td>
<td>0.401</td>
<td>0.143</td>
<td>N/A</td>
<td>N/A</td>
<td>0.652</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Providers</td>
<td>Application</td>
<td>Federated</td>
<td>Amazon</td>
<td>Microsoft</td>
<td>Azure</td>
<td>Rackspace</td>
<td>GoGrid</td>
<td>eApps</td>
<td>Atlantic.net</td>
<td>Joyent</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>-----------</td>
<td>--------</td>
<td>-----------</td>
<td>-------</td>
<td>-----------</td>
<td>--------</td>
<td>---------</td>
<td>--------------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Similarity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>78.72</td>
<td>77.69</td>
<td>78.73</td>
<td>73.18</td>
<td>42.68</td>
<td></td>
<td></td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Total Price</td>
<td>4.731</td>
<td>4.664</td>
<td>7.996</td>
<td>10.566</td>
<td>10.055</td>
<td></td>
<td></td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Average</td>
<td>89.84</td>
<td>87.76</td>
<td>89.07</td>
<td>85.46</td>
<td>82.61</td>
<td></td>
<td></td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Perf</td>
<td>0.994</td>
<td>0.922</td>
<td>0.703</td>
<td>0.331</td>
<td>0.0416</td>
<td></td>
<td></td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Fitness</td>
<td>70.96</td>
<td>70.92</td>
<td>70.21</td>
<td>67.16</td>
<td>49.04</td>
<td></td>
<td></td>
<td>74.21</td>
<td>58.61</td>
<td>69.09</td>
</tr>
<tr>
<td>value</td>
<td>2.206</td>
<td>2.354</td>
<td>2.128</td>
<td>2.922</td>
<td>5.135</td>
<td></td>
<td></td>
<td>0.969</td>
<td>0.740</td>
<td>3.535</td>
</tr>
<tr>
<td>Average</td>
<td>90.4</td>
<td>89.4</td>
<td>90.1</td>
<td>87.6</td>
<td>77.3</td>
<td></td>
<td></td>
<td>74.9</td>
<td>78</td>
<td>84.5</td>
</tr>
<tr>
<td>Perf</td>
<td>0.802</td>
<td>0.769</td>
<td>0.798</td>
<td>0.638</td>
<td>0.038</td>
<td></td>
<td></td>
<td>0.728</td>
<td>0.640</td>
<td>0.540</td>
</tr>
<tr>
<td>Fitness</td>
<td>78.25</td>
<td>71.86</td>
<td>71.58</td>
<td>67.38</td>
<td>59.24</td>
<td></td>
<td></td>
<td>70.18</td>
<td>62.63</td>
<td>78.15</td>
</tr>
<tr>
<td>value</td>
<td>1.338</td>
<td>1.59</td>
<td>1.54</td>
<td>2.895</td>
<td>1.880</td>
<td></td>
<td></td>
<td>0.778</td>
<td>0.768</td>
<td>1.826</td>
</tr>
<tr>
<td>Average</td>
<td>97.66</td>
<td>88.66</td>
<td>94</td>
<td>92</td>
<td>94.33</td>
<td></td>
<td></td>
<td>93.33</td>
<td>94.66</td>
<td>95.33</td>
</tr>
<tr>
<td>Perf</td>
<td>0.866</td>
<td>0.592</td>
<td>0.701</td>
<td>0.316</td>
<td>0.537</td>
<td></td>
<td></td>
<td>0.857</td>
<td>0.829</td>
<td>0.707</td>
</tr>
<tr>
<td>Fitness</td>
<td>78.25</td>
<td>71.86</td>
<td>71.58</td>
<td>67.38</td>
<td>59.24</td>
<td></td>
<td></td>
<td>70.18</td>
<td>62.63</td>
<td>78.15</td>
</tr>
<tr>
<td>value</td>
<td>1.338</td>
<td>1.59</td>
<td>1.54</td>
<td>2.895</td>
<td>1.880</td>
<td></td>
<td></td>
<td>0.778</td>
<td>0.768</td>
<td>1.826</td>
</tr>
</tbody>
</table>

**Service selection using TOPSIS**

Table 6.11 shows the average similarity, total price, average performance, and similarity index (i.e., relative closeness to ideal solution) derived from TOPSIS for each provider for application 1. We can see from the similarity index of the different configurations, that a federated clouds provides the best combination of similarity, price and performance for the application deployment. Table 6.12 shows the same results for the other three applications.
Table 6.11. Similarity and price of different providers for application 1 using TOSIS.

<table>
<thead>
<tr>
<th>Providers</th>
<th>Federated</th>
<th>Amazon</th>
<th>Microsoft Azure</th>
<th>Rackspace</th>
<th>GoGrid</th>
<th>eApps</th>
<th>Atlantic.net</th>
<th>Joyent</th>
<th>e24Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Similarity</td>
<td>84.07</td>
<td>82.41</td>
<td>84.20</td>
<td>78.82</td>
<td>52.47</td>
<td>N/A</td>
<td>N/A</td>
<td>83.35</td>
<td>N/A</td>
</tr>
<tr>
<td>Total Price</td>
<td>2.031</td>
<td>2.713</td>
<td>2.89</td>
<td>5.034</td>
<td>3.437</td>
<td>N/A</td>
<td>N/A</td>
<td>3.171</td>
<td>N/A</td>
</tr>
<tr>
<td>Average Perf</td>
<td>85.37</td>
<td>88.12</td>
<td>87.125</td>
<td>88</td>
<td>83</td>
<td>N/A</td>
<td>N/A</td>
<td>81.87</td>
<td>N/A</td>
</tr>
<tr>
<td>Similarity index</td>
<td>0.981</td>
<td>0.784</td>
<td>0.732</td>
<td>0.214</td>
<td>0.039</td>
<td>N/A</td>
<td>N/A</td>
<td>0.645</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 6.12. Similarity and price of different providers for applications 2, 3 and 4 using TOPSIS.

<table>
<thead>
<tr>
<th>Application</th>
<th>Providers</th>
<th>Federated</th>
<th>Amazon</th>
<th>Microsoft Azure</th>
<th>Rackspace</th>
<th>GoGrid</th>
<th>eApps</th>
<th>Atlantic.net</th>
<th>Joyent</th>
<th>e24Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Average Similarity</td>
<td>78.29</td>
<td>77.30</td>
<td>78.18</td>
<td>74.32</td>
<td>44.93</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Total Price</td>
<td>4.30</td>
<td>4.38</td>
<td>7.79</td>
<td>6.87</td>
<td>6.71</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Average Perf</td>
<td>88.61</td>
<td>87.30</td>
<td>87.07</td>
<td>79.69</td>
<td>77.31</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Similarity index</td>
<td>1</td>
<td>0.972</td>
<td>0.328</td>
<td>0.401</td>
<td>0.267</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Average Similarity</td>
<td>75.31</td>
<td>75.22</td>
<td>74.48</td>
<td>72.34</td>
<td>51.18</td>
<td>75.29</td>
<td>73.76</td>
<td>74.62</td>
<td>N/A</td>
</tr>
<tr>
<td>Total Price</td>
<td>0.875</td>
<td>1.433</td>
<td>1.826</td>
<td>2.559</td>
<td>2.466</td>
<td>0.880</td>
<td>0.901</td>
<td>2.52</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Average Perf</td>
<td>72.7</td>
<td>85.8</td>
<td>87.2</td>
<td>84.6</td>
<td>74</td>
<td>71.9</td>
<td>71.8</td>
<td>76</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Similarity index</td>
<td>1</td>
<td>0.679</td>
<td>0.461</td>
<td>0.159</td>
<td>0.055</td>
<td>0.908</td>
<td>0.905</td>
<td>0.164</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Average Similarity</td>
<td>75.14</td>
<td>69.01</td>
<td>70.69</td>
<td>67.38</td>
<td>62.10</td>
<td>70.18</td>
<td>71.15</td>
<td>78.16</td>
<td>67.40</td>
</tr>
<tr>
<td>Total Price</td>
<td>0.62</td>
<td>1.38</td>
<td>1.48</td>
<td>2.895</td>
<td>1.881</td>
<td>0.798</td>
<td>0.602</td>
<td>1.766</td>
<td>1.72</td>
<td></td>
</tr>
<tr>
<td>Average Perf</td>
<td>87.33</td>
<td>90.66</td>
<td>92.33</td>
<td>92</td>
<td>90.66</td>
<td>72.66</td>
<td>80.66</td>
<td>91.33</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>Similarity index</td>
<td>0.973783</td>
<td>0.661</td>
<td>0.6203</td>
<td>0.102</td>
<td>0.446</td>
<td>0.880</td>
<td>0.943</td>
<td>0.501</td>
<td>0.504</td>
<td></td>
</tr>
</tbody>
</table>
Although the federated cloud initially provides the best configuration in terms of price, performance and similarity, the total running cost of applications on the cloud may increase due to the extra hidden cost of intercommunications between the various application entities. If an application requires heavy intercommunication between entities, the price of data transfer may more than offset the low price of deployment on federated cloud.

For example, in application 1 with a total price of $2.031/hr (based on the configuration suggested by TOPSIS), if the intercommunication between entities deployed on different providers is about 5TB/month, the total price of the application deployment may increase between $1.31/hr - $2.60/hr (based on the providers). Thus the total application deployment price would be between $3.341/hr-$4.631/hr, which is more than deploying the entire application on most of the other providers. Due to the current challenges of federated clouds including networking, management, monitoring and security issues, we decide to set constraints on the deployment options to a single provider.

While clustering improves the overall response time through reducing the number of comparisons, searching for suitable services on every potential service provider poses too much overhead. For example, in our small provider knowledge base, with only 8 providers, we need to conduct the search 8 times and then compare the prices and average similarities of all providers. Our previous experiments show that some providers cannot provide suitable services (price-wise or similarity-wise) for one or more deployment entities of an application. Based on this observation, we decide to reduce this number of comparisons by restricting potential providers to the ones listed in the first phase (i.e., federated cloud). We claim that it is most likely that the most suitable provider is one of the providers that has at least one service with the highest fitness (using LFF) or similarity index (using TOPSIS) for the deployment entities of the application. Table 6.13 and 6.14 illustrate the list of providers that are potentially best for at least one deployment entity of the applications (using LFF and TOPSIS respectively).

As it can be concluded from Tables 6.9 and 6.12, the best service provider to use for each application’s
deployment appears in the list of the federated cloud option (i.e., using LFF: Microsoft Azure for applications 1 and 3, Amazon for application 2, and eApps for application 4. Using TOPSIS: Amazon for application 1 and 2, eApps for application 3, and Atlantic.net for application 4).

Table 6.13. Potential providers for applications deployment (using LFF).

<table>
<thead>
<tr>
<th>Application number</th>
<th>Providers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Amazon, Microsoft Azure, Joyent</td>
</tr>
<tr>
<td>2</td>
<td>Amazon, Microsoft Azure</td>
</tr>
<tr>
<td>3</td>
<td>Amazon, Microsoft Azure, eApps, Atlantic.net</td>
</tr>
<tr>
<td>4</td>
<td>eApps, Atlantic.net</td>
</tr>
</tbody>
</table>

Table 6.14. Potential providers for applications deployment (using TOPSIS).

<table>
<thead>
<tr>
<th>Application number</th>
<th>Providers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Amazon, Microsoft Azure, eApps, Joyent</td>
</tr>
<tr>
<td>2</td>
<td>Amazon, Microsoft Azure, eApp</td>
</tr>
<tr>
<td>3</td>
<td>Amazon, Microsoft Azure, eApps, Atlantic.net</td>
</tr>
<tr>
<td>4</td>
<td>eApps, Atlantic.net</td>
</tr>
</tbody>
</table>

In summary:

1- We find a list of potential providers that can provision the most suitable service for at least one deployment entity of the application (this is to exclude non-potential providers).

2- For each of those providers, we calculate the average similarity, the total service price, and average performance for the entire application.

3- Then, we select the best provider (i.e., based on the similarity index derived using TOPSIS method or fitness value using LFF).

TOPSIS and LFF have different results for the service selection. To decide on the method we use for service selection we compare these two methods of ranking based on the selection results of our four applications.

6.2.2.2 TOPSIS vs. Linear Fitness Function

Figure 6.1 illustrates the comparison between TOPSIS and LFF ranking and selection methods on similarity, performance and price. Each bar shows the percentage difference between these two ranking
methods. For example the bars for application 2 show that TOPSIS result is 1% better than linear fitness function in similarity, linear fitness function is 3.6% better than TOPSIS in performance of selected services, and TOSIS is 20.1% better than linear fitness function in price.

![TOPSIS vs Linear Fitness Function](image)

**Figure 6.1. Comparison of TOSIS and LFF on similarity, price and performance of selected services**

As the figure illustrates, the performance of the selected services by linear fitness function in all the applications is better than selected services by TOPSIS. Except in the first application that the similarity of selected services by LFF is better than TOPSIS, for the other three applications, TOPSIS has a better selection for the similarity. TOPSIS also comes first in price of selected services. In conclusion, TOPSIS is better than LFF similarity-wise and price-wise, while LFF is better performance-wise.

We argue that because of the following reasons we select TOPSIS as the ranking method for our service selection: 1) TOPSIS is the dominant in two of the three criteria in final selection. 2) While the difference
between these two methods in performance is in the range 4.0%-9.5%, the gap between these two methods regarding the price is in range of 14.9%-33.8%. Also in these experiments, weight of price is set to 1.0 and weight of performance is set to 0.5. This indicates that total price of the application is twice as important as the performance of the services.

6.2.2.3 Clustering
In this set of experiments, we want to show that using clustering is a viable method in service selection because while it decreases the total response time significantly, it has insignificant impact on the precision of the selection. To conduct these experiments, we first compare clustering-based with non-clustering-based method with respect to response time. We compare the methods with different available services to show the trends in response time when the number of available services for selection increases. The precision of the selection results are then compared.

We randomly select 10 services form our 1120 available services and put their features as system requirements of the deployment entities in our test set. This means that we can find exact match services for our test set when the 1120 services are all available.

We conducted the experiment for a varying number of available services 200, 400,600,800, 1000, and 1120. For each set, we find the most suitable services for our test set and then average the response time (in ms) and the precision (%) of the retrieved services for the set. We run the experiment once without clustering (brute force search) and once with clustering.

For each set of available services, we first cluster the services using Weka [166]. Using the clustering model, we find the best cluster for each of the deployment entities in the test set. Then, we use the K-Nearest Neighbor algorithm[44] to find the most similar service that can satisfy the requirements of the deployment entity. The list of requirements for a deployment entity is: vCPU, memory, OS, region, and price. We use the similarity measures explained in Chapter 5. To incorporate the preferences of the customer, we add weights in our algorithm. Table 6.15 shows the assigned weights for each requirement.
Table 6.15. Weights for system requirements.

<table>
<thead>
<tr>
<th>System requirements</th>
<th>vCPU</th>
<th>Memory</th>
<th>Storage</th>
<th>OS</th>
<th>Region</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign weights</td>
<td>0.4</td>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>0.4</td>
<td>0.7</td>
</tr>
</tbody>
</table>

The weights show the relative importance of requirements to the customer. Figure 6.2 illustrates the average response time of finding the most suitable service for the 10 test cases using both clustering and exhaustive search.

We observe that the response time gradually increases as the number of available services increases. However, the increase is much lower in the case of clustering compared to the brute-force search. Clustering significantly improves the response time for higher number of available service.

![Figure 6.2. Effect of using clustering in service selection response time.](image)

While the response time improvement is promising, clustering slightly impacts the overall precision of the service selection. To examine this effect, we calculate the similarity of the selected services to the deployment entity requirements in our test set. Figure 6.3 illustrates the precision of the service selection for the number of services available. In this Figure, the average precision of the 10 test cases in our test set
is used in the chart. As shown in the chart, with the exception of 1120 available services, neither clustering nor non-clustering method achieves 100% precision.

![Figure 6.3. Precision of clustering and non-clustering method in service selection.](image)

Although the chart doesn’t show any specific trend in precision using the clustering method, we can see that the precision of the clustering method is about 6% on average less than the non-clustering method which is insignificant. Thus, clustering considerably improves the response time with insignificant impact on the overall precision.

However, the system may present the average response time and precision for clustering and non-clustering approaches, based on previous deployment experiences, and seek the customer’s preferences.

### 6.3 Service Consolidation

Although the service search engine finds the best service (i.e., VM) for each deployment entity of the application, in a typical scenario the VM is underutilized. For example, Table 6.16 shows the requirements of one deployment entity of application 2 and Table 6.17 illustrates the suggested cloud service for this deployment entity.
Table 6.16. A deployment entity requirements of application 2.

<table>
<thead>
<tr>
<th>Deployment entity number</th>
<th>Provider</th>
<th>VCPU</th>
<th>Memory</th>
<th>Storage</th>
<th>Region</th>
<th>Price</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>65</td>
<td>250</td>
<td>US</td>
<td></td>
<td>Windows</td>
</tr>
</tbody>
</table>

Table 6.17. Suggested cloud service for the deployment entity in Table 6.16.

<table>
<thead>
<tr>
<th>Provider</th>
<th>Name</th>
<th>VCPU</th>
<th>Memory (G)</th>
<th>Storage (G)</th>
<th>Region</th>
<th>OS</th>
<th>Price($/hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>r3.4xlarge</td>
<td>16</td>
<td>122</td>
<td>320</td>
<td>US-East</td>
<td>Windows</td>
<td>1.944</td>
</tr>
</tbody>
</table>

We can see that, the deployment entity will not fully utilize the VM resources, while the customer has to pay the full price of the service.

The objective of this part of our research is to increase the resource utilization, while maintaining or decreasing the total application deployment price.

6.3.1 Methodology

To increase the resource utilization of selected cloud services, we propose to consolidate as many deployment entities as possible in each service, thus decreasing the number of required services for application deployment. Service consolidation has advantages and disadvantages. Consolidating deployment entities with intercommunication reduces the network overhead (and cost) and increases the application’s performance. However, service consolidation poses challenges related to fault tolerance.

In our approach, we start with the largest service in the list of suggested services for the application. We use the price as an indicator of service capacity. Then, we accommodate as many deployment entities as possible in this service. Next, we upgrade the service and consolidate more deployment entities in the service. If the new service configuration (i.e., the upgraded service) has an equal or lower price than the earlier configuration of all consolidated services, the upgrade is positive and acceptable. We continue the same process for the remaining deployment entities of the application. Algorithm 1 illustrates our approach in more detail.

To consolidate deployment entities in a service, we equate consolidation to the knapsack problem [47]. In
this case the knapsack is the largest service that is underutilized. A greedy approximation algorithm [47, 169] is used to solve this knapsack problem.

We begin by sorting the list of suggested services in a descending order based on their price. Then we consolidate deployment entities on smaller services with the first (largest) service (i.e., knapsack).

We set the following rules on the consolidation process:

1- Two instances of the same deployment entity cannot be consolidated onto the same service to maintain the system’s fault tolerance features.

2- Consolidated deployment entities must be of the same operating system.

3- The region for the consolidated deployment entities must be the same or the deployment entities must have low preference on the region (which means satisfying other requirements, e.g., price, is more important than the deployment region).

Algorithm 1 illustrates the steps of reconfiguration of services using our proposed method.
Algorithm 1: Merge Services

Input: List of application deployment entities and the assigned services to the entities
Output: The services and associated entities

begin

  tempServices ← services;
  tempEntities ← Entities;
  bestPrice ← CalculatePrice(Application);

  Sort(tempServices, price); // Sort services based on the services’ prices
  largestService ← tempServices[top];
  remainingServices ← tempServices-largestService;
  remainingEntities ← tempEntities-tempEntities [largestService ];

  while (remainingEntities ≠ ∅) do
    Calculate extra resources of largestService;
    Sort(remainingServices, Price); /* Sort services and entities based on Price of
    the services */
    for all Entity ∈ remainingEntities do /* Merge as many entities as possible to
      largestService */
    if Mergable(Entity, largestService) then /* If there is enough resources for the
      entity and the entity OS requirement is the same as the OS the service
      provides */
    Merge(Entity, largestService);
    Remove(Entity, remainingEntities);
    Remove(service, remainingServices); /* the assigned service to the
    entity */
    ;
  end

  Update(tempServices);
  Update(tempEntities);
  bestPrice ← CalculatePrice(TemporaryApplication);

  while (price = increasing ∧ upgrade possible) do
    Upgrade(largestService);
    Calculate extra resources of largestService;
    for all remainingEntities do /* Merge as many entity as possible to largest
    service */
    if Mergable(Entity, largestService) then /* If there is enough resources for
    Entity */
    Merge(Entity, largestService);
    Remove(Entity, remainingEntities);
    Remove(service, remainingServices);
    ;
  end

  newPrice ← CalculatePrice(TemporaryApplication);
  if (newPrice ≤ bestPrice) then
    bestPrice ← newPrice;
    Update(tempServices);
    Update(tempEntities);
  else
    UndoUpgrade();
    UndoMerges();
  end

  largestService ← NextLargest(tempServices);
  remainingServices ← remainingServices-largestservice;
  remainingEntities ← remainingEntities-remainingEntities [largestService ];

  services ← tempServices;
  Entities ← tempEntities;

end
6.3.2 Experimental environment

We use the same four applications and our knowledgebase in the consolidation experiments. The results of our previous experiments on service selection (described in Section 6.2) are used as an initial set of potential services by our consolidation algorithm. It is worth noting that we do not have any knowledge about the deployment entity intercommunication in an application. Thus, our approach overlooks this aspect during service consolidation. The service price is calculated based on the pay-as-you-go pricing scheme, rather than a yearly contract. The prices of reserved instances are different.

6.3.3 Experiments and results

In first phase of service selection, we find a set of services on different providers that can accommodate all the deployment entities of the applications at the lowest price. In these experiments, we run the consolidation algorithm for each potential provider and compare the net price after consolidation with the total price before consolidation. If the price decreases or remains unchanged, we select the services resulting from the consolidation process. The reason is that the smaller the number of services, the better the management and intercommunication of the application. Otherwise, we ignore the consolidation recommendation if it yields a higher price.

Tables 6.18 to 6.20 shows the results of our consolidation experiments on the four applications. Table 6.18 shows the percentage of cost reduction when we use our proposed service consolidation approach, compared to the scenario where services are selected for each deployment entity individually.

We observe that the consolidation approach achieves good results for applications 1, 2, and 3 for all potential providers. This means that the algorithm improves the resource utilization by incorporating multiple deployment entities in a single service, hence, reducing the deployment price. However, in application 4 the algorithm does not bring the cost down for most providers, except for GoGrid. This means that the underutilized resources are not sufficient to incorporate more deployment entities on the selected services. Nevertheless, Table 6.18 shows that the upgrade results in reducing the number of required
services for the application deployment on four cloud providers, which is also good in terms of deployment entity intercommunications, data transfer, and maintainability costs.

Table 6.18. Percentage of cost reduction in tested applications.

<table>
<thead>
<tr>
<th>Application</th>
<th>Federated</th>
<th>Windows Azure</th>
<th>Amazon</th>
<th>Joyent</th>
<th>Rackspace</th>
<th>GoGrid</th>
<th>eApps</th>
<th>Atlantic.net</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.63</td>
<td>0.27</td>
<td>1.37</td>
<td>5.76</td>
<td>25.72</td>
<td>3.79</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>32.06</td>
<td>6.89</td>
<td>8.43</td>
<td>-</td>
<td>23.97</td>
<td>3.129</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>2.41</td>
<td>9.52</td>
<td>1.21</td>
<td>25</td>
<td>22.28</td>
<td>22.00</td>
<td>2.92</td>
<td>10.80</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13.86</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.19 shows the amount of reduction in the number of required services for deployment of each application on each of the providers. The number of required services decreases for all applications except for application 4 on Joyent, eApps, and Atlantic.net cloud providers. Investigating those providers, we found that upgrading the services results in higher prices. Thus, the consolidation recommendations are ignored for this application on those providers.

Table 6-20 shows the amount of dollar saving using the service consolidation method. Although the value seems to be insignificant for shorter periods, it makes significant difference for longer periods. For example, the cost reduction for deployment of application 1 on RackSpace is around $11,000/year, which is a significant dollar savings for the customer. Thus, we conclude that our consolidation approach provides a significant cost savings to cloud service deployments.
Table 6.19. Percentage of reduced number of services for tested applications.

<table>
<thead>
<tr>
<th>Application</th>
<th>Reduction in the number of required services (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Federated</td>
</tr>
<tr>
<td>1</td>
<td>12.5</td>
</tr>
<tr>
<td>2</td>
<td>61.53</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.20. Dollar saving per hour for tested applications.

<table>
<thead>
<tr>
<th>Application</th>
<th>Price decrease ($/hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Federated</td>
</tr>
<tr>
<td>1</td>
<td>0.0138</td>
</tr>
<tr>
<td>2</td>
<td>1.54</td>
</tr>
<tr>
<td>3</td>
<td>0.021</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
6.4 Summary

In this chapter we discuss service selection when there are no similar cases to the customer’s application in the case base or the similarity of the closest case is below a specified threshold. We propose to componentize the application into separate deployment entities according to the TOSCA service template. The case-based recommender searches for similar cases to the deployment entities but if no similar case is found in the case base, QuARAMRecommender searches for suitable services among the available services of different providers.

With the increasingly growing number of cloud services, we argued that a brute force search for a suitable cloud service poses significant overhead on the overall response time. We propose to use clustering to decrease the number of comparisons and hence the response time. Experimental results demonstrate that clustering significantly improves the overall response time, with minimal impact on the recommendation precision.

Several factors influence the selection of a suitable service for a deployment entity of an application including: the similarity of the service to the system requirements of the deployment entity, the price of the service and the performance of the service for the deployment entity. The performance is typically based on the type of the deployment entity and its requirements. We take all these factors into consideration when we search for a suitable service. The preferences of customers over different requirements are integrated into the decision making. We compare a proposed linear fitness function and a multi criteria decision making method (TOPSIS) for service selection using 4 different Web applications with varying number of deployment entities and requirements. Based on the results, we select TOPSIS for our service selection.

Selecting services for each deployment entity individually results in having a combination of services from different providers for the deployment of the entire application. Although this may provide the best combination of price, performance and similarity, the current challenges of federated clouds pose difficulty for application deployments due to interoperability and inter-cloud communication issues [133]. One of the challenges in the deployment of an application on a federated cloud is the network communications.
Although network cost on federated clouds is an issue, providers that do not implement DC-to-DC (data center to data center) networks face the same problem when the application’s deployment entities are deployed on different regions. While the feasibility of deployment of an application on federated clouds is in question now, we expect that federated deployment will become more viable in the future as cloud providers are forced to become more open and integrated.

To increase the resource utilization and reduce the overall deployment cost, we propose a service consolidation method based on the knapsack problem and the greedy approximation algorithm. The objective of our consolidation approach is to decrease the price and the number of required services for application deployment, while improving the resource utilization of the selected services. We conduct a set of experiments on our four applications and the results verify that the approach brings significant dollar savings to the application deployment. It also improves the resource utilization by consolidating services and accordingly enhances the overall application performance.
Chapter 7

QuARAM Service Recommender

7.1 Introduction
We explained three challenges in automatic cloud service selection in Chapter 1. The first challenge is to extract the requirements of an application and the preferences of the customer from the specification of the application. In Chapter 4 we explained that using TOSCA [107] as a standard language for the specification of cloud applications makes the automatic extraction of the requirements and preferences possible. In order to incorporate all the requirements and preferences we introduce a set of new Normative Types that customers can use to better specify their preferences and application’s requirements.

The second challenge is the high number of comparisons required to find suitable services due to the large number of the available services and the heterogeneous selection criteria. In Chapter 5, we explained how to solve this challenge by using case-based reasoning to find the most suitable service for the application. In Chapter 6 we address the problem when the recommender cannot find suitable services for the entire application. In this case, we split the application into separate deployment entities and search for suitable services for each deployment entity. We propose to use clustering to overcome the problem associated with the growing size of available offerings (i.e., clustering-based service selection).

The third challenge is the automatic consolidation of the proposed services for the deployment entities of an application. In Chapter 6 we explain how to consolidate services for the deployment entities of an application to maximize the resource utilization, while minimizing the total price of the application deployment.

In this chapter, we introduce QuARAM Service Recommender, a comprehensive platform that addresses all aforementioned challenges to recommend suitable services for cloud application deployments. We validate our recommender with one of the four applications presented in Chapter 6. We also propose a
method for selecting a threshold to determine whether to use case-based reasoning or clustering-based service selection.

7.2 QuARAM Service Recommender

The architecture of QuARAM Service Recommender is illustrated in Figure 7.1. It is composed of 5 main components: requirement/deployment entity extractor, recommendation manager, case-based recommender, service search engine and service consolidator. It also includes 3 knowledge bases: application case base, adaptation case base and providers knowledge base. Components in the dash-line boxes are the ones that may or may not participate in the process of recommendation based on circumstances.

Figure 7.1. The architecture of QuARAM Service Recommender
7.2.1 General Description
The system input is a TOSCA Service Template for an application which is defined using the Normative Types presented in Chapter 4. This template includes the deployment entities of the application, application requirements, and customer preferences. The QuARAM Service Recommender extracts the application requirements, deployment entities and the customer preferences. Then it sends this information to the recommendation manager, which sends a query based on this information to the case-based recommender subsystem. The case-based recommender searches for similar cases and proposes a solution. The recommended solution is given a similarity value that indicates how similar it is to the target case. If the customer is not satisfied with the recommendations or the similarity values of all solutions are less than a specified threshold, the system searches for a suitable service for each deployment entity separately. This search involves both the case-based recommender and the service search engine which searches among all available offerings in the clouds. When the system finds a suitable service for every deployment entity, it consolidates the services to improve the resource utilization and reduce the deployment price. Then, a set of recommendations is suggested to the customer. Based on the selection of the customer (customer selection feedback) a case is added to the case base. Monitoring systems also provide monitoring feedback on the services performance with respect to QoS requirements. This feedback is used to update the case bases.

7.2.2 Components Description
In the following subsections we describe in more detail the components of the QuARAM Service Recommender system and how they interact with each other.

7.2.2.1 Requirement/Deployment Entity Extractor
The requirement/deployment entity extractor component receives the TOSCA Service Template of the application from the customer and extracts the application requirements, customer preferences, and the deployment entities of the application. This is a subcomponent of QuARAM Deployment Engine
component explained in Chapter 3. It provides a couple of .csv documents based on this information and sends them to the recommendation manager, which in turn distributes them to the other components.

7.2.2.2 Case-Based Recommender
This component receives the requirements of an application or one of its deployment entities as the input and returns a list of recommended deployment configurations and the similarity of the cases (i.e., applications in the application case base) used to solve the problem with the target application. It has access to three knowledge bases. The first one is the application case base which contains previously deployed applications/ deployment entities, their requirements and the suitable platform configuration for cloud deployment. The application case base also includes the customer preferences and the SLAs. The second knowledge base is the adaptation case base which incorporates the knowledge about how to adapt a solution so that it fits the features of the target problem (i.e., the new application). The third knowledge base is the providers knowledge base. This knowledge base contains knowledge about the available cloud service offerings, the performance of each service from different perspectives (e.g., computation, I/O, etc.) and the knowledge about the transition from one service to another. The implementation and more details about this component are explained in Chapter 5.

7.2.2.3 Service Search Engine
The service search engine searches the available offerings from cloud providers for a suitable service for the application’s deployment entities, based on the requirements of each entity, customer preferences and the performance of cloud services (more details in Chapter 6). It uses TOPSIS to rank potential services and returns a list of ranked suitable services for the application’s deployment entities.

7.2.2.4 Service Consolidator
This component consolidates the services proposed for all the deployment entities of an application (as described in Chapter 6) to recommend a list of suitable configurations for the application deployment on the cloud. The final configuration must support all the requirements of the application and the preferences
of the customer. The service consolidator takes into consideration the preferences of the customer with respect to the service price and performance. This component uses the greedy approximation algorithm [169] to handle the problem of large search space for consolidating multiple deployment entities into services.

7.2.2.5 Recommendation Manager
The recommendation manager is the core component of the QuARAM Service Recommender that manages and coordinates the various components. The recommendation manager receives the requirements of the customer's application and its deployment entities from the requirement/deployment entity extractor. Then, it sends these requirements to the case-based recommender, which returns a list of recommendations, along with their similarity to the customer’s query and the adapted solutions. Based on the similarity of the retrieved cases, the recommendation manager decides whether to send the recommendations to the customer based on a specified similarity threshold. The top 5 recommendations (or the top n that have similarity above the threshold where n<=5) are sent to the customer to select from if all of the top 5 have similarity above the threshold. If none of the retrieved cases scores a similarity above the threshold, the recommendation manager uses the information of the deployment entities of the application to find a more fine-grained configuration for the application deployment. All proposed configurations are sent to service consolidator, which returns a list of aggregate recommendations that best fit the whole application. Then the recommendation manager sends this list to the customer to choose from.

Case-based recommender, service search engine, service consolidator and recommendation manager constitute recommender component in QuARAM (Chapter 3).

7.2.2.6 Feedback Components
There are two types of feedback in the recommender: customer selection feedback and monitoring system feedback. When a list of recommendations is presented to the customer, one of them is selected by the customer. This feedback from the customer is utilized by the recommendation system in two ways. First it
is used to make a new case for the application case base. Second, this feedback is used for reinforcement learning by the case-based recommender (as described in Chapter 5). The case in the application case base that is selected by the customer gains more “credit” for further references.

The monitoring feedback provides information pertinent to the performance of the applications that are already deployed on the cloud. The recommendation system is alerted of any SLA violation or performance degradation of a service. This information is used by our reinforcement learning technique to decrease the “credit” of the case related to that application for future recommendations.

### 7.2.3 Recommendation Process

The following steps summarize the recommendation process:

1) The customer sends the Service Template of his/her application to the QuARAM Service Recommender.

2) The requirements/deployment entity extractor parses the template and extracts a list of the application’s deployments entities, requirements and customer preferences and sends them to the recommendation manager.

3) The recommendation manager creates a query based on the application requirements, the deployment entities and customer preferences and sends it to the case-based recommender (a). The case-based recommender provides a list of recommendations for the application deployment using the knowledge bases (application case base, adaptation case base, and providers knowledge base). The list of recommendations along with the case similarity are then sent back to the recommendation manager (b).

The recommendation manager decides on the next step based on the similarity of retrieved cases and a specified similarity threshold.

4) The recommendation manager sends to the customer a list of retrieved cases whose similarity is above the threshold. In the context of the QuARAM framework (Chapter 3) the list is sent to the
Deployment engine to pass it to the customer.

4*) If none of the retrieved cases has a similarity above the specified threshold then, for each of the application’s deployment entities the recommendation manager performs the following steps:
   a. Query the case-based recommender for suitable services for the entity.
   b. The case-based recommender returns a list of recommendations for the deployment entity to the recommendation manager.
   c. If the similarity of all retrieved cases are below the specified threshold then recommendation manager queries the service search engine and receives a list of recommendations for the deployment entity. This list can be an empty list which means there is no available service that can fit the entity requirements.
   d. The recommendation manager sends all the application’s deployment entities information and the recommended services to the service consolidator.
   e. The service consolidator returns a more cost efficient configuration for the deployment of the application to the recommendation manager.
   f. The configurations are return to the customer (or the deployment engine in the terminology of QuARAM framework).

This process assumes that the application can be deployed on federated clouds. However, to avoid the current challenges of federated clouds, we further perform the following steps to restrict our recommendations to only a single provider for the entire application.

1) The recommendation manager sends the list of deployment entities of the application and the recommended provider for each entity to the service search engine.

2) The service search engine returns a list of recommendations for the deployment of the application (i.e., all the deployment entities) on each of the providers that are listed by the recommendation manager (as described in Chapter 6).

3) The recommendation manager sends these recommendations (i.e., the deployment entities and the
set of services recommended on each provider) to the service consolidator.

4) The service consolidator makes possible aggregation for services and sends a more cost efficient configuration of services on each candidate provider.

5) The list is then presented to the customer to make a final selection.

7.3 Similarity Threshold

As explained earlier, the results produced by the case-based recommender may not be sufficiently similar to a target application. We use a threshold value for the similarity of the retrieved cases to the target case, so the cases with similarity lower than the threshold are not returned to the customer. If the case-based recommender cannot find any cases with similarity higher than the threshold, the service search engine is responsible for finding the most suitable services.

The threshold could be defined a priori by an expert, but because the case-based recommender improves as the system runs and receives feedback from customers and monitoring systems, the threshold could dynamically change to narrow down the recommendations to the most suitable list. For example, assume that the similarity threshold is initially set to 55% similarity, due to the lack of a large case base. For a new application, the solution of a case with a similarity of 60% may be selected by a customer. The monitoring system’s feedback shows later that the service is not suitable. Then the credit of this case is reduced. Now, a 60% similarity is not sufficient to find a good solution. The expert may change the threshold periodically based on the historical information of service selections. However, defining the similarity threshold manually based on the similarity is challenging due to the continuous changes and the variety of the cases in the case base.

We propose to define the threshold based on the precision of the recommendations. The precision is calculated by Equation (1) using the historical information on the recommendations performed by the case-based recommender. For each recommendation performed by the case-based recommender, the system keeps the similarity of the case selected as a solution. If the solution is selected by the customer, the system
marks the recommendation as a “positive recommendation”, otherwise, marks it as a “false recommendation”. A recommendation also is marked as “false recommendation” if feedback of the monitoring system shows that the selected service is not suitable. The precision is calculated as:

\[
Precision_r = \frac{C}{T_r}
\]

(1)

Where \(C\) is the number of cases for which the recommender suggests the “positive recommendation” on specific range of similarity \((r)\) (e.g., similarity between 71% to 80%), and \(T_r\) is the total number of recommendations on that specific range of similarity.

Table 7.1. Example calculation of precision based on similarity ranges

<table>
<thead>
<tr>
<th>Similarity range ((r))</th>
<th>Total number of recommendations ((T_r)) from 1000 recommendations</th>
<th>Number of “positive recommendations” ((C))</th>
<th>Number of “false recommendations”</th>
<th>(Precision_r = \frac{C}{T_r}) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%-50%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>51%-60%</td>
<td>20</td>
<td>5</td>
<td>15</td>
<td>25%</td>
</tr>
<tr>
<td>61%-70%</td>
<td>250</td>
<td>100</td>
<td>150</td>
<td>40%</td>
</tr>
<tr>
<td>71%-80%</td>
<td>350</td>
<td>300</td>
<td>50</td>
<td>85%</td>
</tr>
<tr>
<td>81%-90%</td>
<td>380</td>
<td>365</td>
<td>15</td>
<td>96%</td>
</tr>
<tr>
<td>91%-100%</td>
<td>100</td>
<td>98</td>
<td>2</td>
<td>98%</td>
</tr>
</tbody>
</table>

For example assume that we use the last 1000 recommendations performed by the case-based recommender to calculate the relation between the precision and the similarity threshold of the suitable recommendations. Table 7.1 demonstrates the calculation of the similarity threshold for a number of similarity ranges. The last column illustrates the precision for each of the similarity ranges. For instance, the last row of the table
means: based on the last 1000 retrievals, among 100 recommendations that has similarity in range of 91%-100%, 98 of them were “positive recommendations” and 2 were “false recommendations”. So the precision is 98% for this range of similarity.

Using this information, the expert can set the threshold as “similarity when the precision of the recommendations are over 95%”. This means that the similarity threshold is 81%.

Figure 7.2 illustrates an example of the relation between the precision and the similarity threshold. We use the application case base and “leave-one-out” validation method to calculate the precision. As it can be concluded from the figure when the similarity of the retrieved cases to the target case is between 80-89%, the precision of the recommendations is ~83% for the most relevant solutions and ~88% for the overestimated. So, the expert can specify that the similarity threshold is the one associated with over 80% precision. Then, the threshold is automatically updated as the cases in the case base change.

![Figure 7.2. The relation between similarity and precision.](image-url)
7.4 Validation of the QuARAM Service Recommender

To validate the framework, we developed a prototype to find the most suitable service(s) for an application. We show the service selection process step-by-step.

To develop a comprehensive prototype that integrates all the components (i.e., case-based recommender, service search engine, service consolidator), we made the following changes into the case-based recommender that we have tested in Chapter 5.

In the case-based recommender system discussed in Chapter 5, we assume that the applications requirements can be satisfied by just one VM. Here, we assume that an application may require more than one VM. The application case base is updated accordingly. The new application case base includes cases with the problem part attributes as follows: application type, application tiers, maximum number of concurrent users, region, response time, security, availability, maximum latency, number of load balancers, number of servers, servers, priority (weight for each of the attributes), and price.

The “Number of servers” attribute contains the number of deployment entities used by the application. The “servers” attribute is an array of attributes for each of the deployment entities in the application. The attributes of a “server” are as follows: memory, CPU power, number of CPU cores, OS, DBMS, NOSQL storage, application server, storage, bandwidth (download, upload), priority.

The solution part of a case in the case base is also updated to contain the following attributes: VMs and configuration. The “VMs” attribute represents the service instances (e.g., “m1.large , Amazon”, “Performance1, GoGrid”). The “configuration” attribute is an array of the servers and the VMs on which they are deployed. Each server and each VM is identified by a unique number (e.g., “1,1”, “2,1”, “3,2” which means that servers 1 and 2 are deployed on VM 1 and server 3 is deployed on VM 2). Figure 7.3 illustrates the schema of the application case base. The “Services” table is our providers knowledge base.

We use application 1 that we presented in Chapter 6 to validate the comprehensive prototype. As a quick recall, the application includes 7 servers with two instances of one of the servers. The Service Template for
this application is presented in Appendix D. Figure 7.4 shows the process flow of the recommendation system and the interactions between the various components. The Service Template of the application is sent to the requirement/deployment extractor (1). The deployment entities of the application, the requirements and the customer preferences are returned in the form of two .csv files, application.csv and servers.csv. Application.csv contains the requirements and customer preferences of the application. Servers.csv includes the requirements and the customer preferences of each deployment entity. These .csv files are also presented in Appendix D.

Figure 7.3. The EER schema of application case base.
The *Recommendation manager* receives these *.csv* files (2) and queries the *case-based recommender* (3). The *case-based recommender* returns the recommendations in the form of a *.csv* file (4), which contains the servers and their candidate services (the “configuration” part of the solution).

Since in this test case there is no other application in the case base that has more than one server, the *case-based recommender* returns an empty list to the *recommendation manager* for the application.

![Figure 7.4. The process flow between the components of the recommender.](image)

The *recommendation manager* then creates queries for each of the deployment entities separately. Figure 7.5 is an example of the *.csv* files for one of the deployments entities that are passed to the *case-based recommender*.

Some values are set to null for attributes such as *application tiers* since they cannot map to a single server.
The queries are sent to the case-based recommender (5) and a list of recommendations is returned to the recommendation manager (6). The recommendation manager uses the solution of the most similar retrieved case as the recommended solution for the deployment entity. The recommended solution from the case-based recommender for the deployment entity in Figure 7.5 is “Amazon, m2xlarge, sim= 52.8%”, where sim is the similarity of the retrieved case to the entity. We use the information from Figure 7.2 on the relation between precision and threshold. We set the precision as >70% (so the threshold should be larger than 80% based on Figure 7.2). Therefore, the recommended solution with the similarity of 52.8% is unacceptable.

The recommendation manager then generates a query to service search engine as illustrated in Figure 7.6 (7). The search engine proposes “Athlantic.net, Xlarge” and returns it to the recommendation manager (8). The steps 5-8 are repeated for each deployment entity. Table 7.2 illustrates the recommended solution for
each of deployment entities of our test application. The column “Recommended by” indicates whether the solution is provided by the case-based recommender (CBR) or the service search engine (SSE).

![Figure 7.6: Query for Service Search Engine (.csv file).](image)

Then recommendation manager sends the list of recommended services for the deployment entities to the service consolidator to consolidate potential services (9). Table 7.3 illustrates the result of the consolidation that are returned to the recommendation manager (10). The total price for the application dropped to $1.98/hr (compare to the non-consolidated services in Table 7.2) for the deployment of the application.

In this example, we assume that federated clouds is a viable option for the application deployment. If the customer prefers the deployment on a single cloud provider, the recommendation manager forwards the requirements of the deployment entities and the list of potential providers (i.e., Microsoft Azure, eApps, Joyent, and Amazon for this application) to the service search engine to find the best service for each entity on each provider. The results are sent to the service consolidator and then to the recommendation manager (11) (results are as shown in Appendix D).
Table 7.2. Recommended services for each deployment entity.

<table>
<thead>
<tr>
<th>Server</th>
<th>Similarity</th>
<th>Provider</th>
<th>name</th>
<th>Mem (GB)</th>
<th>Storage (GB)</th>
<th>Region</th>
<th>Price ($/hr)</th>
<th>OS</th>
<th>extra storage price ($/hr)</th>
<th>Recommended by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>86.59</td>
<td>eApps</td>
<td>?</td>
<td>8</td>
<td>75</td>
<td>US-West</td>
<td>0.182</td>
<td>Windows</td>
<td>0</td>
<td>SSE</td>
</tr>
<tr>
<td>2</td>
<td>86.98</td>
<td>Amazon</td>
<td>r3.large</td>
<td>15.25</td>
<td>32</td>
<td>US-West</td>
<td>0.3</td>
<td>Windows</td>
<td>0</td>
<td>CBR</td>
</tr>
<tr>
<td>3</td>
<td>86.55</td>
<td>Microsoft Azure</td>
<td>Medium(A2)</td>
<td>3.5</td>
<td>60</td>
<td>US-West</td>
<td>0.154</td>
<td>Windows</td>
<td>0</td>
<td>SSE</td>
</tr>
<tr>
<td>4</td>
<td>82.12</td>
<td>eApps</td>
<td>1024</td>
<td>1</td>
<td>15</td>
<td>US-East</td>
<td>0.029</td>
<td>Linux</td>
<td>.0039</td>
<td>CBR</td>
</tr>
<tr>
<td>5</td>
<td>84.81</td>
<td>Amazon</td>
<td>medium</td>
<td>4</td>
<td>0</td>
<td>US-West</td>
<td>0.072</td>
<td>Windows</td>
<td>0</td>
<td>CBR</td>
</tr>
<tr>
<td>6</td>
<td>85.88</td>
<td>Joyent</td>
<td>High Storage</td>
<td>32</td>
<td>7680</td>
<td>US-South</td>
<td>0.923</td>
<td>Windows</td>
<td>0</td>
<td>SSE</td>
</tr>
<tr>
<td>7</td>
<td>88.47</td>
<td>eApps</td>
<td>?</td>
<td>8</td>
<td>75</td>
<td>US-West</td>
<td>0.182</td>
<td>Windows</td>
<td>0</td>
<td>SSE</td>
</tr>
</tbody>
</table>

Total Price = $2.00/hr
Table 7.3. Results of service consolidation for application 1.

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service number</th>
<th>Provider</th>
<th>Price</th>
<th>OS</th>
<th>Servers</th>
<th>Additional disk size</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1</td>
<td>Amazon</td>
<td>0.3</td>
<td>Windows</td>
<td>2</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>539</td>
<td>2</td>
<td>Microsoft Azure</td>
<td>0.154</td>
<td>Windows</td>
<td>3</td>
<td>0</td>
<td>0.154</td>
</tr>
<tr>
<td>539</td>
<td>3</td>
<td>Microsoft Azure</td>
<td>0.154</td>
<td>Windows</td>
<td>3</td>
<td>0</td>
<td>0.154</td>
</tr>
<tr>
<td>1034</td>
<td>4</td>
<td>eApps</td>
<td>0.029</td>
<td>Linux</td>
<td>4</td>
<td>96</td>
<td>0.029</td>
</tr>
<tr>
<td>72</td>
<td>5</td>
<td>Amazon</td>
<td>0.072</td>
<td>Windows</td>
<td>5</td>
<td>0</td>
<td>0.072</td>
</tr>
<tr>
<td>866</td>
<td>6</td>
<td>Joyent</td>
<td>0.923</td>
<td>Windows</td>
<td>6</td>
<td>0</td>
<td>0.923</td>
</tr>
<tr>
<td>1081</td>
<td>7</td>
<td>eApps</td>
<td>0.349</td>
<td>Windows</td>
<td>1,7</td>
<td>0</td>
<td>0.349</td>
</tr>
</tbody>
</table>
7.5 Summary

We recognized three challenges of automatic service selection: 1) Automatic extraction of application requirements and customer preferences, 2) Selection of suitable services from a large pool of available services that is constantly growing. The heterogeneity and the large number of selection criteria pose additional challenge. Consolidation of selected services to maximum the resource utilization, minimize the deployment price, and improve the application performance is required, 3) Adaptation to the dynamic environment of cloud.

We propose several methods and techniques to address each of these challenges in Chapters 4 to 6. In this chapter, we describe QuARAM Service Recommender, a comprehensive platform that addresses the entire process of cloud service selection and integrates all the proposed methods. The platform receives the specification of an application and returns a list of consolidated services for the deployment of the application. We also propose a dynamic method of the recommendation. We demonstrate a step-by-step case study that shows the entire recommendation process.
Chapter 8

Conclusion

The objective of our thesis is to build a self-adaptable cloud service selection system that recommends a suitable list of services to cloud application deployment based on application requirements and the customer preferences.

The motivation of our research comes from the need to develop a platform for cloud application deployment to cope with the growing cloud market. Although the market growth provides economic benefits to customers due to high competition between providers, the similarity of offered services by different providers makes the decision on the best option challenging. The decision also needs to consider the customer preferences over different features. This requires effective evaluation of available services, discovery of services that satisfy customers’ requirements and integration of potential services to reduce deployment cost.

To develop an automatic cloud service selection system, we need to address the following three aspects:

1) Automatic identification and extraction of application requirements and customer preferences: The application requirements and customer preferences should be automatically extracted from the application description document. This includes defining new descriptive features of application requirements that can be used in decision making. These features must be incorporated into the specification of cloud applications.

2) Automatic evaluation, selection and integration of services: Lacking the proper level of experience in the field of cloud services makes it difficult for customers to select the best deployment plan for their applications on the cloud. Moreover, providing them too many options makes it even more sophisticated and confusing. While decision must balance between the application’s requirements and the customer’s goals, desires and constraints, price and performance are important distinguishing factors.
factors between multiple potential services. This wide range of heterogeneous selection criteria and the large search space of available services mandate the need for an automatic and robust approach that simplifies cloud service selection. A comprehensive approach must consider all determining attributes (both quantitative and qualitative) including customer preferences to present the best options to consumers. The approach should be able to handle large size of alternatives (e.g., AHP method used in multiple studies in the literature can not handle large size of alternatives). It should also handle missing values in queries. Medium and large size applications are likely to need multiple services for deployments. In fact, for many applications, it is more cost effective if they are deployed on multiple small services (i.e., split the application into separate deployment entities and deploy each one separately) instead of one large service. In this case, however, service integration is an additional challenge that the system needs to carefully handle. In addition, the deployment region is an important attribute in the application deployment to facilitate communications between its different components. Another important characteristic of the approach that can affect the performance of the service selection system is to be able to memorize the previous queries to reduce the number of searches and the response time.

3) Adaptation to dynamically changing environment: The cloud is highly dynamic and the status of its services is consciously changing. A robust system must have the ability to adapt to these changes to consciously meet the customer satisfaction. Otherwise, the service reputation will be compromised and customers will provide negative feedback and low ratings. The service selection system must incorporate the customer feedback as well as feedback from monitoring systems to provide better recommendations.

In this thesis, we propose a comprehensive cloud service recommendation system that synthesizes several techniques to resolve all these three challenges as the following:

1) Automatic identification and extraction of application requirements and customer preferences: In Chapter 4, we identified a set of features that have the most impact on the service price and
performance using the information on available services advertised on the providers’ websites. These features are used to define the deployment requirements of cloud applications. TOSCA [107] is used as the base standard for cloud application specifications. Customers can add their preferred deployment requirements into their application’s TOSCA Service Template. However, the standard Node Types defined in the basetypes and specificetypes collections are unable to incorporate all the deployment requirements. Therefore, we extend the standard TOSCA Normative Types by defining a set of new reusable and general Node Types and Policy Types with properties for possible deployment requirements. We also integrate the customer’s preferences by adding weights to the properties of Node Types that express the application’s requirements. In addition, we define new policies to incorporate QoS requirements into the application’s Service Template.

2) Automatic evaluation, selection and integration of services: In Chapter 5 we propose to use case-based reasoning for service selection. Case-based reasoning can handle both quantitative and qualitative criteria. The customer preferences are used as the weights when the similarity between the query and cases in the case base is calculated. Case-based reasoning is tolerant to missing values in queries. The large size of case bases are resolved by using distributed case-based reasoning systems [63]. Retaining the previous service selections in its case base, case-based reasoning can use this memory of previous queries to solve new queries without searching among all the available services. Nevertheless, using case-based reasoning is prone to failure when the cases in the case base are insufficient. To address this challenge, in Chapter 6 we propose to use multi-criteria decision making (MCDM) to find suitable services for an application when case-based reasoning fails to find good recommendations (based on a prespecified threshold). We identify the requirements of the deployment entities of an application from the Service Template and use the TOPSIS method [159] to find the most suitable deployment services for each entity. Also, in Chapter 6, we describe our proposed service consolidation method to increase the resource utilization and lower the total deployment price.
3) Adaptation to the dynamically changing cloud environment: In Chapter 3, we explain the *Runtime manager* component in the QuRAM that is responsible to monitor the deployed application, and make minor adjustment to the deployment to adapt to the changes. The *Recommender* component uses the feedback from the monitoring to provide better recommendations. In Chapter 5, we describe how we use the monitoring feedback to improve recommendation through associating each case with a credit score. This score impacts the future use of the cases in service selection (i.e., cases with high credit are likely to be recommended first). The case credit is also affected by the customer feedback. In Chapter 3, we explain that when a list of recommended services are passed to the customer through the *Deployment Engine*, the service selected by the customer is also passed to the *Recommender* to update the “credit” of the selected cases in the case base.

### 8.1 Summary

In the thesis, we propose a self-adaptive IaaS cloud service selection approach using a hybrid of case-based reasoning and TOPSIS ranking and selection method.

In Chapter 2, we provide a brief background and the state-of-art of cloud service selection. The chapter also covers the literature review on case-based reasoning and recommendation.

Chapter 3 presents the QuARAM framework for autonomic QoS-aware cloud application management. QuARAM facilitates the selection of the appropriate cloud provider for cloud application deployment, efficient resources provisioning on the provider’s site, deploying the application on those resources and managing the execution of the application. QuARAM implements the autonomic MAPE loop [34] to ensure continuous obedience to the specified QoS requirements.

Chapter 4 presents our proposed extension to TOSCA *Normative Types* to support automatic identification and extraction of application’s requirements.

In Chapter 5, we propose to use case-based reasoning as a viable option to assist cloud customers in finding the best service for their applications based on previous experiences of other customers and experts. We
investigate different aspects of the case-based reasoning system to cover all dimensions of the service selection decision. We implemented a proof-of-concept prototype to show the feasibility of our approach. We define a set of similarity measures to calculate the similarity of the new application to previously deployed applications on cloud. We accommodate the customer preferences as a set of weights that are used in the similarity calculations. The system adapts previous deployments to better fit the requirements of the new application. We use reinforcement learning to maintain high precision and consistency. Our experimental results show that using case-based reasoning is feasible for cloud service selection.

In Chapter 6, we discuss an alternative method for service selection when the CBR solutions fail to meet the customer requirements. We propose to split applications into deployment entities and search for suitable services for each deployment entity using the TOPSIS [27] method. We argue that a brute force search for a suitable cloud service poses significant overhead on the overall response time. Therefore, we propose to use clustering to decrease the number of comparisons and hence the response time. To increase the resource utilization and reduce the overall deployment cost, we propose a service consolidation method inspired by the knapsack problem and the greedy approximation algorithm.

In Chapter 7, we describe the QuARAM Service Recommender, a comprehensive platform that addresses the entire process of cloud service selection and integrates all our proposed methods. The platform receives the specification of an application as a TOSCA Service Template and returns a list of consolidated services for the application deployment. We also propose a dynamic method to calculate the threshold by which the case-based reasoning solutions are accepted. We demonstrate step-by-step a case study that shows the entire recommendation process.
8.2 Future works

The promising results presented in this thesis inspire us to plan for several extensions to the proposed framework as the following:

1- Our focus in this thesis is on the selection of IaaS service model using case-based reasoning. We plan to check the feasibility of the proposed approach for SaaS and PaaS service models and compare the results with the state of the art methods for these two cloud service models.

2- The deployment of the application’s component as independent deployment entities entails communications between these entities. This communication presents data transfer overhead and incurs unnecessary network traffic. We plan to study the effect of this deployment method on both the network load and application performance. We also plan to quantitatively study how service consolidation may alleviate the overhead on the network and performance.

3- This research adopts a hybrid approach in service selection, where we alternate between case-based reasoning and MCDM based on whether there are similar cases in the case base to support the CBR selection or not. While the case-based reasoning results are on the services that have been used previously for the deployment of applications, MCDM search-based service selection approach can provide new registered services or providers into consideration for application deployment. Since these methods are not parallel, new providers and services are not recognized to be immediately included into the service selection process when CBR is used as service selector in the recommender. We plan to develop techniques that can better update the information in case bases to use in future searches when new services or providers register in the system. An offline simulator can perform this task (i.e., re-select services) for already deployed applications and provides the information to retain in the case base when new services or providers are registered into the system.

4- Based on the studies on service selection using MCDM, we use TOPSIS [159] for the search-based service selection to combine with case-based reasoning. We are interested to study the performance
of other MCDM methods [159] (e.g., ELECTRE\textsuperscript{1}, PROMETHEE\textsuperscript{2}, etc.) and hybrid methods [108] (e.g., combination of EMO\textsuperscript{3} and MCDM methods) to be replaced with TOPSIS.

---

\textsuperscript{1} Elimination Et Choix Traduisant He Realite
\textsuperscript{2} Preference Ranking Organization Method for Enrichment Evaluations
\textsuperscript{3} Evolutionary Multi-objective Optimization
References


169


Intelligence. 170, 16-17 (Nov. 2006), 1175–1192.


[112] Patterson, D. et al. 2002. Efficient similarity determination and case construction techniques for case-based reasoning. 6th European Conference on Advances in Case-Based Reasoning (ECCBR


Appendix A

Service Template for MyWebApplication

The Service Template for example application in Chapter 4 “MyWebApplication”.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Definitions id="MyWebApplicationDefinitions"
  name="MyWebApplication Definitions"
  targetNamespace="http://www.example.com/tosca/ServiceTemplates/MyWebApplication"
  xmlns="http://docs.oasis-open.org/tosca/ns/2011/12"
  xmlns:ns1="http://docs.oasis-open.org/tosca/ns/2011/12/ToscaBaseTypes"
  xmlns:ns2="http://docs.oasis-open.org/tosca/ns/2011/12/ToscaSpecificTypes"
  xmlns:ns3="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes"
  xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/MyWebApplication"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ServiceTemplate id="MyWebApplication" name="MyWebApplication Template">
    <BoundaryDefinitions>
      <Properties>
        <ApplicationTiers>3</ApplicationTiers>
        <MaximumUsers>1000</MaximumUsers>
      </Properties>
      <Interfaces>
        <Interface name="MyWebApplicationBuildPlan">
          <Operation name="initiate">
            <Plan planRef="DeployMyWebApplication"/>
          </Operation>
        </Interface>
      </Interfaces>
      <Policies>
        <PolicyTemplate id="myAvailability" name="required Availability" type="ns4.Availability">
          <Properties>
            <minimumAvailability>99</minimumAvailability>
            <Weight>0.5</Weight>
          </Properties>
        </PolicyTemplate>
      </Policies>
    </BoundaryDefinitions>
    <TopologyTemplate>
      <NodeTemplate id="VmApache" name="VM for Apache" type="ns4:QuARAMSserver">
        <Properties>
          <ns4:QuARAMSserverProperties>
            <NumCPUs>
              <numCPUs>1</numCPUs>
              <Weight>0.5</Weight>
            </NumCPUs>
          </ns4:QuARAMSserverProperties>
        </Properties>
      </NodeTemplate>
    </TopologyTemplate>
  </ServiceTemplate>
</Definitions>
```
<NumCPUs>
  <Memory>
    <memory>1024</memory>
    <Weight>0.5</Weight>
  </Memory>
  <Storage>
    <storage>10</storage>
    <Weight>0.5</Weight>
  </Storage>
  <DataTransfer>
    <download>1</download>
    <upload>10</upload>
    <Weight>0.5</Weight>
  </DataTransfer>
</ns4:QuARAMServerProperties>
</Properties>
<Requirements>
  <Requirement id="VmApache_container" name="container"
    type="ns1:ServerContainerRequirement" />
  <Requirement id="VmApache_volumes" name="volumes"
    type="ns1:VolumeAttachmentRequirement" />
</Requirements>
<Capabilities>
  <Capability id="VmApache_os" name="os"
    type="ns1:OperatingSystemContainerCapability" />
</Capabilities>
</NodeTemplate>

<NodeTemplate id="OsApache" name="OS for Apache"
  type="ns4:QuARAMOperatingSystem">
  <properties>
    <ns4:QuARAMOSProperties>
      <OSType>
        <osType>Linux</osType>
        <Weight>0.3</Weight>
      </OSType>
      <OSVersion>
        <version>Ubuntu</version>
        <Weight>0.3</Weight>
      </OSVersion>
    </ns4:QuARAMOSProperties>
  </properties>
  <Requirements>
    <Requirement id="OsApache_container" name="container"
      type="ns1:OperatingSystemContainerRequirement" />
  </Requirements>
  <Capabilities>
    <Capability id="OsApache_software" name="software"
      type="ns1:SoftwareContainerCapability" />
  </Capabilities>
</NodeTemplate>

<RelationshipTemplate id="OsApache_HostedOn_VmApache"
  name="hosted on" type="ns1:HostedOn">
  <SourceElement ref="OsApache_container"/>
  <TargetElement ref="VmApache_os"/>
</RelationshipTemplate>
<NodeTemplate id="ApacheWebServer" name="Apache Web Server"
    type="ns2:ApacheWebServer">
    <Properties>
        <ns2:ApacheWebServerProperties>
            <httpdport>80</httpdport>
        </ns2:ApacheWebServerProperties>
    </Properties>
    <Requirements>
        <Requirement id="ApacheWebServer_container" name="container"
            type="ns1:SoftwareContainerRequirement" />
    </Requirements>
    <Capabilities>
        <Capability id="ApacheWebServer_webapps" name="webapps"
            type="ns2:ApacheWebApplicationContainerCapability" />
        <Capability id="ApacheWebServer_modules" name="modules"
            type="ns2:ApacheModuleContainerCapability" />
        <Capability id="ApacheWebServer_clients" name="clients"
            type="ns1:HTTPEndpointCapability" />
    </Capabilities>
</NodeTemplate>

<RelationshipTemplate id="Apache_HostedOn_OsApache"
    name="hosted on" type="ns1:HostedOn">
    <SourceElement ref="ApacheWebServer_container"/>
    <TargetElement ref="OsApache_software"/>
</RelationshipTemplate>

<NodeTemplate id="MyWebApp" name="My Web App"
    type="ns3:MyWebApplication">
    <Properties>
        <ns3:MyWebApplicationProperties>
            <AdminUser>admin</AdminUser>
            <AdminPassword>admin</AdminPassword>
            <DataRoot>/home/Mydata</DataRoot>
        </ns3:MyWebApplicationProperties>
    </Properties>
    <Requirements>
        <Requirement id="MyWebApp_phpRuntime" name="phpRuntime"
            type="ns2:PHPRuntimeRequirement" />
        <Requirement id="MyWebApp_database" name="database"
            type="ns2:MySQLDatabaseEndpointRequirement" />
        <Requirement id="MyWebApp_container" name="container"
            type="ns2:ApacheWebApplicationContainerRequirement" />
    </Requirements>
</NodeTemplate>

<RelationshipTemplate id="MyApp_HostedOn_Apache"
    name="hosted on" type="ns1:HostedOn">
    <SourceElement ref="MyWebApp_container"/>
    <TargetElement ref="ApacheWebServer_webapps"/>
</RelationshipTemplate>

<NodeTemplate id="PhpModule" name="PHP Module"
    type="ns2:ApachePHPModule">
    <Requirements>
        <Requirement id="PhpModule_container" name="container"
            type="ns2:ApacheModuleContainerRequirement" />
    </Requirements>
</NodeTemplate>

<Capabilities>
<Capability id="PhpModule_phpApps" name="phpApps" type="ns2:PHPRuntimeCapability" />
</Capabilities>
</NodeTemplate>

<RelationshipTemplate id="PhpModule_HostedOn_Apache" name="hosted on" type="ns2:ApacheModuleHostedOnApache">
  <SourceElement ref="PhpModule_container"/>
  <TargetElement ref="ApacheWebServer_modules"/>
</RelationshipTemplate>

<RelationshipTemplate id="MyApp DependsOn_PhpModule" name="depends on" type="ns1:DependsOn">
  <SourceElement ref="MyWebApp_phpRuntime"/>
  <TargetElement ref="PhpModule_phpApps"/>
</RelationshipTemplate>

<NodeTemplate id="VmMySql" name="VM for MySQL" type="ns4:QuARAMServer">
  <Properties>
    <ns4:QuARAMServerProperties>
      <NumCPUs>
        <numCPUs>1</numCPUs>
      </NumCPUs>
      <Weight>0.5</Weight>
    </NS4:QuARAMServerProperties>
    <CPUType>
      <cpuType>Xeon 2007 1.0-1.2GHZ</cpuType>
    </CPUType>
    <Memory>
      <memory>1024</memory>
      <Weight>0.5</Weight>
    </Memory>
    <Storage>
      <storage>10</storage>
      <Weight>0.5</Weight>
    </Storage>
  </Properties>
  <Requirements>
    <Requirement id="VmMySql_container" name="container" type="ns1:ServerContainerRequirement" />
    <Requirement id="VmMySql_volumes" name="volumes" type="ns1:VolumeAttachmentRequirement" />
  </Requirements>
  <Capabilities>
    <Capability id="VmMySql_os" name="os" type="ns1:OperatingSystemContainerCapability" />
  </Capabilities>
</NodeTemplate>

<NodeTemplate id="OsMySQL" name="OS for MySQL" type="ns4:QuARAMOperatingSystem">
  <properties>
    <ns4:QuARAMOSProperties>
      <OSType>
        <osType>Linux</osType>
      </OSType>
      <Weight>0.3</Weight>
    </OSType>
  </properties>
</NodeTemplate>
<OSVersion>
  <version>Ubuntu</version>
  <Weight>0.3</Weight>
</OSVersion>

<Requirements>
  <Requirement id="OsMySQL_container" name="container"
    type="ns1:OperatingSystemContainerRequirement" />
</Requirements>

<Capabilities>
  <Capability id="OsMySQL_software" name="software"
    type="ns1:SoftwareContainerCapability" />
</Capabilities>
</NodeTemplate>

<RelationshipTemplate id="OsMySQL_HostedOn_VmMySQL"
  name="hosted on" type="ns1:HostedOn">
  <SourceElement ref="OsMySQL_container"/>
  <TargetElement ref="VmMySql_os"/>
</RelationshipTemplate>

<NodeTemplate id="MySql" name="MySQL" type="ns2:MySQL">
  <Properties>
    <ns4:QuARAMOSProperties>
      <DBMSType>Relational</DBMSType>
      <DBMSName>MySQL</DBMSName>
      <Weight>1</Weight>
    </ns4:QuARAMOSProperties>
    <ns2:MySQLProperties>
      <RootPassword>password</RootPassword>
    </ns2:MySQLProperties>
  </Properties>
  <Requirements>
    <Requirement id="MySql_container" name="container"
      type="ns1:SoftwareContainerRequirement" />
  </Requirements>
  <Capabilities>
    <Capability id="MySql_databases" name="databases"
      type="ns2:MySQLDatabaseContainerCapability" />
  </Capabilities>
</NodeTemplate>

<RelationshipTemplate id="MySqlHostedOnOsMySQL"
  name="hosted on" type="ns1:HostedOn">
  <SourceElement ref="MySql_container"/>
  <TargetElement ref="OsMySQL_software"/>
</RelationshipTemplate>

<NodeTemplate id="MyDB" name="MyDatabase" type="ns3:MyApplicationDb">
  <Properties>
    <ns3:MyApplicationDbProperties>
      <DBName>MyAppDB</DBName>
      <DBUser>Sima</DBUser>
      <DBPassword>Kian</DBPassword>
      <mySqlPort>3306</mySqlPort>
    </ns3:MyApplicationDbProperties>
  </Properties>
</NodeTemplate>

<RelationshipTemplate id="MySqlHostedOnOsMySQL"
  name="hosted on" type="ns1:HostedOn">
  <SourceElement ref="MySqlContainer"/>
  <TargetElement ref="OsMySQL_software"/>
</RelationshipTemplate>

<NodeTemplate id="MyDB" name="MyDatabase" type="ns3:MyApplicationDb">
  <Properties>
    <ns3:MyApplicationDbProperties>
      <DBName>MyAppDB</DBName>
      <DBUser>Sima</DBUser>
      <DBPassword>Kian</DBPassword>
      <mySqlPort>3306</mySqlPort>
    </ns3:MyApplicationDbProperties>
  </Properties>
</NodeTemplate>
<Requirements>
  <Requirement id="MyDB_container" name="container"
    type="ns2:MySQLDatabaseContainerRequirement" />
</Requirements>

<Capabilities>
  <Capability id="MyDB_clients" name="clients"
    type="ns2:MySQLDatabaseEndpointCapability" />
</Capabilities>

<RelationshipTemplate id="MyDb_HostedOn_MySql"
  name="hosted on" type="ns3:MyDbHostedOnMySQL">
  <SourceElement ref="MyDB_container"/>
  <TargetElement ref="MySql_databases"/>
</RelationshipTemplate>

<RelationshipTemplate id="MyApp_ConnectsTo_MyDB"
  name="connects to" type="ns3:MyDbConnection">
  <SourceElement ref="MyWebApp_database"/>
  <TargetElement ref="MyDB_clients"/>
</RelationshipTemplate>
</TopologyTemplate>
</ServiceTemplate>
</Definitions>
Appendix B

Explanation of application attributes (features)

**Application Type**: Applications can be of different of compute-intensive, memory-intensive, I/O intensive, etc.

**Memory (MB)**: Indicate the amount of required memory by the deployment entity or application.

**CPU power**: Indicates the CPU capacity or the type of required processor. Examples are: 1.0-1.2 GHz 2007 Opteron, 2007 Xeon.

**Number of CPU cores**: number of processors the application requires.

**OS**: The required operating system for the deployment entity or the application. It contains Type (i.e., Windows, Linux), Version (e.g., XP, Ubuntu) and the Platform (i.e., X32 or X64) of the required operating system.

**DBMS**: the database management system (e.g., MySQL, SQL Server) the entity or application requires.

**NoSQL database**: Type of the NoSQL database management system (e.g., Redis, Cassandra) the entity or application requires.

**Application servers**: List of application servers (e.g., Apache TOMCAT,…) that the application requires.

**Storage (GB)**: Indicates the required storage that a VM is required to have for the application or entity deployment.

**Bandwidth**: The upload and download requirement for the application.

**MaximumNumberOfUsers**: Indicate the maximum number of concurrent users for the application. The service requires to support this number of concurrent users.

**NumberOf Instances**: Indicates number of instances of application/deployment entities that need to be deployed on the cloud. Multiple instances are for fault-tolerance and security reasons.

**NumberOfLoadbalancers**: For applications/deployment entities with multiple instances, loadbalancer is
required and this attribute show the number of loadbalancers the application needs.

**Region**: Indicates the region that the application is required to be deployed in. It could be the datacenter regions (US-West, US-East, Singapore, …) or general regions (e.g., US, Europe, Australia,…) for the deployment.

**ResponseTime (msec)**: Indicates the minimum and average response time the application requires. The value is in millisecond.

**Security**: Indicates the level of security the application requires. The range is between 1 and 10, which 1 indicates low security requirement and 10 indicates that high level of security is required.

**Availability (%)**: Indicates the required availability for the application.

**I/O performance**: Indicates the required I/O performance for the service that the application is deployed on. This attribute can get three values: low, medium and high.

**MaximumLatency**: Indicates the maximum Latency acceptable for the deployed application. It is an integer value that has the maximum latency in milliseconds.

**Price ($)**: Indicates the preferred acceptable price for the deployment of the application.

**Priority**: Contain the customer preferences over all the attributes above. It gets values between 0 and 1.
Appendix C

Service selection results for applications 2 to 4

The results of service selection for applications 2 to 4 using search-based service selection is as follows:

Application 2:

<table>
<thead>
<tr>
<th>Deployment Similarity</th>
<th>Provider</th>
<th>Name</th>
<th>vCPU</th>
<th>Mem (GB)</th>
<th>Storage (GB)</th>
<th>Region</th>
<th>Category</th>
<th>Price ($/hr)</th>
<th>OS</th>
<th>Performance</th>
<th>extra storage price ($/hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 84.652</td>
<td>Microsoft Azure</td>
<td>Medium (A2)</td>
<td>2</td>
<td>3.5</td>
<td>60</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.154</td>
<td>Windows</td>
<td>87</td>
<td>0</td>
</tr>
<tr>
<td>2 81.8897</td>
<td>Microsoft Azure</td>
<td>Medium (A2)</td>
<td>2</td>
<td>3.5</td>
<td>60</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.154</td>
<td>Windows</td>
<td>87</td>
<td>0</td>
</tr>
<tr>
<td>3 81.8897</td>
<td>Microsoft Azure</td>
<td>Medium (A2)</td>
<td>2</td>
<td>3.5</td>
<td>60</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.154</td>
<td>Windows</td>
<td>87</td>
<td>0</td>
</tr>
<tr>
<td>4 78.0692</td>
<td>Microsoft Azure</td>
<td>A5</td>
<td>2</td>
<td>14</td>
<td>135</td>
<td>Asia</td>
<td>'Memory Optimized'</td>
<td>0.31</td>
<td>Windows</td>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>5 78.0692</td>
<td>Microsoft Azure</td>
<td>A5</td>
<td>2</td>
<td>14</td>
<td>135</td>
<td>Asia</td>
<td>'Memory Optimized'</td>
<td>0.31</td>
<td>Windows</td>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>6 78.0692</td>
<td>Microsoft Azure</td>
<td>A5</td>
<td>2</td>
<td>14</td>
<td>135</td>
<td>Asia</td>
<td>'Memory Optimized'</td>
<td>0.31</td>
<td>Windows</td>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>7 75.865</td>
<td>Microsoft Azure</td>
<td>Medium (A2)</td>
<td>2</td>
<td>3.5</td>
<td>60</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.154</td>
<td>Windows</td>
<td>87</td>
<td>0.00274</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>7</td>
<td>75.865</td>
<td>Microsoft Azure</td>
<td>Medium (A2)</td>
<td>2</td>
<td>3.5</td>
<td>60</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.154</td>
<td>Windows</td>
<td>87</td>
</tr>
<tr>
<td>8</td>
<td>78.5714</td>
<td>Amazon</td>
<td>t2.medium</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>Australia</td>
<td>'General Purpose'</td>
<td>0.1</td>
<td>Windows</td>
<td>89</td>
</tr>
<tr>
<td>9</td>
<td>73.2143</td>
<td>Amazon</td>
<td>t2.small</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.036</td>
<td>Windows</td>
<td>10 0</td>
</tr>
<tr>
<td>10</td>
<td>78.4877</td>
<td>Amazon</td>
<td>t2.medium</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>Australia</td>
<td>'General Purpose'</td>
<td>0.1</td>
<td>Windows</td>
<td>89</td>
</tr>
<tr>
<td>11</td>
<td>68.1501</td>
<td>Amazon</td>
<td>r3.4xlarge</td>
<td>16</td>
<td>122</td>
<td>320</td>
<td>US-West</td>
<td>'Memory Optimized'</td>
<td>1.944</td>
<td>Windows</td>
<td>99</td>
</tr>
<tr>
<td>12</td>
<td>84.9851</td>
<td>eApps</td>
<td>?</td>
<td>8</td>
<td>15</td>
<td>150</td>
<td>US-East</td>
<td>'CPU Optimized'</td>
<td>0.349</td>
<td>Windows</td>
<td>76</td>
</tr>
</tbody>
</table>

Average Similarity= 78.29

Average performance= 88.61

Total Price = 4.306
<table>
<thead>
<tr>
<th>Deployment Similarity</th>
<th>Provider</th>
<th>Name</th>
<th>vCPU</th>
<th>Mem (GB)</th>
<th>Storage (GB)</th>
<th>Region</th>
<th>Category</th>
<th>Price ($/hr)</th>
<th>OS</th>
<th>Performance extra storage price ($/hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Microsoft Azure</td>
<td>Medium (A2)</td>
<td>2</td>
<td>3.5</td>
<td>60</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.15</td>
<td>Windows</td>
<td>87</td>
</tr>
<tr>
<td>2</td>
<td>Atlantic.net</td>
<td>Medium</td>
<td>1</td>
<td>2</td>
<td>80</td>
<td>US-East</td>
<td>'General Purpose'</td>
<td>0.04</td>
<td>Windows</td>
<td>61</td>
</tr>
<tr>
<td>3</td>
<td>Atlantic.net</td>
<td>Medium</td>
<td>1</td>
<td>2</td>
<td>80</td>
<td>US-East</td>
<td>'General Purpose'</td>
<td>0.04</td>
<td>Windows</td>
<td>61</td>
</tr>
<tr>
<td>4</td>
<td>Amazon</td>
<td>t2.medium</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.07</td>
<td>Windows</td>
<td>82, 0.006849</td>
</tr>
<tr>
<td>5</td>
<td>eApps</td>
<td>?</td>
<td>2</td>
<td>7.8</td>
<td>25</td>
<td>US-West</td>
<td>'General Purpose'</td>
<td>0.14</td>
<td>Windows</td>
<td>62</td>
</tr>
<tr>
<td>6</td>
<td>eApps</td>
<td>1536</td>
<td>2</td>
<td>1.5</td>
<td>25</td>
<td>US-East</td>
<td>'General Purpose'</td>
<td>0.04</td>
<td>Linux</td>
<td>78, 0.007192</td>
</tr>
<tr>
<td>6</td>
<td>eApps</td>
<td>1536</td>
<td>2</td>
<td>1.5</td>
<td>25</td>
<td>US-East</td>
<td>'General Purpose'</td>
<td>0.04</td>
<td>Linux</td>
<td>78, 0.007192</td>
</tr>
<tr>
<td>6</td>
<td>eApps</td>
<td>1536</td>
<td>2</td>
<td>1.5</td>
<td>25</td>
<td>US-East</td>
<td>'General Purpose'</td>
<td>0.04</td>
<td>Linux</td>
<td>78, 0.007192</td>
</tr>
<tr>
<td>Application 4:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Deployment</strong></td>
<td><strong>Similarity</strong></td>
<td><strong>Provider</strong></td>
<td><strong>name</strong></td>
<td><strong>vCPU</strong></td>
<td><strong>Mem (GB)</strong></td>
<td><strong>Storage (GB)</strong></td>
<td><strong>Region</strong></td>
<td><strong>Category</strong></td>
<td><strong>Price ($/hr)</strong></td>
<td><strong>OS</strong></td>
</tr>
<tr>
<td>1</td>
<td>84.6939</td>
<td>eApps</td>
<td>?</td>
<td>4</td>
<td>8</td>
<td>75</td>
<td>US-West</td>
<td>'CPU Optimized'</td>
<td>0.182</td>
<td>Windows</td>
</tr>
<tr>
<td>2</td>
<td>72.25</td>
<td>Atlantic.net</td>
<td>XXLarge</td>
<td>4</td>
<td>16</td>
<td>200</td>
<td>US-Central</td>
<td>'Memory Optimized'</td>
<td>0.219</td>
<td>Linux</td>
</tr>
<tr>
<td>3</td>
<td>68.50</td>
<td>Atlantic.net</td>
<td>XXLarge</td>
<td>4</td>
<td>16</td>
<td>200</td>
<td>US-Central</td>
<td>'Memory Optimized'</td>
<td>0.219</td>
<td>Linux</td>
</tr>
</tbody>
</table>

Average Similarity = 75.14
Average Performance = 87.33
Total Price = 0.62

Average Similarity = 75.31
Average Performance = 72.7
Total Price = 0.8752
Appendix D

Application 1 specification and service selection results

The Service Template for application 1 is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Definitions id="Application1Definitions" name="Application1 Definitions"
    targetNamespace="http://www.example.com/tosca/ServiceTemplates/Application1"
    xmlns="http://docs.oasis-open.org/tosca/ns/2011/12"
    xmlns:ns1="http://docs.oasis-open.org/tosca/ns/2011/12/ToscaBaseTypes"
    xmlns:ns2="http://docs.oasis-open.org/tosca/ns/2011/12/ToscaSpecificTypes"
    xmlns:ns3="http://queensdsl.weebly.com/QuARAM/ns/2014/1/Application1"
    xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ServiceTemplate id="Application1" name="Application1 Template">
    <BoundaryDefinitions>
      <Properties>
        <ApplicationType>Compute-Intensive</ApplicationType>
        <ApplicationTiers>3</ApplicationTiers>
        <MaximumUsers>2000</MaximumUsers>
      </Properties>
      <Interfaces>
        <Interface name="Application1BuildPlan">
          <Operation name="initiate">
            <Plan planRef="DeployMyApplication"/>
          </Operation>
        </Interface>
      </Interfaces>
      <Policies>
        <PolicyTemplate id="myAvailability" name="required Availability" type="ns4.Availability">
          <Properties>
            <minimumAvailability>99</minimumAvailability>
            <Weight>0.8</Weight>
          </Properties>
        </PolicyTemplate>
        <PolicyTemplate id="MySecurity" name="required Security" type="n4.Security">
          <Properties>
            <minimumSecurity>8</minimumSecurity>
            <Weight>0.5</Weight>
          </Properties>
        </PolicyTemplate>
        <PolicyTemplate id="MyRegion" name="Preferred Region" type="n4.Region">
          <Properties>
            <PreferredRegion>US</PreferredRegion>
            <Weight>0.5</Weight>
          </Properties>
        </PolicyTemplate>
        <PolicyTemplate id="MyPrice" name="Price preference" type="n4.Price">
          
```

194
<Properties>
  <Weight>1</Weight>
</Properties>
</PolicyTemplate>
</Policies>
</BoundaryDefinitions>
</TopologyTemplate>
<!-- deployment entity 1 and its properties -->
<NodeTemplate id="VmServer1" name="VM Server1" type="ns4:QuARAMServer">
  <Properties>
    <ns4:QuARAMServerProperties
      xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes"
      xmlns="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
      <NumCPUs>
        <numCPUs>4</numCPUs>
        <Weight>0.6</Weight>
      </NumCPUs>
      <Memory>
        <memory>6000</memory>
        <Weight>0.6</Weight>
      </Memory>
    </ns4:QuARAMServerProperties>
  </Properties>
  <Capabilities>
    <Capability id="VmServer1_os" name="os"
      type="ns1:OperatingSystemContainerCapability" />
  </Capabilities>
</NodeTemplate>

<NodeTemplate id="OsServer1" name="OS for Server1" type="ns4:QuARAMOperatingSystem">
  <properties>
    <ns4:QuARAMOSProperties
      xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes"
      xmlns="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
      <OSType>
        <osType>Windows</osType>
        <Weight>1</Weight>
      </OSType>
      <OSVersion>
        <version>Server2000</version>
        <Weight>1</Weight>
      </OSVersion>
      <OSPlatform>
        <platform>64</platform>
        <Weight>1</Weight>
      </OSPlatform>
    </ns4:QuARAMOSProperties>
  </properties>
  <Requirements>
    <Requirement id="OsServer1_container" name="container"
      type="ns1:OperatingSystemContainerRequirement" />
  </Requirements>
</NodeTemplate>
<NodeTemplate id="OsServer2" name="OS for Server2"
type="ns4:QuARAMOperatingSystem">
  <properties>
    <ns4:QuARAMOSProperties
      xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes"
      xmlns="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
      <OSType>
        <osType>Windows</osType>
        <Weight>1</Weight>
      </OSType>
      <OSVersion>
        <version>Server2000</version>
        <Weight>1</Weight>
      </OSVersion>
      <OSPlatform>
        <platform>64</platform>
        <Weight>1</Weight>
      </OSPlatform>
    </ns4:QuARAMOSProperties>
  </properties>
  <Requirements>
    <Requirement id="OsServer2_container" name="container"
type="ns1:OperatingSystemContainerRequirement"/>
  </Requirements>
  <Capabilities>
    <Capability id="OsServer2_software" name="software"
type="ns1:SoftwareContainerCapability"/>
  </Capabilities>
</NodeTemplate>

<RelationshipTemplate id="OsServer2_HostedOn_VmServer2"
  name="hosted on" type="ns1:HostedOn">
  <SourceElement ref="OsServer2_container"/>
  <TargetElement ref="VmServer2_os"/>
</RelationshipTemplate>

<!-- deployment entity 3 and its properties -->
<NodeTemplate id="VmServer3" name="VM Server3" type="ns4:QuARAMServer">
  <Properties>
    <ns4:QuARAMServerProperties
      xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes"
      xmlns="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
      <NumCPUs>
        <numCPUs>2</numCPUs>
        <Weight>0.6</Weight>
      </NumCPUs>
      <Memory>
        <memory>8000</memory>
        <Weight>0.6</Weight>
      </Memory>
      <Storage>
        <Storage>20</Storage>
        <Weight>1</Weight>
      </Storage>
    </ns4:QuARAMServerProperties>
  </Properties>
  <Properties>
  </Properties>
</NodeTemplate>
<Storage>
  <NumberOfInstances>
    <NumberOfInstances>2</NumberOfInstances>
  </NumberOfInstances>
</Storage>

<Capabilities>
  <Capability id="VmServer3_os" name="os" type="ns1:OperatingSystemContainerCapability" />
</Capabilities>

<NodeTemplate id="OsServer3" name="OS for Server3">
  <properties>
    <ns4:QuARAMOSProperties xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
      <OSType>
        <osType>Windows</osType>
        <Weight>1</Weight>
      </OSType>
      <OSVersion>
        <version>Server2000</version>
        <Weight>1</Weight>
      </OSVersion>
      <OSPlatform>
        <platform>64</platform>
        <Weight>1</Weight>
      </OSPlatform>
    </ns4:QuARAMOSProperties>
  </properties>
  <Requirements>
    <Requirement id="OsServer3_container" name="container" type="ns1:OperatingSystemContainerRequirement" />
  </Requirements>
  <Capabilities>
    <Capability id="OsServer3_software" name="software" type="ns1:SoftwareContainerCapability" />
  </Capabilities>
</NodeTemplate>

<RelationshipTemplate id="OsServer3_HostedOn_VmServer3" name="hosted on">
  <SourceElement ref="OsServer3_container"/>
  <TargetElement ref="VmServer3_os"/>
</RelationshipTemplate>

<!-- deployment entity 4 and its properties -->
<NodeTemplate id="VmServer4" name="VM Server4" type="ns4:QuARAMServer">
  <Properties>
    <ns4:QuARAMServerProperties xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
      ...
    </ns4:QuARAMServerProperties>
  </Properties>
</NodeTemplate>
<Storage>
  <Storage>111</Storage>
  <Weight>1</Weight>
</Storage>
</ns4:QuARAMServerProperties>
</Properties>
<Capabilities>
 <Capability id="VmServer4_os" name="os"
 type="ns1:OperatingSystemContainerCapability" />
</Capabilities>
</NodeTemplate>
<NodeTemplate id="OsServer4" name="OS for Server4"
type="ns4:QuARAMOperatingSystem">
 <properties>
  <ns4:QuARAMOSProperties
   xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes"
   xmlns="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
   <OSType>
    <osType>Linux</osType>
    <Weight>1</Weight>
  </OSType>
 </ns4:QuARAMOSProperties>
 </properties>
 <Requirements>
  <Requirement id="OsServer4_container" name="container"
   type="ns1:OperatingSystemContainerRequirement" />
 </Requirements>
<Capabilities>
 <Capability id="OsServer4_software" name="software"
   type="ns1:SoftwareContainerCapability" />
</Capabilities>
</NodeTemplate>
<RelationshipTemplate id="OsServer4_HostedOn_VmServer4"
 name="hosted on" type="ns1:HostedOn">
 <SourceElement ref="OsServer4_container"/>
 <TargetElement ref="VmServer4_os"/>
</RelationshipTemplate>

<!-- deployment entity 5 and its properties -->
<NodeTemplate id="VmServer5" name="VM Server5" type="ns4:QuARAMServer">
 <Properties>
  <ns4:QuARAMServerProperties
   xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes"
   xmlns="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
   <NumCPUs>
    <numCPUs>2</numCPUs>
    <Weight>0.6</Weight>
  </NumCPUs>
  <Memory>
    <memory>3000</memory>
    <Weight>0.6</Weight>
  </Memory>
</ns4:QuARAMServerProperties>
</Properties>
</NodeTemplate>
<ns4:QuARAMServerProperties>
</Properties>
<Capabilities>
  <Capability id="VmServer5_os" name="os"
  type="ns1:OperatingSystemContainerCapability" />
</Capabilities>
</NodeTemplate>

<NodeTemplate id="OsServer5" name="OS for Server5"
  type="ns4:QuARAMOperatingSystem">
  <properties>
    <ns4:QuARAMOSProperties
     xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes"
     xmlns="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
      <OSType>
        <osType>Windows</osType>
        <Weight>1</Weight>
      </OSType>
      <OSVersion>
        <version>Server2000</version>
        <Weight>1</Weight>
      </OSVersion>
      <OSPlatform>
        <platform>64</platform>
        <Weight>1</Weight>
      </OSPlatform>
    </ns4:QuARAMOSProperties>
  </properties>
  <Requirements>
    <Requirement id="OsServer5_container" name="container"
    type="ns1:OperatingSystemContainerRequirement" />
  </Requirements>
  <Capabilities>
    <Capability id="OsServer5_software" name="software"
    type="ns1:SoftwareContainerCapability" />
  </Capabilities>
</NodeTemplate>

<RelationshipTemplate id="OsServer5_HostedOn_VmServer1"
  name="hosted on" type="ns1:HostedOn">
  <SourceElement ref="OsServer5_container"/>
  <TargetElement ref="VmServer5_os"/>
</RelationshipTemplate>

<NodeTemplate id="DBMSServer5" name="DBMS for Server5"
  type="ns2:SQLServer">
  <Properties>
    <ns4:QuARAMDBMSProperties
     xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes"
     xmlns="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
      <DBMSType>Relational</DBMSType>
      <DBMSName>SQLServer2008</DBMSName>
      <Weight>1</Weight>
    </ns4:QuARAMDBMSProperties>
  </Properties>
</NodeTemplate>
<Properties>
  <ns4:QuARAMServerProperties
    xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
    <NumCPUs>
      <numCPUs>8</numCPUs>
      <Weight>0.8</Weight>
    </NumCPUs>
    <Memory>
      <memory>32000</memory>
      <Weight>0.8</Weight>
    </Memory>
    <Storage>
      <Storage>50</Storage>
      <Weight>0.6</Weight>
    </Storage>
  </ns4:QuARAMServerProperties>
</Properties>

<Capabilities>
  <Capability id="VmServer6_os" name="os" type="ns1:OperatingSystemContainerCapability" />
</Capabilities>

<NodeTemplate id="VM Server6" name="VM Server6" type="ns4:QuARAMServer">
  <Properties>
    <ns4:QuARAMServerProperties
      xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
      <NumCPUs>
        <numCPUs>8</numCPUs>
        <Weight>0.8</Weight>
      </NumCPUs>
      <Memory>
        <memory>32000</memory>
        <Weight>0.8</Weight>
      </Memory>
      <Storage>
        <Storage>50</Storage>
        <Weight>0.6</Weight>
      </Storage>
    </ns4:QuARAMServerProperties>
  </Properties>
</NodeTemplate>

<OsServer6 name="OS for Server6" type="ns4:QuARAMOperatingSystem">
  <properties>
    <ns4:QuARAMOSProperties
      xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
      <OSType>
        <osType>Windows</osType>
        <Weight>1</Weight>
      </OSType>
      <OSVersion>
        <version>Server2000</version>
        <Weight>1</Weight>
      </OSVersion>
      <OSPlatform>
        <platform>64</platform>
    </ns4:QuARAMOSProperties>
  </properties>
</OsServer6>
<ns4:QuARAMOSProperties>
  <Property name="DBMSType" type="ns1:OperatingSystemContainerRequirement"/>
  <Property name="DBMSName" type="ns1:SoftwareContainerCapability"/>
</ns4:QuARAMOSProperties>

<ns4:QuARAMDBMSProperties>
  <Property name="DBMSType" type="ns1:OperatingSystemContainerRequirement"/>
  <Property name="DBMSName" type="ns1:SoftwareContainerCapability"/>
</ns4:QuARAMDBMSProperties>

<ns4:QuARAMServerProperties>
  <Property name="NumCPUs" type="ns1:OperatingSystemContainerRequirement"/>
  <Property name="Memory" type="ns1:SoftwareContainerCapability"/>
</ns4:QuARAMServerProperties>
</Memory>
</ns4:QuARAMServerProperties>
</Properties>
<Capabilities>
<Capability id="VmServer7_os" name="os" type="ns1:OperatingSystemContainerCapability" />
</Capabilities>
<NodeTemplate id="OsServer7" name="OS for Server7" type="ns4:QuARAMOperatingSystem">
<properties>
<ns4:QuARAMOSProperties xmlns:ns4="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes" xmlns="http://queensdsl.weebly.com/QuARAM/ns/2014/1/QuARAMTypes">
<OSType>
<osType>Windows</osType>
<Weight>1</Weight>
</OSType>
<OSVersion>
<version>Server2000</version>
<Weight>1</Weight>
</OSVersion>
<OSPlatform>
<platform>64</platform>
<Weight>1</Weight>
</OSPlatform>
</ns4:QuARAMOSProperties>
</properties>
<Requirements>
<Requirement id="OsServer7_container" name="container" type="ns1:OperatingSystemContainerRequirement" />
</Requirements>
<Capabilities>
<Capability id="OsServer7_software" name="software" type="ns1:SoftwareContainerCapability" />
</Capabilities>
</NodeTemplate>
<RelationshipTemplate id="OsServer7_HostedOn_VmServer7" name="hosted on" type="ns1:HostedOn">
<SourceElement ref="OsServer7_container"/>
<TargetElement ref="VmServer7_os"/>
</RelationshipTemplate>
</TopologyTemplate>
</ServiceTemplate>
</Definitions>
The .csv files receive by recommendation manager from the application requirement/deployment entity extractor component are as follows:

<table>
<thead>
<tr>
<th>application.csv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provider,AppType,Apptier,MaxUser,Region,ResponseTime,Security,Availability,Latency,LoadBalancer,Price,RegionWeight,ResponseTimeWeight,SecurityWeight,AvailabilityWeight,LatencyWeight,LoadBalancerWeight,PriceWeight</td>
</tr>
<tr>
<td>Compute-intensive,3,2000,US,,8,99,,,,0.5,0.5,0.8,,,,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>servers.csv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number,vCPU,CPUpower,Mem,Storage,OSType,OSPlatform,OSVersion,DBMS,NoSQL,AppServer,Download,Upload,vCPUWeight,CPUpowerWeight,MemWeight,StorageWeight,OSTypeWeight,OSPlatformWeight,OSVersionWeight,DBMSWeight,NoSQLWeight,AppServerWeight,DownloadWeight,UploadWeight</td>
</tr>
<tr>
<td>1,4,,6,0,Windows,64,Server2008,SQLServer2008,,0.6,,0.6,,1,1,1,1,,</td>
</tr>
<tr>
<td>2,2,,8,0,Windows,64,Server2008,,,,0.6,,0.6,,1,1,1,1,</td>
</tr>
<tr>
<td>3,2,,1,20,Windows,64,Server2008,,,,0.6,,0.6,,1,1,1,1,</td>
</tr>
<tr>
<td>3,2,,1,20,Windows,64,Server2008,,,,0.6,,0.6,1,1,1,1,</td>
</tr>
<tr>
<td>4,0,,0,111,Linux,,,,,,,,0,,0,1,1,1,1,1,1,</td>
</tr>
<tr>
<td>5,2,,3,0,Windows,64,Server2008,SQLServer2008,,0.6,,0.6,,1,1,1,1,</td>
</tr>
<tr>
<td>6,8,,32,50,Windows,64,Server2008,SQLServer2008,,,,0.8,,0.8,0.6,1,1,1,1,</td>
</tr>
<tr>
<td>7,4,,8,0,Windows,64,Server2008,,,,0.8,,0.8,1,1,1,1,</td>
</tr>
</tbody>
</table>
Service selection results on potential providers after consolidation is as follows:

**Microsoft Azure:**

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service number</th>
<th>Provider</th>
<th>Price</th>
<th>OS</th>
<th>Servers</th>
<th>Additional disk size</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>421</td>
<td>0</td>
<td>Microsoft Azure</td>
<td>0.592</td>
<td>Windows 1,3,5</td>
<td>0</td>
<td>0.592</td>
<td></td>
</tr>
<tr>
<td>737</td>
<td>1</td>
<td>Microsoft Azure</td>
<td>0.31</td>
<td>Windows 2</td>
<td>0</td>
<td>0.31</td>
<td></td>
</tr>
<tr>
<td>539</td>
<td>3</td>
<td>Microsoft Azure</td>
<td>0.154</td>
<td>Windows 3</td>
<td>0</td>
<td>0.154</td>
<td></td>
</tr>
<tr>
<td>432</td>
<td>4</td>
<td>Microsoft Azure</td>
<td>0.02</td>
<td>Linux 4</td>
<td>91</td>
<td>0.026233</td>
<td></td>
</tr>
<tr>
<td>469</td>
<td>6</td>
<td>Microsoft Azure</td>
<td>1.2</td>
<td>Windows 6</td>
<td>0</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>528</td>
<td>7</td>
<td>Microsoft Azure</td>
<td>0.6</td>
<td>Windows 7</td>
<td>0</td>
<td>0.6</td>
<td></td>
</tr>
</tbody>
</table>

**Joyent:**

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service number</th>
<th>Provider</th>
<th>Price</th>
<th>OS</th>
<th>Servers</th>
<th>Additional disk size</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>839</td>
<td>0</td>
<td>Joyent</td>
<td>1.02</td>
<td>Windows 1,2,3</td>
<td>0</td>
<td>1.02</td>
<td></td>
</tr>
<tr>
<td>822</td>
<td>4</td>
<td>Joyent</td>
<td>0.02</td>
<td>Linux 4</td>
<td>91</td>
<td>0.02536</td>
<td></td>
</tr>
<tr>
<td>866</td>
<td>6</td>
<td>Joyent</td>
<td>0.923</td>
<td>Windows 6</td>
<td>0</td>
<td>0.923</td>
<td></td>
</tr>
<tr>
<td>839</td>
<td>7</td>
<td>Joyent</td>
<td>1.02</td>
<td>Windows 3,7</td>
<td>0</td>
<td>1.02</td>
<td></td>
</tr>
</tbody>
</table>

**Amazon:**

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service number</th>
<th>Provider</th>
<th>Price</th>
<th>OS</th>
<th>Servers</th>
<th>Additional disk size</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>0</td>
<td>Amazon</td>
<td>0.752</td>
<td>Windows 1,2,3</td>
<td>0</td>
<td>0.752</td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>3</td>
<td>Amazon</td>
<td>0.1</td>
<td>Windows 3</td>
<td>20</td>
<td>0.10274</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>4</td>
<td>Amazon</td>
<td>0.026</td>
<td>Linux 4</td>
<td>111</td>
<td>0.041205</td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>5</td>
<td>Amazon</td>
<td>0.1</td>
<td>Windows 5</td>
<td>0</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>6</td>
<td>Amazon</td>
<td>1.08</td>
<td>Windows 6</td>
<td>0</td>
<td>1.08</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>7</td>
<td>Amazon</td>
<td>0.6</td>
<td>Windows 7</td>
<td>0</td>
<td>0.6</td>
<td></td>
</tr>
</tbody>
</table>