Cloud Infrastructure Services Selection and Evaluation

Qian Zhang

A thesis submitted for the degree of
Doctor of Philosophy
of The Australian National University

August 2021

Except where otherwise indicated, this thesis is my own original work.

Qian Zhang
20 August 2021

List of Publications

Primary publications

1. Miranda Zhang et al. "A Declarative Recommender System for Cloud Infrastructure Services Selection". In: Economics of Grids, Clouds, Systems, and Services. Ed. by Kurt Vanmechelen, Jörn Altmann, and Omer F. Rana. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 102–113. isbn: 978-3-642-35194-5

2. Miranda Zhang et al. "An Infrastructure Service Recommendation System for Cloud Applications with Real-time QoS Requirement Constraints". In: IEEE Systems Journal PP.99 (2015), pp. 1–11. issn: 1932-8184. doi: 10.1109/JSYST.2015.2427338. url: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7124433

3. Miranda Zhang et al. "An ontology-based system for Cloud infrastructure services' discovery". In: CollaborateCom. 2012. url: http://ieeexplore.ieee.org/document/6450944/

4. Miranda Zhang et al. "Investigating decision support techniques for automating Cloud service selection". English. In: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings. IEEE, Dec. 2012, pp. 759– 764. isbn: 978-1-4673-4510-1. doi: 10.1109/CloudCom.2012.6427501. url: http://ieeexplore.ieee.org/document/6427501/

5. Miranda Zhang et al. "Investigating Techniques for Automating the Selection of Cloud Infrastructure Services". en. In: INTERNATIONAL JOURNAL OF NEXT-GENERATION COMPUTING 4.3 (2013). issn: 0976-5034. url: http://perpetualinnovation.net/ojs/index.php/ijngc/article/view/227

6. Qian Zhang, Armin Haller, and Qing Wang. "CoCoOn: Cloud Computing Ontology for IaaS Price and Performance Comparison". In: ISWC2019: The 18th International Semantic Web Conference. Auckland: Springer International Publishing, 2019, pp. 325–341. isbn: 978-3-030-30796-7. url: https://link.springer.com/chapter/10.1007/978-3-030-30796-7_21

Secondary publications

1. Dimitrios Georgakopoulos et al. "Discovery-Driven Service Oriented IoT Architecture". In: 2015 IEEE Conference on Collaboration and Internet Computing (CIC). IEEE, Oct. 2015, pp. 142–149. isbn: 978-1-5090-0089-0. doi: 10.1109/CIC.2015.34. url: http://ieeexplore.ieee.org/document/7423076/

2. Zheng Li et al. "Early Observations on Performance of Google Compute Engine for Scientific Computing". In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science. Vol. 1. IEEE, Dec. 2013, pp. 1–8. isbn: 978-0-7695-5095-4. doi: 10.1109/CloudCom.2013.7. url:http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6753771

3. Zheng Li et al. "Towards understanding the runtime configuration management of do-it-yourself content delivery network applications over public clouds". In: Future Generation Computer Systems 37 (July 2014), pp. 297–308. issn: 0167739X. doi: 10.1016/j.future.2013.12.019. url: http://www.sciencedirect.com/science/article/pii/S0167739X13002835

4. Rajiv Ranjan et al. "City Data Fusion: Sensor Data Fusion in the Internet of Things". In: Int. J. Distrib. Syst. Technol. 7.1 (Jan. 2016), pp. 15–36. issn: 1947-3532. doi: 10.4018/IJDST.2016010102. url: http://dx.doi.org/10.4018/IJDST.2016010102

5. Meisong Wang et al. "An Overview of Cloud Based Content Delivery Networks: Research Dimensions and State-of-the-Art". In: Transactions on Large-Scale Data-and Knowledge-Centered Systems XX. Springer Berlin Heidelberg, 2015, pp. 131–158. url: https://link.springer.com/chapter/10.1007%2F978-3-662-46703-9_6

6. Inger Mewburn et al. "Do transferable skills programs really add value?" In: 13th Quality in Postgraduate Research Conference (QPR2018). 2018

Acknowledgments

Throughout the writing of this dissertation, I have received a great deal of assistance. I would like to thank ANU, CSIRO, Australian-French Association for Science and Technology (AFAS) and IEEE for providing me with the research funding. First and foremost, I would like to thank my primary supervisor, Qing Wang, who helped me overcome lots of obstacles in my academic and life journey. She is exceptionally caring, supportive and provided directions and tools for completing my thesis. Also, I would like to thank Rajiv Ranjan, Alistair Rendell and Armin Haller for being on my thesis committee, who provided many thoughtful advise during various stages of my research. I also received academic help from many other researchers, among whom I especially would like to thank Dimitrios Georgakopoulos, Peter Strazdins and Surya Nepal. Besides, I am incredibly grateful to Peter Kanowski, who is the housemaster of my accommodation. He is extremely responsible and attentive during my prolonged stay. I also learnt a lot from the other co-authors and editors, who are experts in their fields.

Apart from my primary research experience in Australia, I also have a valuable visiting scholar experience in Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, Beijing under the guidance of Lizhe Wang, who is also one of my co-authors. Moreover, I would like to thank Laurent Lefevre for providing the opportunities for me to visit the AVALON (Algorithms and Software Architectures for Service Oriented Platforms) team from Inria and the LIP Laboratory in Ecole Normale Superieure de Lyon, France.

Moreover, I would like to thank my beloved girlfriend Shiqin Huo, for all her love and support. For numerous times, she cheered me up with her good sense of humor. For numerous times, she saved me from procrastination and game addiction with her persistent encouragement and constant reminder.

Additionally, I would like to acknowledge the talented colleagues whom I shared offices with, and we had many stimulating conversations together. Moreover, I would like to thank my parents for their wise counsel and sympathetic ear. Furthermore, there are my friends who were of great support, providing happy distractions to rest my mind outside of my research. Finally, I would like to thank everyone who haven't been mention explicitly, including the considerate college and graduate house and ANU administration staffs for their great support.

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

"To begin well is common; To end well is rare indeed."

The Ancient Chinese Classic of Poetry (Shi-jing, published c. 600 BC) [37].
Translated by Arthur Waley.

Abstract

The proliferation of cloud computing has revolutionized the hosting and delivery of Internet-based application services. However, with the constant increase of new cloud services almost every month by both large corporations (e.g., Amazon Web Service and Microsoft Azure) and small companies (e.g. Rackspace and FlexiScale), the selection scenarios become more and more complex. This is aggregated by confusing and ambiguous terminology, and non-standardized interfaces. This is challenging for decision-makers such as application developers and chief information officers as they are overwhelmed by various choices available.

In this thesis, I will address the above challenges by developing several techniques. Firstly, I define the Cloud Computing Ontology (CoCoOn). CoCoOn defines concepts, features, attributes and relations of Cloud infrastructure services. Secondly, I propose a service selection method that adopts an analytic hierarchy process (AHP)-based multi-criteria decision-making technique. It allows users to define multiple design-time constraints like renting costs, data centre locations, service features and real-time constraints, such as end-to-end message latency and throughput. These constraints are then matched against our model to compute the possible best-fit combinations of cloud Infrastructure, offered as a Service (IaaS). Pairwise comparisons are used to help users determine a relative preference among a pool of nonnumerical attributes. Criteria that are taken into consideration during comparison can be grouped into two categories: the benefit and the cost. Based on this, I define a cost–benefit-ratio-based evaluation function to calculate the ranking for Cloud service options. Thirdly, I suggest a theory-based queuing approach for estimating IaaS usage. Queuing theory is a widely studied method in QoS modelling and optimization. From the infrastructure system administrator perspective, I explore serveral ways to apply the queuing theory model to estimate the best-fit resource allocation for achieving the desired SLA. Finally, the thesis shows how an integrated system, CloudRecommender, can be built from our proposed approaches.

Keywords: Cloud computing, Semantic Technology, Semantic Web, Recommender System, Operations Research

Contents

List of Figures

List of Tables

List of Abbreviations

AHP	**A**nalytic **H**ierarchy **P**rocess
API	**A**pplication **P**rogram **I**nterface
AWS	**A**mazon **W**eb **S**ervice
CoCoOn	**Co**ud **Co**mputing **On**tology
EC2	**E**lastic **C**ompute **C**loud unit (AWS)
GUI	**G**raphical **U**ser **I**nterface
IaaS	**I**nfrastructure **a**s **a** **S**ervice
MADM	**M**ulti **A**ttribute **D**ecision **M**aking
MCDM	**M**ulti **C**riteria **D**ecision **M**aking
ML	**M**achine **L**earning
MODM	**M**ulti **O**bjective **D**ecision **M**aking
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
NLP	**N**atural **L**anguage **P**rocessing
OWL	**W**eb **O**ntology **L**anguage
PaaS	**P**latform **a**s **a** **S**ervice
QoS	**Q**uality **o**f **S**ervice
QUDT	**Q**uantity Kinds, **U**nits of Measure, **D**imensions and **D**ata Types
REST	**RE**presentational **S**state **T**ransfer

RDF	**R**esource **D**escription **F**ramework
RDFS	**R**esource **D**escription **F**ramework **S**chema
SaaS	**S**oftware **a**s **a** **S**ervice
SAW	**S**imple **A**dditive **W**eighting
SOA	**S**ervice **O**riented **A**rchitecture
SOAP	**S**imple **O**bject **A**ccess **P**rotocol
SPARQL	**P**rotocol **A**nd **R**DF **Q**uery **L**anguage
SQL	**S**tructured **Q**uery **L**anguage
TFIDF	**T**erm **F**requency **I**nverse **D**ocument **F**requency
TOPSIS	**T**echnique for **O**rder **P**reference by **S**imilarity to **I**dentical **S**olution
Turtle	**T**erse **R**DF **T**riple **L**ang**u**ag**e**
UDDI	**U**niversal **D**escription, **D**iscovery, and **I**ntegration
VIKOR	**V**IseKriterijumska **O**ptimizacija **I** **K**ompromisno **R**esenje (Serbian)
	Means: Multicriteria Optimization and Compromise Solution
VM	**V**irtual **M**achine
WPM	**W**eighted **P**roduct **M**ethod
WSDL	**W**eb **S**ervices **D**escription **L**anguage

Chapter 1

Introduction

According to the National Institute of Standards and Technology's (NIST) definition [133], "Cloud computing is a model for enabling ubiquitous, convenient and on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management efforts or service provider interaction."

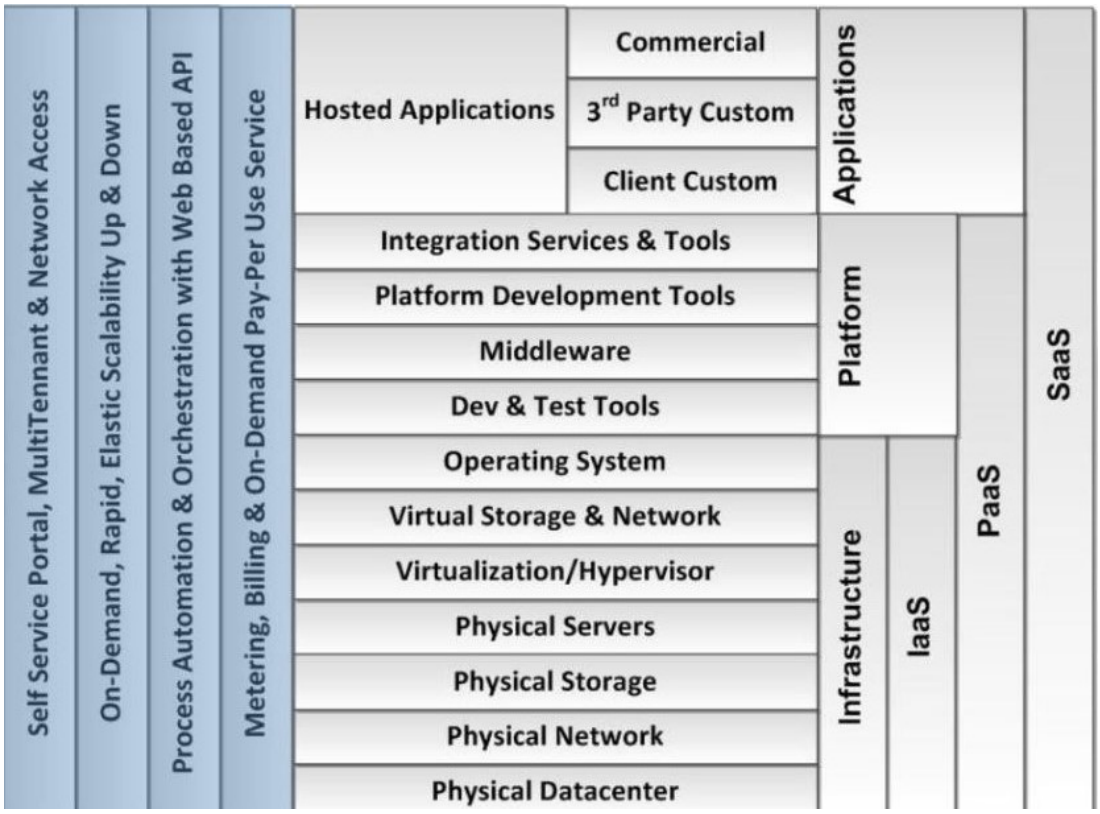


Figure 1.1: Cloud Computing Architecture [89]

The essential characteristics of Cloud computing include self service portal, multi-tenant and network access, on-demand rapid elastic scalability, process automation and orchestration with web based API and pay per user billing. The three Cloud

service models are Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Figure 1.1 illustrates the architecture and general components of the Cloud.

Within the IaaS layer, hypervisor software [20] enables flexible provisioning of virtual machines, storage and network services. Virtual machines sometimes are referred to as servers or compute services. Some Cloud providers include the load-balancing service or the load-balancer as a separate component.

Built on the IaaS layer, additional middle-wares, tools or integration services can be provided as PaaS. Apps can also be hosted on IaaS and PaaS. Those software applications available to the customer with the pay-as-you-go model are SaaS. The SaaS layer focuses on application services by making use of services provided by the other layers. PaaS/SaaS services can be developed and provided by third-party service providers different from the IaaS providers. Some examples of SaaS providers are Salesforce.com, Microsoft Office 365, Oracle CRM On Demand and Google Apps. Some examples of PaaS providers are Google App Engine, Force.com and Heroku. Some examples of IaaS providers are Amazon Web Service [10], Google Compute Engine [73], Microsoft Azure, Rackspace [165], Alibaba Cloud [7], IBM Softlayer [95], FlexiScale [157] and DataCentred [50].

In the Cloud computing model, users have access to services according to their requirements without the need to know where the services are hosted or how they are delivered. Cloud computing embraces an elastic paradigm where applications establish on-demand interactions with services to satisfy the required Quality of Service (QoS) including response time and throughput. As Cloud continues to revolutionise applications in academia, industry, government, and many other fields, the transition to this efficient and flexible platform presents serious challenges at both theoretical and practical levels [208]. As large ISPs (Internet Service Providers) use multiple network providers, failures by a single network provider will not take them off the air. The solution to extreme high availability is multiple Cloud computing providers [15]. To have no single point of failure, applications need to be provisioned in different geographic locations, aka cloud regions. However, no cloud can guarantee 100% availability. When availability is critical, distribute servers/services across multiple providers is needed. An international enterprise may also be forced to used multiple Cloud providers due to political, regulatory, strategically reasons, e.g. new accusation of a business with dependency on a different cloud and new market in a country that forbids data to be stored in foreign cloud company. However, selecting and composing the right services that meet up with requirements is a challenging problem. Consider an example of a medium-scale enterprise that would like to move its enterprise applications to the Cloud. Multiple providers offer infrastructure services with various configurations, e.g. Amazon, Microsoft Azure, Google, Alibaba, among many others. With varied options, enterprises are facing a complicated task when trying to decide the right service for their needs. In this thesis, we are concerned with how to simplify the selection and comparison of a set of infrastructure service offerings for hosting enterprise applications and corresponding datasets, meanwhile meeting multiple criteria, such as specific configurations and costs, emanating from

the enterprise's QoS needs. This is a complicated problem for the enterprise and needs to be addressed.

1.1 Challenges in Cloud Selection and Comparison

The Cloud computing paradigm is shifting computing from in-house managed hardware and software resources to virtualized Cloud-hosted services. Cloud computing assembles large networks of virtualized services: infrastructure services (e.g., compute, storage, network) and software services (e.g., databases, message queuing systems, monitoring systems, load-balancers)

Cloud providers give users the option to deploy their applications over a pool of virtually infinite services with practically no capital investment and with modest operating costs proportional to the actual uses. Elasticity, cost benefits and abundance of resources motivate many organisations to migrate their enterprise applications to the Cloud. Although the Cloud offers the opportunity to focus on revenue growth and innovation, decision-makers (e.g., CIOs, scientists, developers, engineers) are confronted with the complexity of choosing the appropriate service, delivery model and infrastructure.

Numerous information technology vendors claim to offer applications, storage, and computation resources as cloud hosting services. As a result, an exceeding number of competing services are available to users. In such a context, the migration of applications (e.g., multi-layered enterprise applications, scientific experiments, video-on-demand streaming applications) to the Cloud demands selecting the best mix of services from an abundance of possibilities. Any such Cloud service selection decision has to cater to many conflicting criteria while ensuring the fulfilment of QoS requirements, e.g. minimize cost and network latency while maximizing throughput, storage space, CPU power (for analytical programs to process more jobs or do it faster) and RAM (bigger buffer size). QoS requirements for different applications also are varied. For example, scientific experiments need to meet deadlines, thereby time/duration being constrained. On the other hand, video-on-demand streaming applications need to satisfy streaming latency and resolution requirements.

Naturally, it is challenging for users to select the right services that meet their QoS requirements in the service cycle from selection and deployment to orchestration. This thesis will focus on the following problems.

1.1.1 Confusing and Ambiguous Terminology

Cloud providers use non-standardized naming terminologies for self-branding their uniqueness. For example, Compute services are called Elastic Compute Cloud (EC2) Unit by Amazon Web Service (AWS), Compute Engine by Google, Elastic Compute Service (ECS) by Alibaba, or just Virtual Machines (VMs), Instances, Servers or Units by others. The NIST definition describes Compute services as the provision of processing computing resources.

Ambiguous terminologies are often used to describe similar configurations. For instance, different units of measurements are used for measuring CPU clock speed by different Cloud computing providers. Amazon refers to it as "ECUs" [9]: "*One EC2 Compute Unit provides the equivalent COMPUTE capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. This is also the equivalent to an early-2006 1.7 GHz Xeon processor referenced in our original documentation*". In 2007, AMD and Intel released both dual-core and quad-core models of the Opteron and Xeon chips, respectively. So it is not clear what an Amazon EC2 Compute Unit is compared to. To eliminate this ambiguity, we obtained the compute service clock speed by trying out the actual instance under Linux OS and run "`more /proc/cpuinfo`" on it. We performed unit conversions during the instantiation of concepts to simplify the discovery process. For example, an Amazon EC2 Micro Instance has 613 MB of memory, which is converted to approximately 0.599 GB, so values are in the same units when queried later.

Furthermore, Cloud providers typically publish their service description, pricing policies and Service-Level-Agreement (SLA) rules on their websites in various layouts. The relevant information may be updated without prior notice to the users. The structure of web pages can be changed significantly, leading to confusion. Hence, it is not an easy task to obtain reliable service descriptions from Cloud providers' websites and documentation, which often is the only sources of information. This leads to the following challenges: How to automatically fetch service descriptions published by Cloud providers and present them to decision-makers in a human-readable way? One way to tackle this problem is to build a generic "superclass" as a standard model so that everyone can talk the same language. Since most of the information is published on the web, we leverage existing semantic web technology to build such a model - the web ontology language (OWL). This model is also called ontology, and in Section 1.3.1 we will discuss in more detail how can we develop a unified and generic Cloud ontology to describe the services of any Cloud provider.

1.1.2 Heterogeneous Service Offers

The multi-layered Cloud Services (e.g., SaaS, PaaS, and IaaS), along with their heterogeneous types (e.g. compute, storage, network, web server, databases) and features (e.g. virtualization technology, SLA model, billing model, location) make the task of service selection become a complex problem.

There are also various pricing models [214]. Apart from the most well-known pay-as-you-go and the free model, there are other models like spot instance, two-part tariff, declining block rate, and so on. Within the pay-as-you-go model, typically users are charged per unit usage per period. However, there are many variations of unit and period measure, such as per ram hour, Input/output Operations Per Second (IOPs) per hour, GB per month etc. Spot instance or spot market pricing refers to services that have very volatile prices that are proportional to the demand and inversely proportional to availability. Thus, when the demand decreases, more instances become available, and the spot price goes down. Similarly, when the de-

mand increases, fewer instances are available, and the spot price goes up. A two-part tariff (TPT) is a form of price discrimination wherein the price of a product or service is composed of two parts - a lump-sum fee as well as a per-unit charge. AWS reserved instances belong to this category. Declining block rate is a price structure where the per-unit price of services decreases as the consumption increases, which is often used in cost models for Cloud storage and network traffic. Figure 1.2 illustrates some different billing models with the comparison. Pay-as-you-go price is referred to as "Retail Pricing", which is often more expensive than the "Long-Term Contract" (i.e. two-part tariff), with "Spot Market Pricing" being the cheapest option. While a Long-Term Contract offers you a better price, the elasticity and the agility of your resource decrease as you can not merely deprovision since you already purchased the resource for the long term.

Figure 1.2: Different Billing Models [54]

The diversity of offerings in the Cloud landscape leads to practical research questions: how is the service of a Cloud provider compared to similar offers from other providers? How could we optimize the process of composite Cloud service selection and bundling? For example, how does a decision-maker compare the cost/performance features of infrastructure services offered by different providers such as AWS, Microsoft Azure, FelxiScale? Though branded calculators are available from individual Cloud providers to calculate service leasing costs. It is not easy for decision-makers to generalize their requirements to fit different service offers with various quotas and limitations while comparing costs.

1.1.3 Complex Selection Scenarios

It is a cumbersome task for decision-makers to manually read Cloud providers' documentation for finding out which services are suitable for building their Cloud-based

application architecture (e.g., a biologist intends to host his genomics experiment in the Cloud). This problem is further aggravated due to the rapid emergence of services in the Cloud landscape.

Burstorm's [29] survey in 2013 showed that there were over 426 various compute and storage service providers with deployments in over 11 072 locations. Even within a particular provider, there are different variations of services. For example, AWS has 674 different offerings differentiated by price, QoS features and locations. In addition to these, every quarter, they add about four new services, changing business models (prices and terms), and sometimes adding more locations.

Varied service configurations and application provisioning QoS constraints further complicate the problem. Application owners must simultaneously consider and optimize complex dependence and various sets of criteria (price, features, location, QoS) to select the best mix of service offerings from an abundance of possibilities. Often it is not enough to consider one single type of service. For example, a content distribution solution not only requires storage but also involves satisfying the computing capabilities. Therefore, the desired architecture should be able to both store and process data as fast as possible while minimizing the cost.

The process of matching offers to decision makers' requirements involves bundling of multiple related Cloud services, computing combined cost under different billing models and discount offers and considering all possible or only valuable alternatives. The selection criteria can further include constraints on: location, memory, storage capacity, CPU speed, operating system and so on.

Moreover, the network QoS (e.g. the data transfer speed and latency) varies across the Internet. This variation is dependent on the location of the data centre and the location of the input data stream. Hence the research question: how to optimally choose the services in terms of price, availability, processing speed and QoS (e.g., the network throughput and response delivery latency).

1.1.4 User Requirements Realization

Price calculator services assume that users know their requirements exactly. For example, it assumes that users know the expected usage of each service in the Cloud, or users have a very rigid budget or QoS expectation. However, in practice, this is not the case. Instead, decision-makers often only have a vague idea of what they want to achieve. Users' requirements may not be in terms of usage or price. They may only know how many things like: customer requests arrive at the peak hour, or server average workload, or some bench-marking metrics. In this case, how can we make it easier for users to translate this into appropriate requirements? In other situations, Cloud users may have an existing in-house cluster where some monitoring statistics exist. Then, how can we incorporate this information when estimating resource usage in the Cloud? There is a lack of support for Cloud migration planning, either from an in-house environment to the Cloud or between Cloud services. For example, in 2018, GitLab migrated from Microsoft Azure Cloud to Google Cloud [1, 210]. Migration planning relies solely on the experience of engineers.

1.1.5 Non-standardized User-Unfriendly Interfaces

Different systems have different user interfaces, which often require high IT expertise, as the consequence of accessibility to those solutions is limited to decision-makers with expert knowledge., which is inadequate given the proliferation of the Cloud. Hence a set of research questions are raised: How to develop interfaces that can transform low, system-level programming to easy-to-use drag and drop operations? How do we improve and simplify the process of Cloud service selection and comparison?

1.2 Research Objectives

The research objectives of this thesis is to tackle the following problems:

1. Cloud Ontology: How to automatically fetch service descriptions published by Cloud providers and present them to decision-makers in a human-readable way? Can we develop a unified and generic Cloud ontology to describe the services of any Cloud provider?

2. Service Comparison: How does a decision-maker compare the cost/performance features of infrastructure services offered by different providers?

3. Service Selection: How to optimally choose the services in terms of price, availability, processing speed and QoS?

4. User Requirements: How to translate user expectations that are not in terms of usage, price or QoS?

5. User Interface and Tools: How to develop interfaces that can transform low, system-level programming to easy-to-use drag and drop operations? How do we improve and simplify the process of Cloud service selection and comparison with a visual aid?

1.3 Contributions

In addressing the research objectives mentioned in Section 1.2, this thesis has made the following contributions.

1.3.1 Cloud Computing Ontology

To address the problems of ambiguous terminology and heterogeneous service offers discussed in Sections 1.1.1 and 1.1.2, we designed the Cloud Computing Ontology (CoCoOn). CoCoOn defines concepts, features, attributes and relations of Cloud infrastructure services.

In 2012, we proposed the first version of CoCoOn, later retrospectively named v0.1.0, which models Cloud resources, including IaaS, SaaS and PaaS; location and

region; and some other attributes. Later on, we revised and extended this model to become CoCoOn v1.0.1. The significant additions of CoCoOn v1.0.1 in terms of the ontology features are the Cloud service pricing and QoS modelling. When CoCoOn was first developed, there were a few existing domain ontologies to reuse (e.g. CoCoOn predated PROV-O [160], schema.org, QUDT [163], SSN [45, 82], wikidata [220] etc.). The new CoCoOn makes use of those now popular existing ontologies. Also, to improve the re-usability, we added more "rdfs:comments", metadata, documentation, and use cases. We released all the research outputs in a GitHub repository under MIT license, so anyone can reuse and contribute to the ontology, datasets, code and tools developed throughout this research.

CoCoOn facilitates the description of Cloud infrastructure services; moreover, through mappings from provider descriptions, facilitates the discovery of infrastructure services based on their properties and features. More details are in Chapter 3.

1.3.2 Multicriteria Decision Support

Some of the "Cloud discovery" research just focused on the SaaS and PaaS domain, which I consider mostly can be categorized into web service discovery research. Other research projects may have included IaaS, but have over-simplified the billing model or offering types. We have explained that the complexity of IaaS offers selection in Section 1.1.3, and existing methods are impractical for most of the scenarios. Overall, existing projects included in Section 2.2 mostly has no reference to the code base of the implemented systems, so it is impossible to reproduce results, and not many of them provide an evaluation of real-world data.

To this end, we developed a service selection method that adopts an analytic hierarchy process (AHP)-based decision-making technique. Hence, optimal selection can be made regardless of whether the requirements are conflicting or not. It also enables the handling of multiple quantitative (i.e., numeric) and qualitative (a descriptive and non-numeric, such as location, CPU architecture, i.e., a 32- or 64-bit operating system) criteria. Note that pairwise comparisons [153] are used to help users determine a relative preference among a pool of nonnumerical attributes. For each pair of criteria, the user is required to provide a subjective the opinion of their relative importance to them. Then, the overall composite weight for each criterion can be calculated. Criteria that are taken into consideration during comparison can be grouped into two categories: the benefit and the cost. "Benefit" groups the "good" criteria that are meant to be maximized. Similarly, "Cost" groups the "bad" criteria to be minimized. Based on this, we defined a cost–benefit-ratio-based evaluation function to calculate the ranking for Cloud service options.

1.3.3 Usage and Cost Estimation

This thesis further investigates the problem of planning Cloud deployments discussed in Section 1.1.4. We suggested a theory-based queuing approach for estimating IaaS usage. Queuing theory is a widely studied method in QoS modelling and optimization. From the infrastructure system administrator perspective, we explored

ways to apply the queuing theory model to estimate the best-fit resource allocation for achieving the desired SLA. Many studies [115, 31, 225] have applied queuing theory in their Cloud provisioning mechanism. Since we do not focus on real-time provisioning or load balancing, we only use a queuing theory based method to estimate the required number of VMs, when other parameters are constrained (i.e. tight budget, performance target on waiting time or throughput).

To calculate Cloud resources' renting costs, the user needs to suggest their planned usage. Our approach has taken into account different workload patterns during renting cost calculations. For instance, we defined many patterns for users to quickly choose from (i.e. flat/capped usage, regular periodic bursts, liner incremental).

Alternatively, users may already have some historical usage statistics, in situations like they want to move in-house systems to the Cloud or from one Cloud/server renting service to another. Queuing theory modelling was investigated for the usage estimation based on historical customer workload and benchmarking.

1.3.4 CloudRecommender

Finally, the thesis shows how an integrated system, CloudRecommender, can be built from our proposed approaches, to address the problem of the non-standardized user-unfriendly interface, discussed in Section 1.1.5. It allows users to compare and select a Cloud service based on criteria, such as the total cost, the maximum size limit for the storage, and the memory size for the compute instance. Alternatively, users can combine multiple selection criteria by telling the system their preferences over interested parameters. By providing both a web Graphical User Interface (GUI) and an Application Program Interface (API), we exploited the power of a visual programming language to further enable intuitive Cloud service selection.

QoS Profiler is part of the data collection components. It collects network QoS statistics from different endpoints on the Internet to the Cloud data centres. Based on CloudHarmony's network test service [40], we set up multiple agents at geographically dispersed locations to collect end to end network QoS to various Cloud. We profiled the network performance continuously and later used the average for comparisons.

1.4 Outline of the Thesis

This thesis provides an in-depth study of the work we have undertaken to address the problems as mentioned above in Section 1.2.

Chapter 1 briefly describes the research problems and presents an overview of the solutions and the structure of the thesis.

Chapter 2 provides the background information and reviews the related works.

Chapter 3 presents the Cloud Computing Ontology we developed for automatically identifying Cloud services.

Chapter 4 introduces our decision support algorithm for ranking Cloud services.

Chapter 5 investigates techniques for resource and performance estimation.

Chapter 6 presents the tools we developed, including scripts and websites for CoCoOn, QoS profiler and CloudRecommender. This chapter details the architecture of the tools above and experiments conducted.

At last, Chapter 7 summarizes our research findings and recaps the core contributions.

Chapter 2

Background and Related Work

With the proliferation of a range of Cloud services over the last decade, efficient and accurate service discovery and selection based on user-specific requirements has become a significant challenge for decision makers [66], for example, determining an optimal web service when making service selection, identifying suitable virtual machine servers for deploying web service instances, etc. Effective service recommendation techniques are becoming important to help users (including developers) in their decision-making processes for critical application developments and deployments.

In this chapter, we give an overview of state of the art in our research area. We first provide background on the semantic web technologies in Section 2.1. Those semantic techniques are what we employes for standardizing Cloud computing concepts. At the end of this section, we review ontological researches that are related to Cloud and QoS modelling. Then we review related work on web services in Section 2.2 and summarize their limitations under the Cloud context. Next, in Section 2.3, we discuss related work on Cloud IaaS service selection, which can be classified into two categories: multi-criteria decision-making techniques and optimization-based approaches. In Section 2.4 we provide background on queueing theory, which we use for Cloud performance modelling.

2.1 Semantic Techniques

According to the W3C, "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries". The term was coined by Tim Berners-Lee for a web of data that can be processed by machines (i.e. machine-readable). While its critics have questioned its feasibility, proponents argued that applications in industry, biology and human sciences have already proven the validity of the original concept [182].

The term "Semantic Web" is often used more specifically to refer to the formats and technologies that enable it. The collection, structuring and recovery of linked data are enabled by technologies that provide a formal description of concepts, terms, and relationships within a given knowledge domain. These technologies are specified as W3C standards, and some examples are:

1. Resource Description Framework (RDF), a general method for describing information.
2. SPARQL, an RDF query language.
3. Web Ontology Language (OWL), a family of knowledge representation languages. They are used for adding meaning to web content by annotating it with terms defined in ontologies. They are supported by tools (e.g. Protégé) and APIs (e.g. OWL API). OWL is developed based on description logic [91].

2.1.1 Ontology

Many industries (i.e. smart factories, business analytics, etc.) are becoming increasingly dependent on complex automatic software systems for tasks like resource allocation, business decision making, etc. For example, to make decentralized decisions, those systems need to cooperate as well as with humans. However, those systems typically access data from different models, which have been independently developed in different (often incompatible) formats using different types of proprietary software. Furthermore, these models may not come with well-defined semantics, and their specifications can be ambiguous. As a result, model development, maintenance, and integration, as well as data exchange and sharing, pose significant challenges in practice [108].

Nowadays, adoption of semantic technologies has been welcomed in many large companies such as in Google, Facebook, IBM [71] and Siemens [108]. For example, OWL2 ontologies are often used to capture conceptual information models. OWL2 is a vibrant and flexible modelling language. It not only comes with an unambiguous and standardized semantics but also with a wide range of tools that can be used to develop, validate, integrate, and reason with such models. Data can be stored in RDF triplestores and effectively queried in conjunction with the available ontologies. RDF is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, so it specifically supports the evolution of schemas over time without requiring all data consumers to be changed [166].

Like the software design patterns, many **Ontology Design patterns**(ODP) [44] also, have been proposed to suggest standardized solutions for common problems. Similarly, there are ontology engineering methodologies [205] (like software development methodologies), which people can follow during the design and development process of ontologies.

2.1.2 Linked Data

In computing, linked data (often capitalized as Linked Data) is a method of publishing structured data so that it can be interlinked and become more useful through semantic queries. It builds upon standard Web technologies such as HTTP, RDF and URIs, rather than using them to serve web pages for human readers, it extends them to share information in a way that can be read automatically by computers.

Tim Berners-Lee, director of the World Wide Web Consortium (W3C), coined the term in a 2006 design note about the Semantic Web project [119]. Linked data may also be open data, in which case it is usually described as linked open data (LOD). The standard serialization format for linked data is JSON-LD [103].

2.1.3 Schema.org

Schema.org [180] is an initiative launched on 2 June 2011 by Bing, Google and Yahoo to "create and support a common set of schemas for structured data markup on web pages." In November 2011 Yandex (whose search engine was the largest one in Russia at the time) joined the initiative.

Knowledge Graph is a knowledge base used by Google to enhance its search engine's results with information gathered from a variety of sources. Such information is presented to users in a box to the right of search results. Knowledge Graph boxes were added to Google's search engine in May 2012. The information covered by Knowledge Graph grew significantly after launch, tripling its original size within seven months, and being able to answer "roughly one-third" of the 100 billion monthly searches Google processed in May 2016 [111]. In October 2016, Google announced that Knowledge Graph held over 70 billion facts. The information is often used as a spoken answer in Google Assistant and Google Home Searches. The Knowledge Graph Search API uses standard schema.org types and is compliant with the JSON-LD specification.

However, Knowledge Graph and Google have been criticized for providing answers without source attribution. According to The Register [76], the implementation of direct answers in Google search results has caused significant readership declines for the online encyclopedia Wikipedia, from which the Knowledge Graph obtains some of its information. Dario Taraborelli, head of research at the Wikimedia Foundation, told The Washington Post that Google's omission of sources in its knowledge boxes "undermines people's ability to verify the information and, ultimately, to develop well-informed opinions".

2.1.4 RDF

The purpose of RDF is to provide a structure (aka framework) for describing identified things (aka resources) [168]. The RDF data model is based on statements to describe and feature resources, especially web resources, in the form of subject-predicate-object (resource–property–value) expressions, which is also called RDF triples [170].

RDF datasets can be expressed in a variety of syntax notations or data serialization formats, for example: RDF/XML, Resource Description Framework in Attributes (RDFa), JSON-LD, N-Triples [140], Terse RDF Triple Language (Turtle) [204], Notation3 (N3), etc. Figure 2.1 shows the serialization formats in RDF 1.0 and 1.1.

N-Triples is a format for storing and transmitting data. It is a line-based, plain text serialisation format for RDF graphs and a subset of the Turtle format. N-Triples should not be confused with Notation3, which is a superset of Turtle.

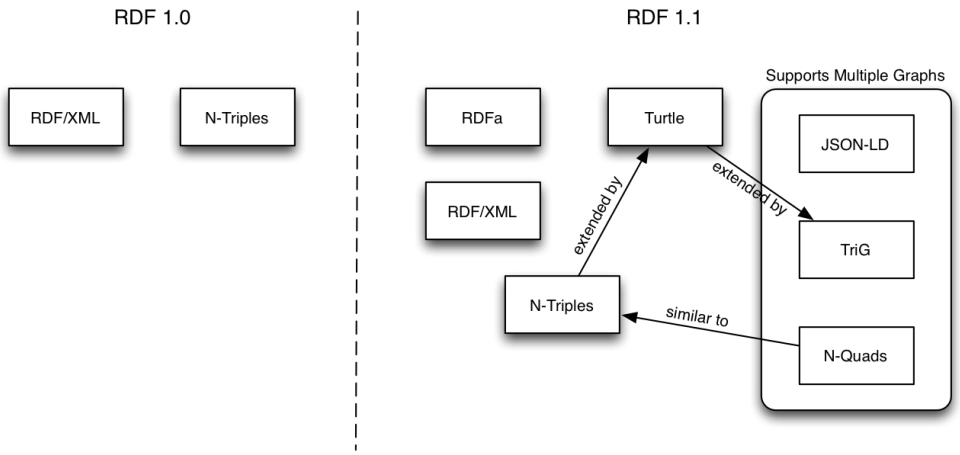


Figure 2.1: RDF 1.0 and 1.1 Serialization Formats [218]

Feature		Notation3	Turtle	N-Triples
Character encoding		UTF-8	UTF-8	ASCII
Directives	`@base`	✓	✓	✗
	`@forAll`	✓	✗	✗
	`@forSome`	✓	✗	✗
	`@keywords`	✓	✗	✗
	`@prefix`	✓	✓	✗
Lists	`()` (DAML lists)	✓	✓	✗
	`{ … }` (statement lists)	✓	✗	✗
Literals	`true` / `false` (Boolean)	✓	✓	✗
	`xsd:decimal` (decimal arbitrary length)	✓	✓	✗
	`xsd:double` (decimal double)	✓	✓	✗
	`xsd:integer` (decimal integer)	✓	✓	✗
Syntactic sugar	RDF paths	✓	✗	✗
	QNames	✓	✓	✗
	`a` / `@a` (equiv. to `rdf:type`)	✓	✓	✗
	`[]` (shorthand for blank node)	✓	✓	✗
	`=>` (*x* implies *y*)	✓	✗	✗
	`<=` (*y* implies *x*)	✓	✗	✗
	`=` (*x* is equivalent to *y*)	✓	✗	✗
		✓	✓	✗

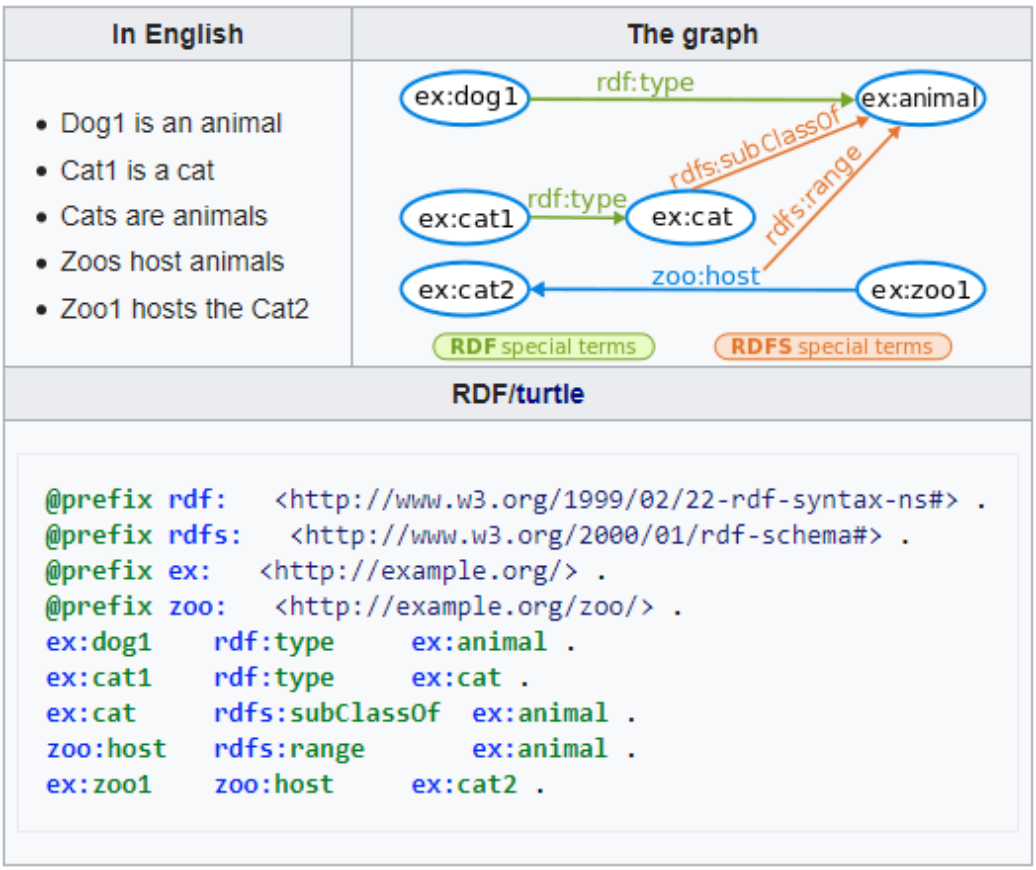


Figure 2.3: RDF Example vs. RDFS Example [169]

Turtle is a syntax and file format for expressing data in the RDF data model. It provides a syntax to describe RDF graphs in a compact textual form, which is easy to develop. Turtle is a subset of N3 and a superset of N-Triples. Turtle is popular among Semantic Web developers and considered as an easy-to-read alternative to RDF/XML. The typical file extension of Turtle files is `.ttl`. The character encoding of Turtle files should be `UTF-8`. The MIME type of Turtle is `text/turtle`. Turtle is supported by many software frameworks that can be used for querying and analyzing RDF data, such as Jena and Sesame. Turtle files consist of a sequence of directives, statements representing triples, and blank lines.

N3 is a shorthand non-XML serialization of RDF models, designed with human-readability in mind: N3 is much more compact and readable than XML RDF notation. N3 has several features that go beyond a serialization for RDF models, such as support for RDF-based rules. Turtle is a simplified, RDF-only subset of N3. Figure 2.2 shows a comparison of N3, Turtle, and N-Triples.

As explained above, RDF allows you to link resources/concepts together so you could say that "Cat1 is a cat". However, you cannot classify objects so you cannot say, for example, that cat is a subclass of the animal. See Figure 2.3 for an illustration.

On the other hand, **RDFS (RDF Schema)** gives you more expressive vocabulary.

This means you can start making statements about classes of things and types of relationships. It also allows you to describe in human readable text the meaning of a relationship or a class. It allows you to classify resources by using the class and subclass (i.e. `rdfs:class`, `rdfs:subclass`) notions. It also allows you to set restriction on the properties/relationships using `rdfs:Domain` and `rdfs:range`. It tells you legal uses of various classes and relationships. It is also used to indicate that a class or property is a sub-type of a more general type. For example "HumanParent" is a subclass of "Person". "Loves" is a sub-class of "Knows" [183].

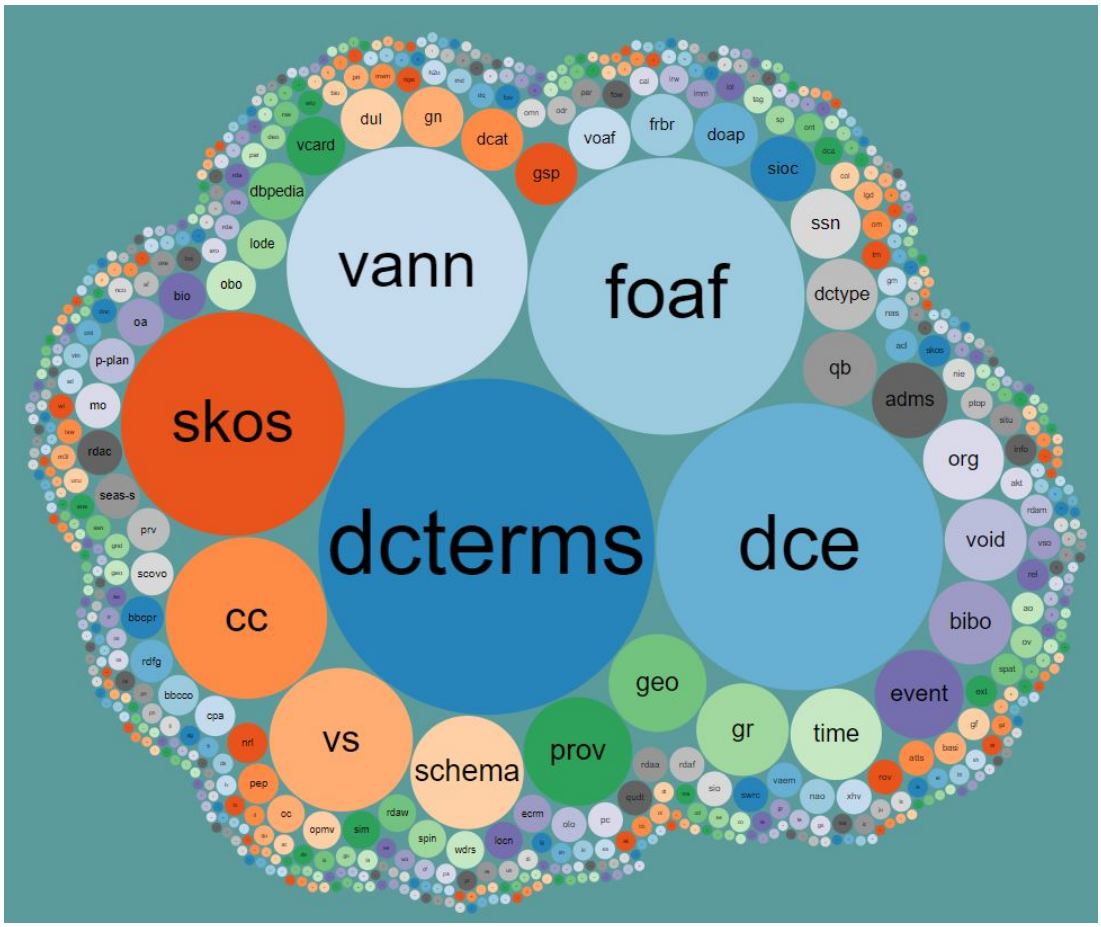


Figure 2.4: Existing Ontologies [120]

2.1.5 OWL

The purpose of OWL is to develop ontologies that are compatible with the World Wide Web [168]. OWL is closely related to RDF. However, OWL is not an extension to RDF, and in the same sense that DTDs and XML Schema are not extensions to XML [216]. OWL is a way of adding meaning / semantic richness to RDF. Among other things, this allows automated reasoning. OWL is a way to define types for RDF data, though OWL "typing" differs from conventional type systems in that it

has an open world assumption. OWL is represented using RDF triples and typically expressed using RDF/XML syntax. OWL allows you to add more restrictions. It categorises properties into object properties and data properties and allows you to add restrictions on them.

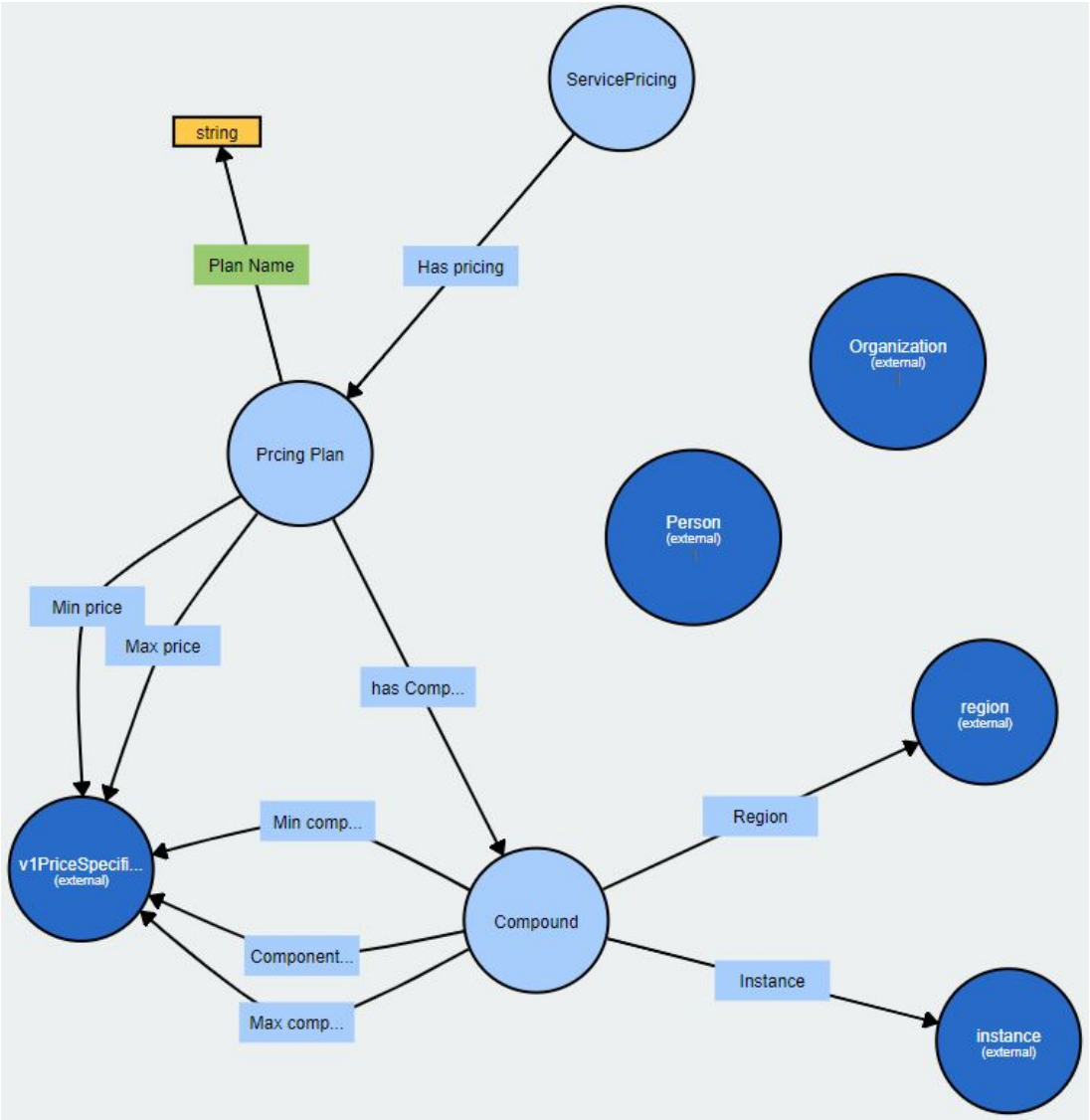


Figure 2.5: Ontology: ccpricing [51]

2.1.6 Ontologies Related to Cloud

Different from all the other models, our focus is on modelling concepts, features, attributes and relations of Cloud infrastructure services. We do not consider models for orchestration [136, 102] nor brokerage processes [27] in this work. Nonetheless, our ontology could be extended using the models proposed in those works. There are existing ontologies and models that focus on Web Services [174, 132] and their archi-

tectures [83] in general. Unlike these works, our model focuses on Cloud computing Infrastructure as a Service (IaaS), i.e. its features and price models. Furthermore, we have also developed tools for automatically adding semantics to information from providers' APIs.

We have used existing ontologies whenever fits, such as the Unit of Measure Ontology (QUDT) [163] for defining priced with currency values. Figure 2.4 shows some of the existing ontologies, the larger the size, the more it is referenced/used by other ontologies. In the following paragraphs, we review related works on Cloud ontology.

Previously, Parra-Royon and Benítez [51] have developed a few small Cloud ontologies, including **ccsla**, **ccpricing**, **ccinstances**, **ccregions**,**dmcc-schema** and **ccdm**. They modelled features including SLA, Price, VM instance feature and Region, Vendor and data mining experiments parameters. Their schemas are online. Each schema has one or two examples of modelling Services from Amazon. One exception is **ccdm** which has 6 examples of different ML experiments. The set of concepts and features they cover are limited and, as a result, their examples are limited to some simple cases. For instance, the examples presented in Section 3.3.3.5.1 cannot be modelled with their ontologies.

Figure 2.5 shows the structure of **ccpricing**, it can not handle the complexity of most common price options. For example, the cost of OS to be installed on the VM, network data transfer cost differed by destination and usage, snapshot storage costs, etc. In addition to these, **ccinstances** does not allow the unit to be specified in data. Furthermore, another reason why we did not use their "ccpricing", "ccinstances", and "ccregions" ontologies is because they used global scope constraints (i.e. rdfs:domain and rdfs:range) on most (if not all) of the classes and object properties, which we believe are too restrictive and can cause unintended inferences.

Boukadi et al. [27] have developed a Cloud Service Description Ontology (CSO), for modelling of Cloud service brokerage, which is shown in Figure 2.6. CSO includes features such as Cloud price, VM instance features, Region, Vendor, and so on. Their price model is rather simple and cannot model real-world scenarios. Their model and data are also not available online anymore to be evaluated further or to be reused in other contexts. Only figures for top-level topology are shown in their paper; a complete definition of the proposed ontology is not available. Furthermore, their service definition is too simple, and experiments can not model real situations. For example, in their experiment "Network" is assigned with a number between 0 and 100 without unit. How could this represent data transfer size and latency all together? It could be daily or yearly usage. Without knowing the destination, translating this to cost is impossible.

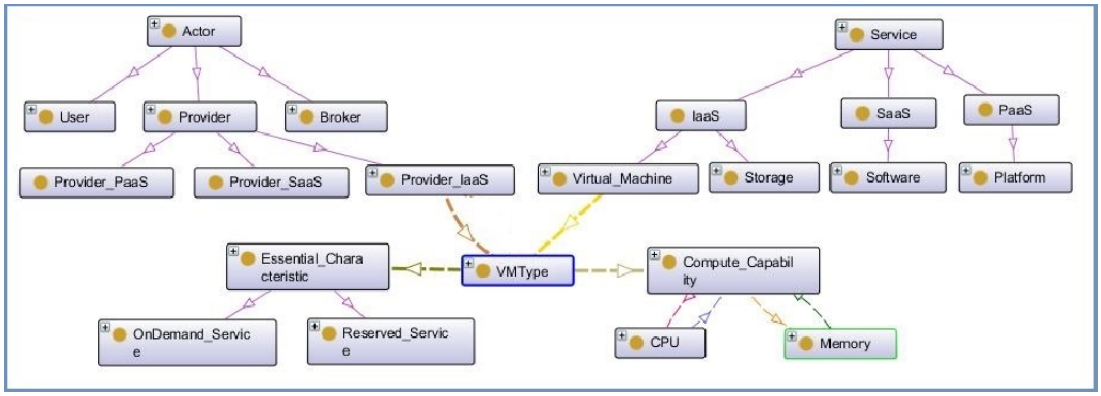
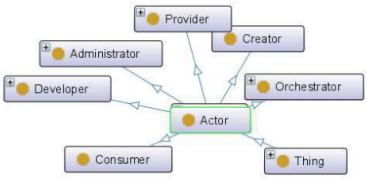
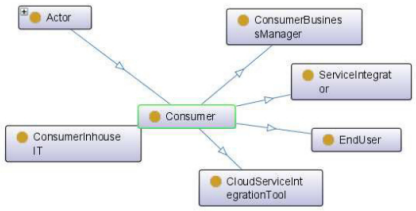


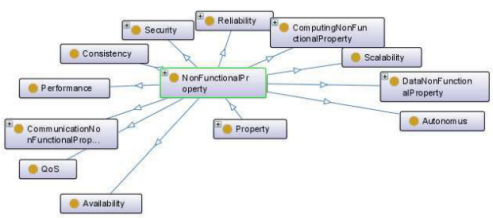
Figure 2.6: CSO [27]



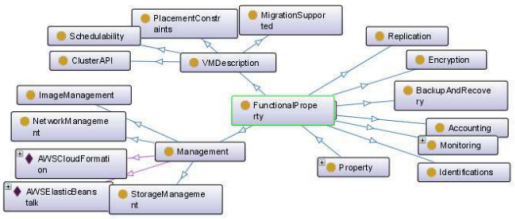
(a) Actor



(b) Consumer



(a) Non Functional Properties



(b) Functional Properties

Figure 2.7: mOSAIC ontology [136]

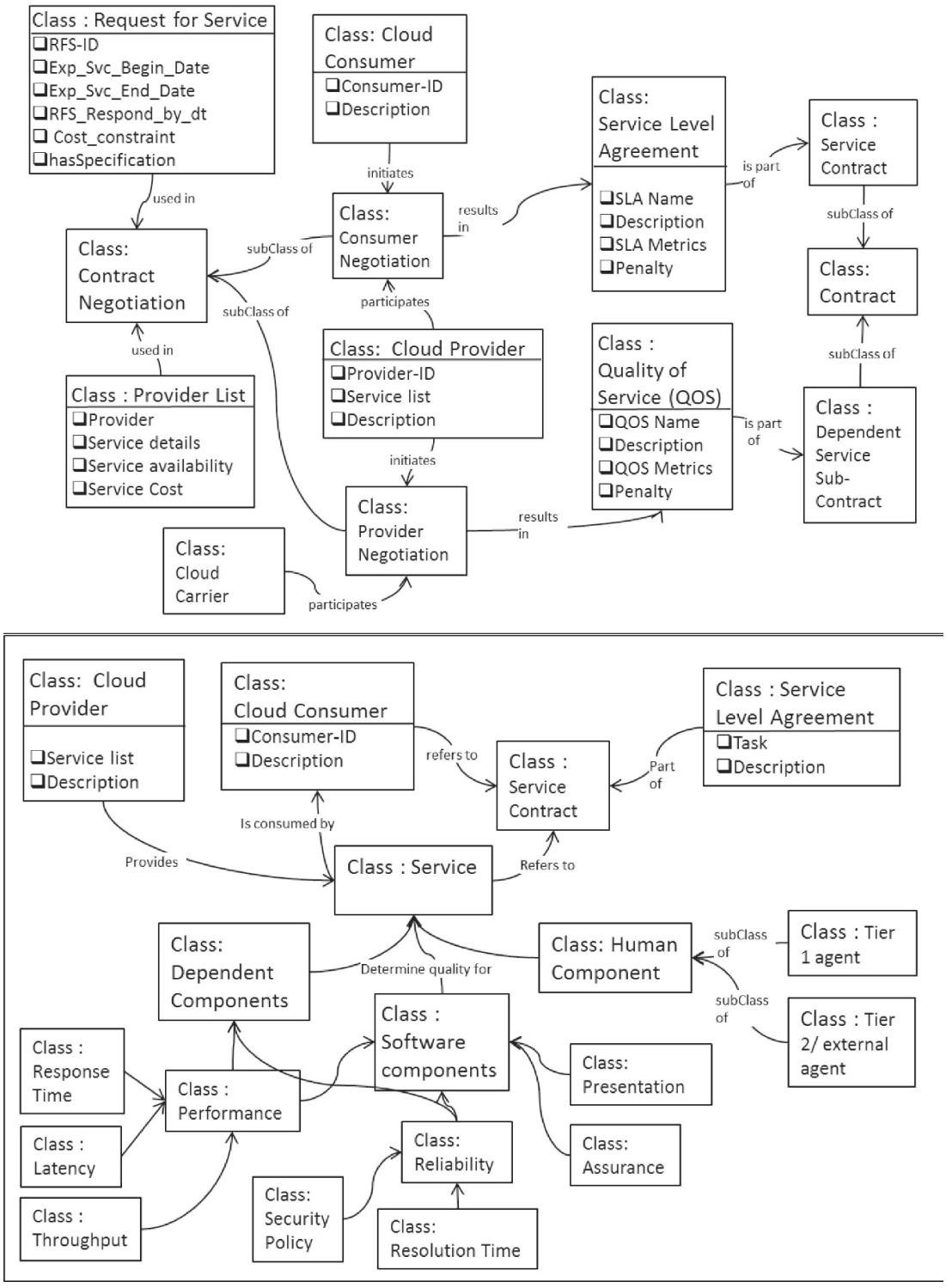


Figure 2.8: Ontology for the Lifecycle of IT Services in the Cloud [102]

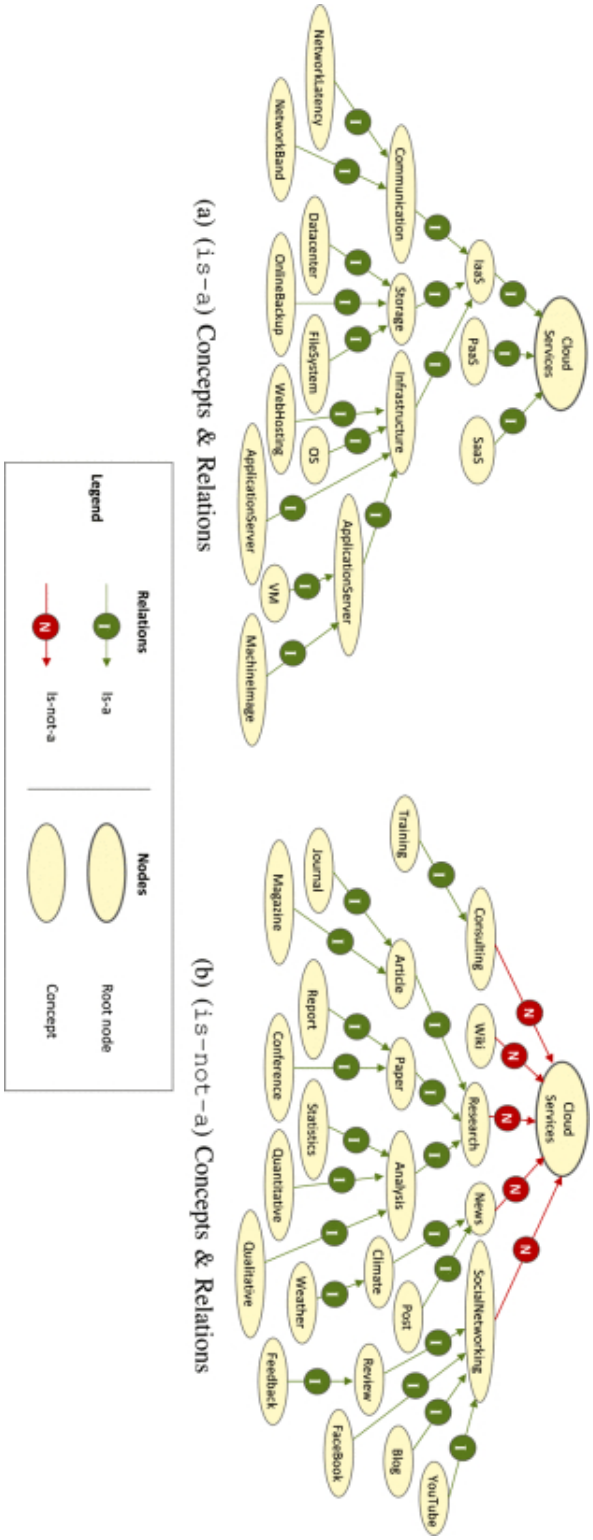


Figure 2.9: CSCE ontology [147]

Moscato et al. [136] developed an ontology for the mOSAIC project. This OWL ontology can be used for Cloud services negotiation (i.e. between customers and providers) and composition (i.e. by an administrator). The mOSAIC ontology has modelled actor (e.g. consumer and vendor), language (e.g. Java, Python), essential characteristics (e.g. resource pooling, maintenance, on-demand self-service, rapid elasticity), non-functional properties (e.g. scalability; autonomy; availability; QoS; performance; consistency; security; reliability), functional properties (e.g. replication, encryption, backup and recovery, accounting, monitoring, VM description, identification policies, management policies), layer (e.g. business layer consumer layer) and component (e.g. runtime component, environment).

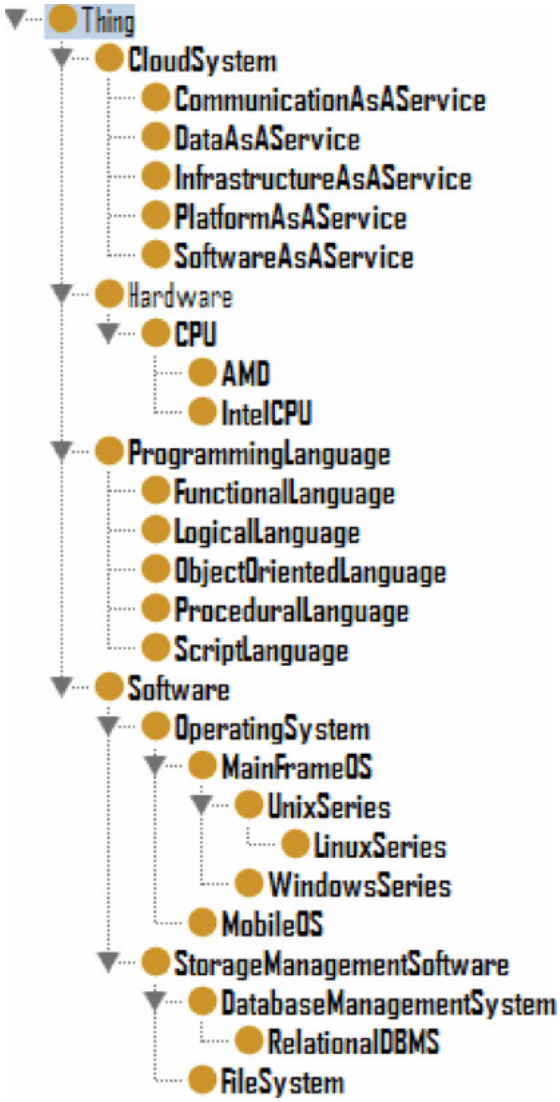


Figure 2.10: Cloudle ontology [104]

Figure 2.7 shows part of the mOSAIC ontology. The mOSAIC ontology has a

very different scope compared to ours, i.e. it does not cover IaaS resources in details nor the price of them. Also, only important concepts are modelled. For example, there is a QoS class, but QoS is a comprehensive concept. We have explained it in details in Chapter 3. One problem with this work is that the actual OWL file is not provided in their paper, and there is no usage example. There is one picture showing "InfrastructureSoftware" and "Computational" class, but it did not show their subclasses. Without the OWL file or example, we could not tell the difference between these two classes.

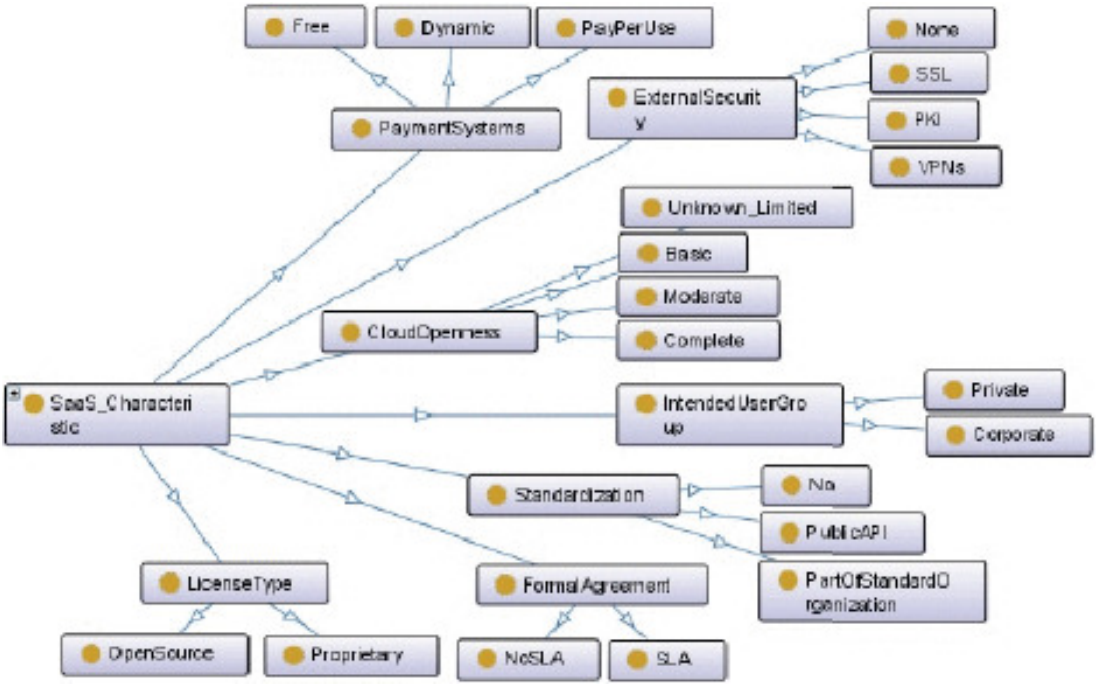


Figure 2.11: SaaS ontology [4]

Joshi, Yesha, and Finin [102] developed an OWL Ontology for the Lifecycle of IT Services in the Cloud, which is shown in Figure 2.8. These ontology models the steps involved in the phases of discovery, negotiation, composition, and consumption of Cloud services. The modelling of Cloud service features is minimal, and their link to an example of a storage service [109] is no longer accessible. This ontology modelled features such as Cloud provider, consumer, auditor, broker, SLA, latency, throughput, response time, request for service, contract, consumer negotiation security policy, and so on. The ontology models the steps involved in the phases of discovery, negotiation, composition, and consumption of Cloud services. However, their Cloud services are only defined at the very top level, like IaaS, PaaS and SaaS. Although there is one example of storage service (itso.owl), the schema file (storage_ontology.owl) is not accessible anymore.

Noor et al. [147] proposed a crawler engine (CSCE) for Cloud services discovery, and the ontology they developed for CSCE models IaaS, PaaS, SaaS, OS, VM, NetworkLatency, Storage, etc. This work mainly addressed the entry point problem for

fully automatic discovery. Since they automatically crawled terms to add to the ontology, a lot of irrelevant terms are included such as "Climate", "Weather", "Article", which are shown in Figure 2.9. Also, the actual ontology is not available, and only an image of CSCE ontology was shown in their paper. Besides, the data collected from each service is limited, and there is no mentioning of addressing the more complex problems of comparing or filtering services on certain attributes (price, VM types and RAM or performance).

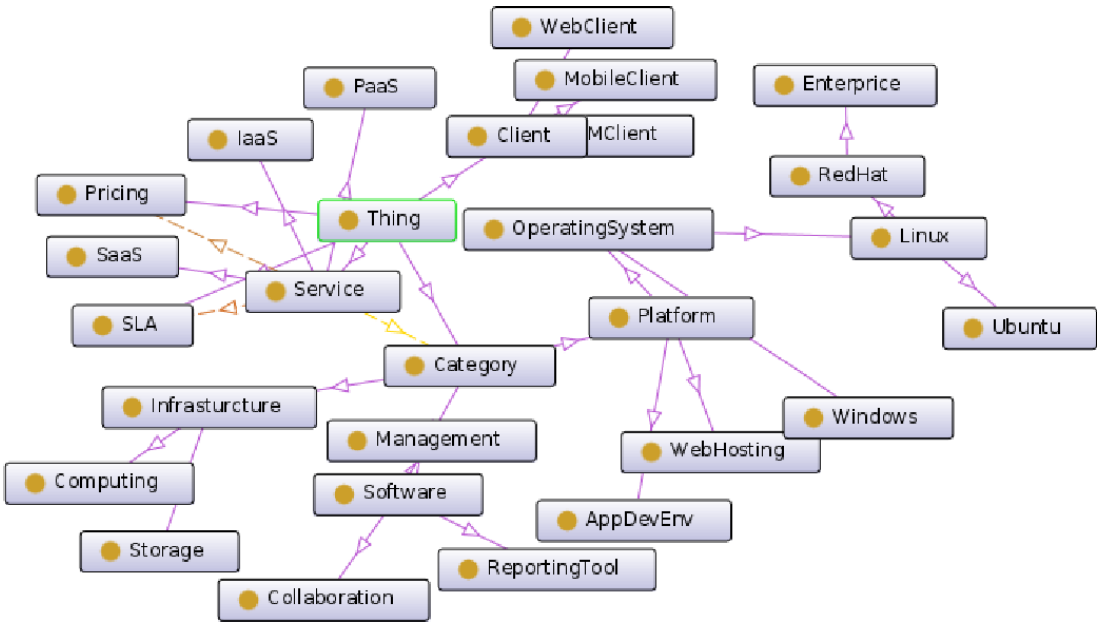


Figure 2.12: Auto Gathered Cloud Ontology [206]

Kang and Sim [104] and Kwang Mong Sim [114] developed Cloudle, which uses agents, ontology and k-means clustering algorithm to discover services over the Internet. Figure 2.10 shows the ontology they developed, which models IaaS, PaaS, SaaS, Communication as a Service, Data as a Service, CPU, Programming Language and Software. Cloudle ontology did not cover any details under the IaaS category. Their work only showed an image instead of providing the ontology file. Since only a minimal number of attributes are used in the similarity calculation, the comparison based on this metric is questionable. Furthermore, their experiment was evaluated using virtual (provider) websites, which did not consider many issues possibly occurring in practical scenarios. Cloudle also requires Cloud providers to register their services in a central database, which makes it inconvenient as well.

Afify et al. [4] proposed a semantic-based SaaS discovery and selection system. This system works as follows. Firstly, Cloud service providers register their services, the data provided by Cloud providers are processed by consulting the WordNet [223] ontology to expand the service description using the token synonyms before stored as an ontology. TFIDF algorithm is used to calculate a weight vector for different services, then the cosine similarity between those vectors are calculated.

Finally, the agglomerative clustering algorithm is applied to categorize services based on similarity. This system offers text/term based search. However, despite all the information retrieval techniques applied, it still requires service providers to register themselves manually. Figure 2.11 shows the ontology they developed, which focused on SaaS, and is different in scope to ours. Neither of the designed ontology or data is made available.

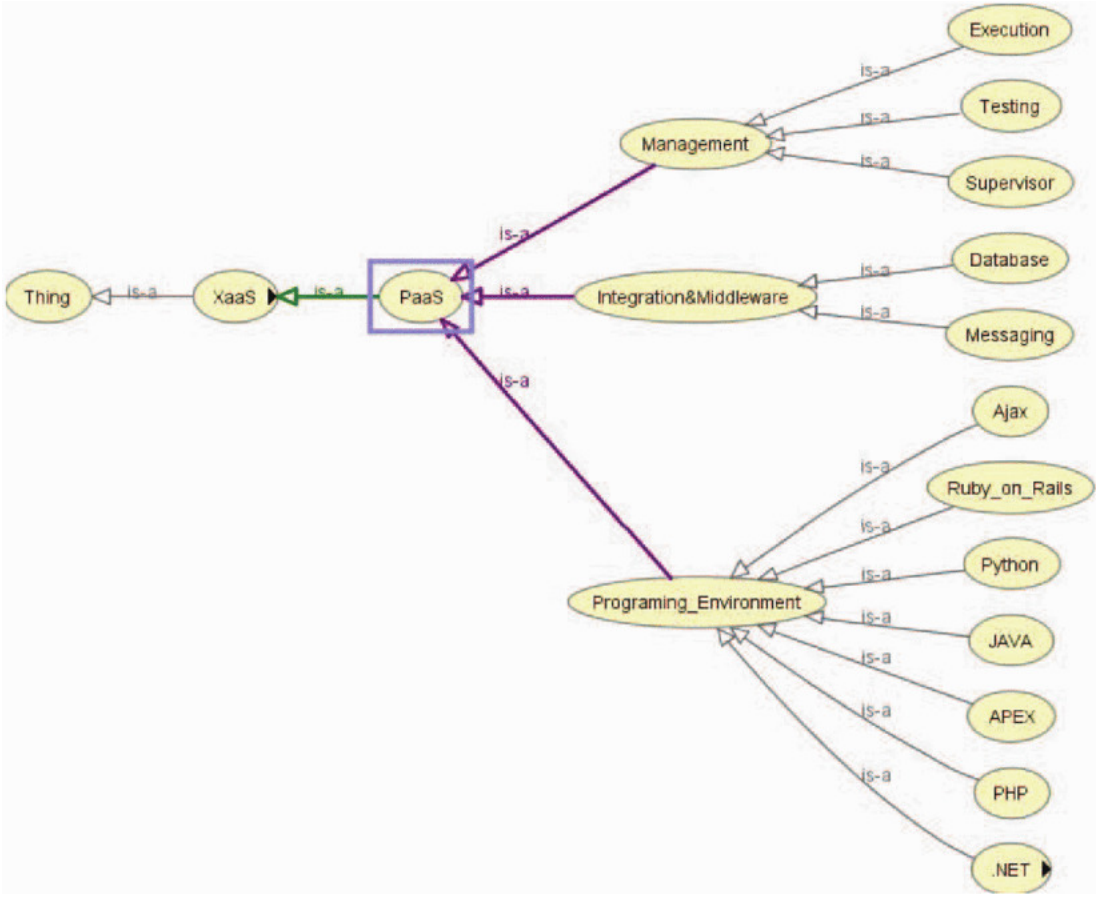


Figure 2.13: PaaS Ontology [33]

Vasudevan [206] proposed methods for semantic discovery of Cloud service published over RDF. Figure 2.12 shows their ontology, which models SaaS, PaaS, SaaS etc. They performed merging on ontologies based on similarities. Concept similarity is calculated by finding the Least Common Hypernym (LCH) in WordNet [223], and string similarities are calculated by Levenshtein (edit) distance. This system was implemented with the Jena inference rules engine [13]. However, ontologies are only partially presented in their paper, and no experimental data is made available, making it hard to evaluate and reproduce the work.

Chang et al. [33] proposed a method for services discovery on cloud environment by integrating intelligent agent and ontology. This work focused on PaaS scope, as shown in Figure 2.13, which has a very different character to IaaS, and the ontology

file is not available. They are searching for cloud services using inference rules executed by Jena engine [13]. Despite the different methods for retrieving information, the evaluation did not demonstrate much superiority compared to other researches. It achieved no more than 40 per cent recall, meaning that it can only retrieve 40 per cent of what should be discovered (with human review).

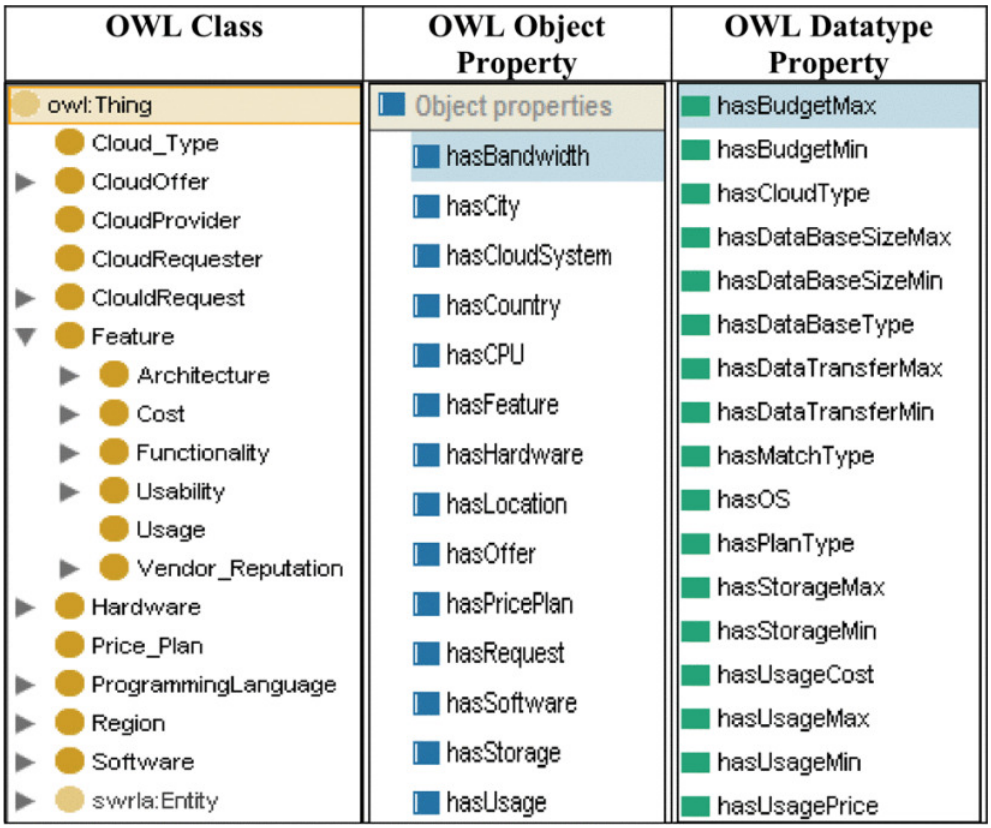


Figure 2.14: OWL-S based Cloud Service Ontology [144]

Ngan and Kanagasabai [144] proposed an OWL-S Based Semantic Cloud Service Broker, which uses SWRL [193] rule for matching criteria. The ontology they developed models provider, requester, cost, region and functionality, etc. The actual OWL file is not available. Instead, ontology is shown as Figure 2.14. In this figure, only top tow levels are shown, IaaS functionalities modeled are limited, and QoS parameters are not considered. Also, their online resource in the paper is no longer accessible.

Rodríguez-García et al. [173] proposed the ICT ontology, which models software. They annotate cloud service natural language descriptions using semantic techniques such as TFIDF and ontology (OWL2). The validation of the system was performed on a small set of services without statistical analysis on large data sets. Attributes covered are limited to software, but features specific to Cloud are not covered. Since the OWL file is not provided, we can only look at the figure showing part of the ontology, as shown in Figure 2.15. They have also used the namespace "soft", but prefix.cc shows that it is already claimed by some other ontology.

Figure 2.15: ICT ontology [173]

Mittal et al. [135] proposed methods for automatic extraction of metrics from SLAs of Cloud services. They developed an ontology modelling SLA, penalty, rebate, terms and conditions, availability, cost, provider, consumer, location, and so on. A prototype system was developed to extract the terminologies in SLA text documents automatically. The extracted terminologies are then saved as an RDF graph to represent the knowledge base. This work focused on SLA only. It did not provide a comparison of SLA among different providers. Since the OWL file is not available, we have to look at the provided Figure 2.16 to study ontology. It is hard to reuse this ontology, with no file, nor defined prefix.

Tahamtan et al. [195] proposed a Cloud repository and discovery framework, in which they introduced a unified business and Cloud service ontology. This ontology models business functions, deployment models, PaaS, Cloud management tools, SLA, IaaS, server location, data storage, etc. This work has broad coverage and has defined many classes, for example, the class "BusinessFunctions_and_Processes" includes 162 subclasses and the class "Cloud_Taxonomy" has 157 subclasses. However, the OWL file is not provided, and they did not define a prefix. They only partially

presented their classes in figures, for example, Figure 2.17 shows the "IaaS_concept". Despite the large vocabulary, there is no evidence of reusing existing ontology and object properties are not shown in figures.

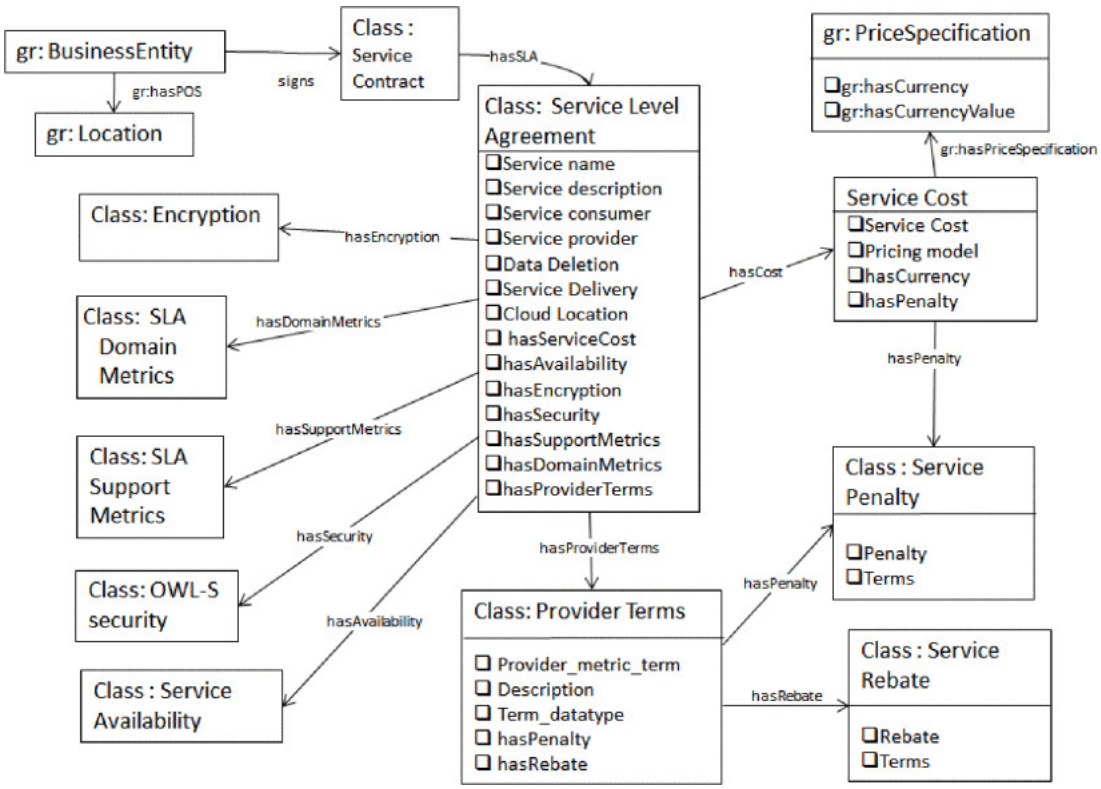


Figure 2.16: SLA ontology [135]

In the area of Quality of Service (QoS) modelling, some papers have proposed QoS ontologies (i.e. QoSOnt [55] and OWL-QoS [237]). However, they did not publish the actual specifications, and only figures/graphs were given. In this thesis, we provide formal modelling of QoS parameters and make it readily available for general use (see Section 3.3.3.6).

Zhang et al. [237] proposed an QoS measurements extension to "OWL-S", called "OWL-QoS". It models latency, accuracy, reliability, condition, parameter, delay and error tolerance (see Figure 2.18). However, the OWL file has not been made available for use, and there is no usage example either.

Dobson, Lock, and Sommerville [55] proposed an QoS focused ontology (QoSOnt) in 2005. QoSOnt models availability, time to complete, throughput, reliability, confidentiality, quantity and conversion rate (see Figure 2.19). Some of the topology graphs are blurry (refer to the classes from the attribute layer in Figure 3 of their paper). QoSOnt is now outdated, and the ontology file is not available.

Khanfir et al. [107] proposed a Web service selection framework based on the user's context and QoS. They developed an ontology for quality of service (see Figure 2.20). However, it is very simple and has no unit components nor a proper OWL

file definition.

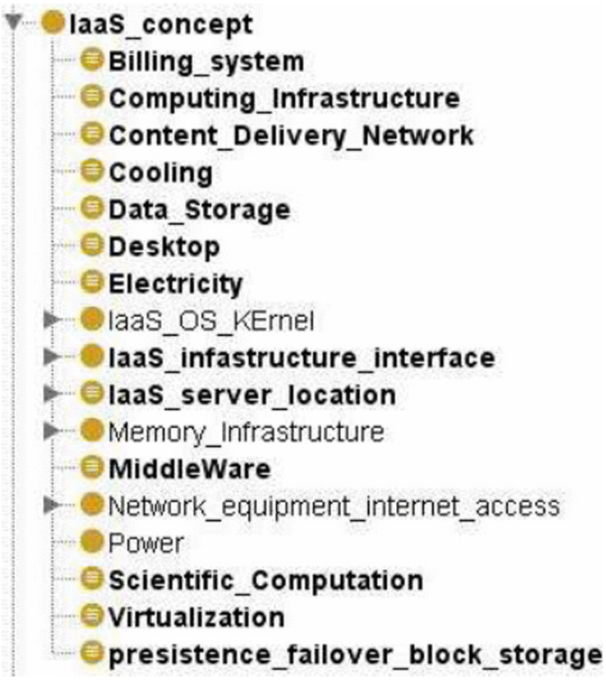


Figure 2.17: IaaS ontology [195]

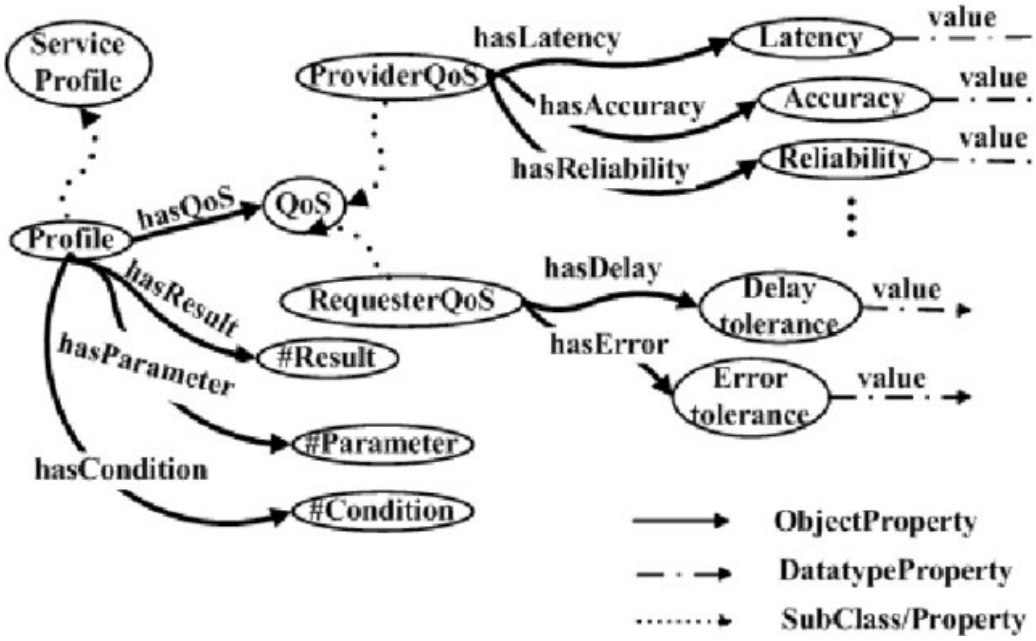


Figure 2.18: The structure of OWL-QoS [237]

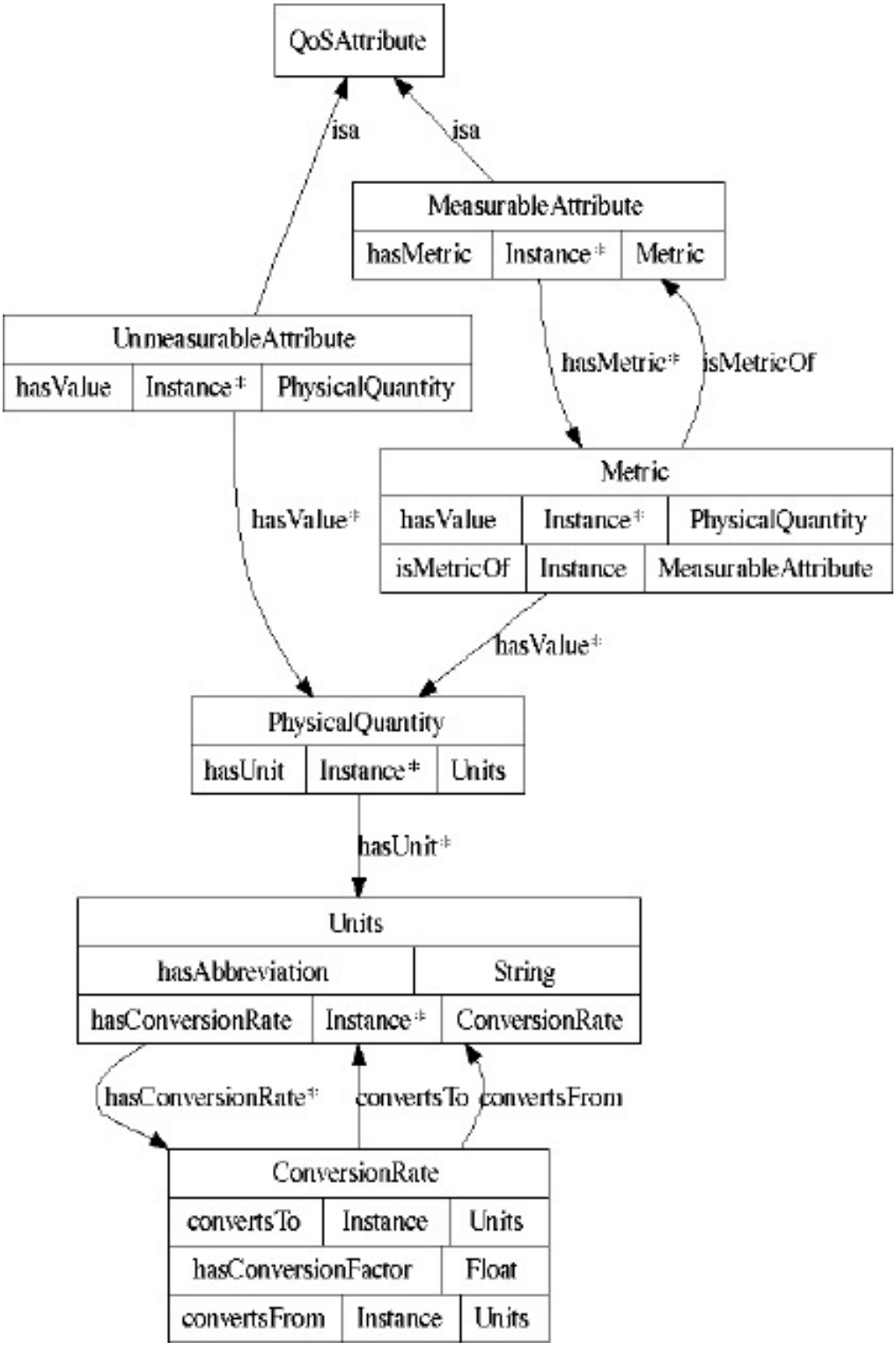


Figure 2.19: QoSOnt [55]

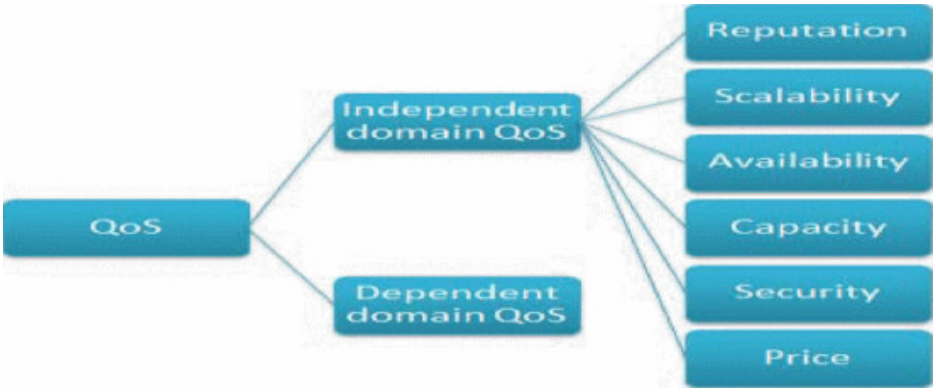


Figure 2.20: An Ontology for Quality of Service [107]

2.2 Web Service Discovery Techniques

In this section, we surveyed the state of the art in service selection and comparison techniques. We highlight their limitations, their relationship and dependency on some of the prior concepts from other fields in computing.

Traditional **Web service discovery techniques** includes Web Services Description Language (**WSDL**) and Universal Description, Discovery, and Integration (**UDDI**). UDDI has not been as widely adopted as its designers had hoped. On the other hand, **Semantic Web** technologies become popular, for which we will discuss in Section 2.1.

2.2.1 Fully Automatic Service Discovery Techniques

In general, all fully automatic techniques face the challenge of natural language processing, thus features captured automatically often are limited, and most research can only offer keyword-based search or does not consider service performance.

Hamza et al. [84] proposed a Cloud computing approach based on mobile agents for Web services discovery. They crawl information using mobile agents. Keyword-based search is used to compare between user requests and cloud service descriptions. However, it does not cover numeric attributes such as cost and performance.

Alkalbani et al. [8] designed a Hadoop-Based [12] crawler for SaaS Service Discovery. Their implementation used the search platform *Apache Solr* [14]. However, they did not distinguish between IaaS, PaaS or SaaS.

Nabeeh, El-Ghareeb, and Riad [141] proposed a Service Oriented Architecture [184] based Cloud services discovery framework. They used a multi-agent system for Cloud service discovery and ranking. They illustrated their design with architecture design and process flow diagrams, but they did not include design or method on how to implement a system following the proposed architecture. It is critical because the tasks mentioned in the paper can be very complex, without concrete demonstrations it is hard to know how they can be fully realised. The tasks include generating taxonomy tree, similarity reasoning, measuring compliance and evaluating client

source code.

Wheal and Yang [219] proposed a search engine system called CSRecommender, which was optimised, especially for cloud services. It identified some sites of interest to crawl their web pages, and a score was assigned to each page based on fundamental word frequencies using TFIDF (term frequency-inverse document frequency algorithm). It also collected users' ratings on search results to improve the recommendation relevance. It was applied to a few cloud services. Accuracy is low, and it does not cover all types of Cloud services.

2.2.2 Semi-automatic Service Discovery with More Complex Filtering

Parhi, Pattanayak, and Patra [154] proposed a multi-agent-based QoS-driven Web service discovery and composition framework, which is a prototype system for describing and discovering Cloud services using agents. Although their title says "Using Ontology", they did not implement it in their paper, only saying in the future work that they would include semantic-based searching techniques using web ontology language along with keyword-based searching techniques. Their design required service providers to implement agent systems to interact with the proposed service registry, which imposes high adoption cost. There is no evaluation with real data.

Goscinski and Brock [77] proposed a Resources Via Web Services (RVWS) framework for dynamic and attribute based publication, discovery and selection for cloud computing. This framework used WSDL documents to publish the states and characteristics of resources. It also includes a broker service layer to handle service discovery and selection. Twelve characteristic attributes and nine state attributes are summarised. Those attributes cover minimal features in Cloud, cost and performance were not considered in their work. For example, they were more interested in the state of the resource, as the amount of free disk space or memory, the number of processes running or the percentage of CPU used. The underlying technologies (WSDL, SOAP, UDDI) are a bit outdated compared to more popular recent technologies.

Chen, Bai, and Liu [34] proposed a service discovery method for Cloud Computing. They stored Cloud service in WSDL files for publishing in a UDDI directory. However, UDDI is an old (outdated) technology and abandoned by market and community.

2.3 Cloud Infrastructure Service Selection Techniques

The approaches to Cloud service selection can be categorised into: Multiple Criteria Decision Making (MCDM) methods, optimization methods and others [191]. In this thesis, we employed the Analytic Hierarchy Process (AHP) [177] method in the MCDM category.

Consider IaaS pricing calculation, although branded calculators are available from individual cloud providers [17, 158, 74, 39, 187] for calculating the service leasing cost, it is not easy for users to generalize their requirements to fit different Cloud service offers (with various quota and limitations). It is hard to conduct

a decent comparison between Cloud service offers from different providers, due to the very different natures of what things are part of billing and how they are part of billing. For some Cloud provider, a year consists of 365 days, whereas for other Cloud vendors a year consists of 30 days * 12 months = 360 days. While it is excellent that Cloud vendors provide an API for pricing estimation and calculation, the very different natures of the offerings themselves make it very hard to compare them [41].

2.3.1 Multiple Criteria Decision Analysis

Multiple criteria decision analysis (MCDA) or MCDM is a sub-discipline of operations research that explicitly evaluates multiple conflicting criteria in decision making (both in daily life and in settings such as business, government and medicine). Conflicting criteria typically occur in evaluating options: cost or price is usually one of the main criteria, and some measure of quality is typically another criterion, which may conflict with the cost [138].

MCDM problems have two categories: Multiple Objective Decision Making (MODM) and Multiple Attribute Decision Making (MADM). Methods for MODM have decision variables which are determined in a continuous or integer domain, with a large number of choices. Methods for MADM are generally discrete, with a limited number of pre-specified alternatives. Each decision has four main parts, namely: (a) alternatives, (b) criteria, (c) weights for criteria and (d) performance measures of alternatives for the criteria [67]. There are many methods for MADM and the commonly used ones are: Weighted Sum Model (WSM) , Weighted Product Method (WPM) , AHP, Techniques for Order Preference by Similarity to Ideal Solution (TOPSIS) and compromise ranking method (VIKOR) and **P**reference **R**anking **O**rganization **METH**od for **E**nrichment of **E**valuations (PROMETHEE) .

Triantaphyllou and Mann [203] demonstrated that it is impossible to determine precisely the best decision-making method to do so; one needs to use the best decision-making method. This problem of finding the best decision-making method always reaches a Decision Making Paradox. However, the results of this study recommend that for most of the cases the revised AHP appears to be the best decision-making method of the four examined, while the original AHP, appears to be the most inaccurate one.

The decision-making paradox arises from the quest for determining reliable decision-making methods. To find the best decision-making method, a decision problem needs to be formulated, for which different decision-making methods are the alternatives. Since in the beginning, it was assumed that the best method is not known, the problem of selecting the best method was solved by successively using different methods.

2.3.1.1 WSM

Table 2.1: Example alternatives

	C1	C2	C3	C4
A1	25	20	15	30
A2	10	30	20	30
A3	30	10	30	10
Weight	0.20	0.15	0.40	0.25

Han and Sim [94], Saripalli and Pingali [179], Zhao [238] applied WSM in their work. If there are M alternatives and N criteria, the WSM score can be calculated as follows:

$$A_i^{WSM} = \sum_{j=1}^{n} a_{ij} w_j \quad for \quad i = 1, 2, 3, ..., m \tag{2.1}$$

where n is the number of criteria, a_{ij} is the actual value of the ith alternative in terms of the jth criterion, and w_j is the weight of importance of the jth criterion.

Suppose we have a decision problem with three alternatives A1, A2, A3 where each described in terms of four criteria C1, C2, C3 and C4, as shown in Table 2.1. By Equation (2.1), the WSM scores for the three alternatives are:

$$A_1^{WSM} = 25 \times 0.20 + 20 \times 0.15 + 15 \times 0.40 + 30 \times 0.25 = 21.50 \tag{2.2}$$
$$A_2^{WSM} = 22.00 \tag{2.3}$$
$$A_3^{WSM} = 22.00 \tag{2.4}$$

Thus, these numerical results imply the following ranking of these three alternatives (in the maximization case): A2 = A3 > A1.

The advantage of this method is that it is a proportional linear transformation of the raw data. This means that the relative order of magnitude of the standardized scores remains equal [5].

2.3.1.2 WPM

This method is similar to WSM. However, instead of doing addition, multiplication is used to calculate aggregated values [212]. Suppose that a given MCDA problem is defined on malternatives and n decision criteria. Furthermore, let us assume that all the criteria are benefited criteria, that is, the higher the values are, the better it is. Next suppose that w_j denotes the relative weight of importance of the criterion C_j

and a_{ij} is the performance value of alternative A_i in terms of criterion C_j. Then, to compare two alternatives A_K and A_L, the following product is calculated:

$$P(\frac{A_i}{A_k}) = \prod_{j=1}^{n}(\frac{a_{ij}}{A_{kj}})^{w_j} \; for \; 1 \leq k \neq i \leq m. \tag{2.5}$$

If the ratio $P(\frac{A_i}{A_k})$ is greater than or equal to 1, then alternative A_i is more desirable than alternative A_k (in the maximization case). Then the best alternative is the one that is better than or at least equal to all other alternatives.

I have taken the same example in Table 2.1. WPM compares the alternatives by computing P values:

$$P(A1/A2) = (\frac{25}{10})^{0.20} * (\frac{20}{30})^{0.15} * (\frac{15}{20})^{0.40} * (\frac{30}{30})^{0.25} = 1.007 > 1 \tag{2.6}$$

Similarly, we also get:

$$P(A1/A3) = 1.067 > 1 \tag{2.7}$$

$$P(A2/A3) = 1.059 > 1 \tag{2.8}$$

Therefore, the best alternative is A1, and we have the ranking: A1 > A2 > A3.

It is worth noting that a useful way of choosing between an additive score function (i.e. WSM) and a multiplicative one (i.e. WPM) is to consider whether one is willing to trade off one criterion in exchange for some other criterion, even to the point where one has zero of the first criterion. If this is not acceptable to the decision maker, then the additive score the function is not appropriate. Likewise, if one does not believe the trade-off rate between alternatives should stay fixed, then again, an additive score function is not suitable [200]. For example, the UNDP (United Nations Development Programme) publishes an annual ranking of nations known as the Human Development Index, which is very influential and is used by the first world nations to guide their aid allocations. It is also used by pharmaceutical companies to decide which countries should receive discounted prices. This index is an aggregate of three criteria: life expectancy, education, and gross national income per capita. For many years the aggregation was carried out using additive weighting. This was criticised because this assumed that the criteria were perfectly substitutable. Consequently, the UNDP chose to change its methodology and the index is now calculated using a multiplicative scheme.

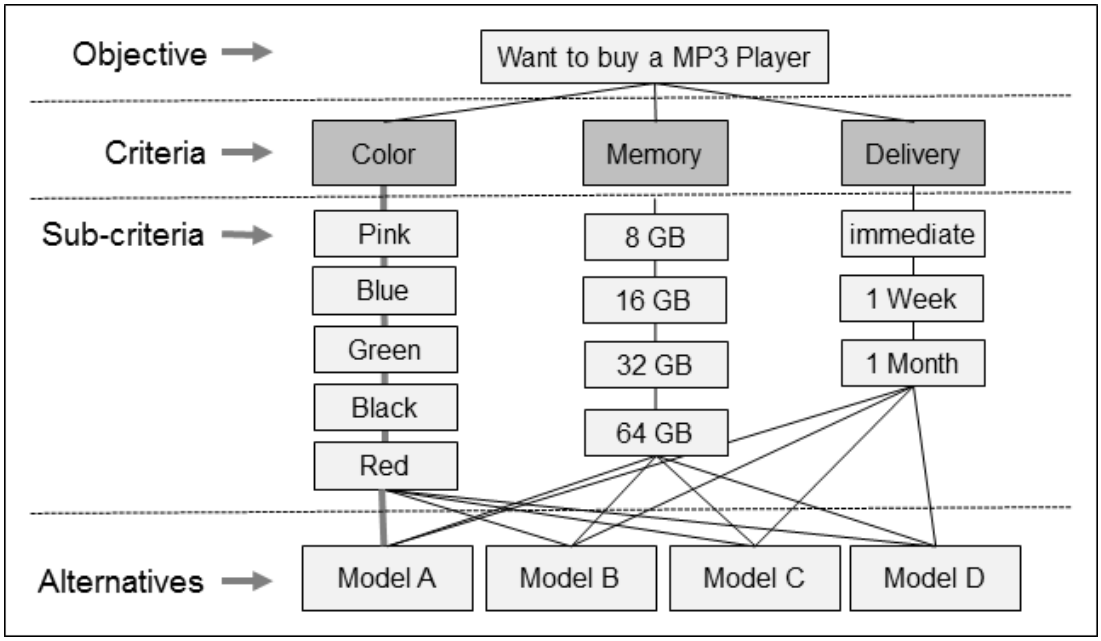


Figure 2.21: AHP Hierarchy for Solving a Decision Problem [215]

2.3.1.3 AHP

One of the most popular analytical techniques for complex decision-making problems is AHP [67]. There are a number of literatures [66, 72, 105, 145, 146] applied AHP or its variation Analytical Network Process (ANP) [134].

AHP [176, 178, 191] decomposes a decision-making problem into a system of hierarchies of objectives, criteria and alternatives. An AHP hierarchy can have as many levels as needed to characterize a particular decision situation fully. Many functional characteristics make AHP a useful methodology. These include the ability to handle decision situations involving subjective judgments, multiple decision makers and the ability to provide measures of consistency of preference. AHP is designed to reflect the way people think. It can efficiently deal with tangible as well as non-tangible criteria, mainly where the subjective judgments of different individuals constitute an essential part of the decision process. AHP computes ranking scores in a way similar to WSM. However, a key difference is that WSM assumes the weights for criteria are known, while this is not true in most cases. AHP applies pairwise comparison methods to determine a weight for each criterion in the comparative judgment phase. AHP derives ratio scales from pairwise comparisons of criteria and allows for some small inconsistencies in judgments. Inputs can be actual measurements, but also subjective opinions. As a result, ratio scales (weightings) and a consistency index will be calculated. For decision making with multiple inputs from different stakeholders, the geometric mean of individual inputs is used. Mathematically the method is based on the solution of an eigenvalue problem. The results of pairwise comparisons are arranged in a matrix. The first normalized eigenvector of the matrix gives the ratio scale (weighting). The largest eigenvalue determines the consistency

ratio.

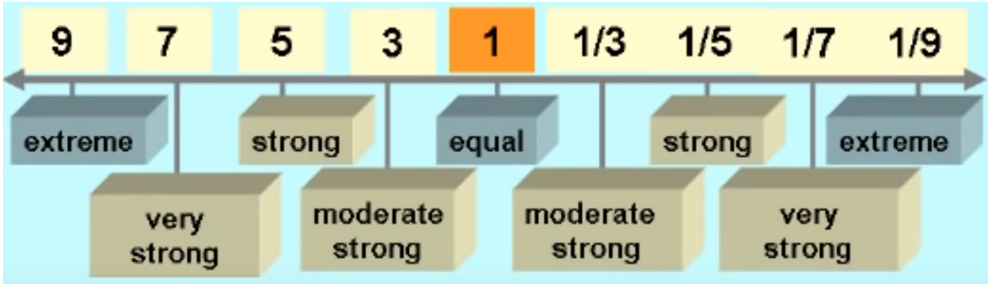


Figure 2.22: AHP Preference Scale for Pairwise Comparison [11]

In general, the steps of AHP [11] are:

1. Define objectives;
2. Structure elements in criteria, sub-criteria, alternatives, etc.;
3. Make a pairwise comparison of elements in each group;
4. Calculate local weights and consistency ratios;
5. Evaluate alternatives according to the final score.

Figure 2.21 shows a simple example of how to apply AHP to make a buying decision on MP3. The commonly used preference scale is shown in Figure 2.22. The matrix constructed from pairwise comparisons is called Pairwise Comparison Matrix (PCM) [112]. PCM of n criteria can be mathematically expressed as a square matrix $M = \{m_{ij}\}_{i,j=1}^{n}$ whose element m_{ij} represents the relative preference (or importance) of criterion C_i over criterion C_j.

The normalization method in the originally proposed AHP is criticized by Belton and Gear [23], who proposed **a revised version of the AHP model**. They demonstrated that rank reversal could occur because of relative values for each

Table 2.2: AHP Example

	C1	C2	C3	C4	
Weight	0.2000	0.1500	0.4000	0.2500	w_j
A1	25	20	15	30	a_{ij}
A2	10	30	20	30	
A3	30	10	30	10	
Sum	65	60	65	70	S_j
Original AHP Normalization: Value Divided by Sum					
A1	0.3846	0.3333	0.2308	0.4286	$\frac{a_{ij}}{S_j}$
A2	0.1538	0.5000	0.3077	0.4286	
A3	0.4615	0.1667	0.4615	0.1429	
Value multiplied by Weight ($w_j \frac{a_{ij}}{S_j}$)					Weighted Sum / Rank
A1	0.0769	0.0500	0.0923	0.1071	0.3264 / 3rd
A2	0.0308	0.0750	0.1231	0.1071	0.3360 / 2nd
A3	0.0923	0.0250	0.1846	0.0357	0.3376 / 1st
Revised AHP Normalization: Value Divided by Max					
Revised Weight	0.5000	0.3750	1.0000	0.6250	$w_j^R = \frac{w_j}{max(W)}$ *where* $W = \{w_1 ... w_4\}$
A1	0.8333	0.6667	0.5000	1.0000	$a_{ij}^R = \frac{a_{ij}}{max(a_{1j}, a_{2j}, a_{3j})}$
A2	0.3333	1.0000	0.6667	1.0000	
A3	1.0000	0.3333	1.0000	0.3333	
Value Multiplied by Weight ($w_j^R a_{ij}^R$)					Weighted Sum / Rank
A1	0.4167	0.2500	0.5000	0.6250	1.7917 / 2nd
A2	0.1667	0.3750	0.6667	0.6250	1.8333 / 1st
A3	0.5000	0.1250	1.0000	0.2083	1.8333 / 1st

criterion sum up to one. Rank reversal [18] means that the rankings of two alternatives are sometimes reversed when another alternative is added to or deleted from the set of alternatives. Instead of having the relative values of the alternatives which sum up to one, they propose to divide each relative value by the maximum value of the relative values. For cost criteria, we need to divide each value with the minimal value of the relative values. As shown in Table 2.2, values are first normalized using the original AHP method. That is to divide each value by the sum of all values in that column. In comparison, the revised AHP normalizes the values by dividing each value with the max value in the column. From the final weighted sum, we can see that these two methods result in different ranks.

AHP can be applied in many fields, e.g. Evaluation of product features, cost-benefit analysis, strategy development, selection of Key Performance Indicators (KPI), deriving weights for a combined performance index, and deriving a consolidated scale of importance from different inputs.

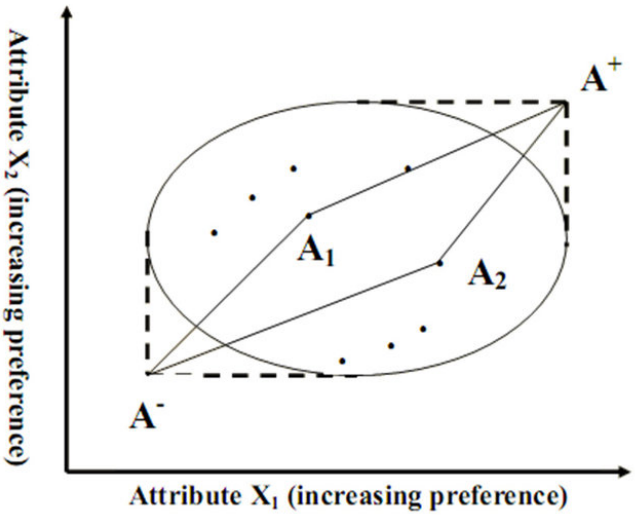


Figure 2.23: Basic concept of TOPSIS method [19]

2.3.1.4 TOPSIS

This method is based on the concepts that a chosen alternative should have the shortest Euclidean distance to the ideal solution and the farthest Euclidean distance from the negative ideal solution [67]. The ideal solution is a hypothetical solution for which all attribute values correspond to the maximum attribute values comprising the satisfying solutions. The negative ideal solution is a hypothetical solution for which all attribute values correspond to the minimum attribute values. TOPSIS thus gives a solution that is not only closest to the hypothetically best but also the farthest from the hypothetically worst. As illustrated in Figure 2.23, A^+ is the ideal positive point, and A^- is the ideal negative point. Compared with A_1, A_2 has a shorter distance to A^+ and it is further away from A^-, so it is a better alternative compared to A_1.

TOPSIS can suffer from ranking abnormality [143]. Ranking abnormality means that the ranking of candidate networks changes when the low ranking alternative is removed from the candidate list.

2.3.1.5 VIKOR

The VIKOR method determines a compromise solution which provides a maximum "group utility" for the majority and a minimum of an "individual regret" for the "opponent" [236]. Figure 2.24 provides a visual illustration, the noninferior set contains the Pareto optimal solutions, and the feasible set contains the feasible alternatives. The compromise solution F_c is a feasible solution that is the closest to the ideal F*, and compromise means an agreement established by mutual concessions.

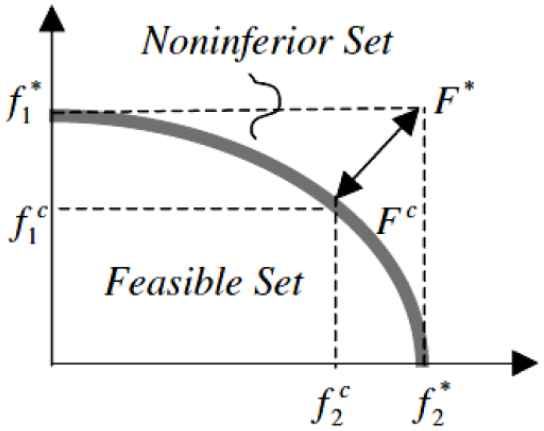


Figure 2.24: VIKOR: Ideal and Compromise Solution [241]

The VIKOR method was developed for multi-criteria optimization of complex systems. It determines the compromise ranking-list, the compromise solution, and the weight stability intervals for preference stability of the compromise solution obtained with initially given weights. This method focuses on ranking and selecting from a set of alternatives in the presence of conflicting criteria. It introduces the multicriteria ranking index based on the particular measure of closeness to the ideal solution [241].

2.3.1.6 PROMETHEE and Gaia Method

The prescriptive approach, named Promethee, provides the decision maker with both complete and partial rankings of the actions [159]. It is often used with the complement geometrical analysis approach, named Gaia [156]. Gaia is a descriptive approach that allows the decision maker to visualize the main features of a decision problem. The decision maker can quickly identify conflicts or synergies between criteria, to identify clusters of actions and to highlight remarkable performances. Promethee and Gaia together are often referred to as the PROMETHEE-GAIA method. While it can be used by individuals, it is most useful where groups of

people are working on complex problems, especially when involving a lot of human perceptions and judgments, multiple criteria and decisions with long-term impacts. It has unique advantages when essential elements of a decision are difficult to quantify or compare, or where collaboration among departments or team members are constrained by their different specializations or perspectives.

2.3.2 Optimization Based Methods

Optimization-based methods can be developed based on dynamic programming, greedy algorithm, integer programming, etc.

Chang, Liu, and Wu [32] proposed a Cloud storage provider selection algorithm, for maximising the probability of high availability, subject to a fixed budget. They transformed the selection problem to a 0–1 knapsack problem, then applied a dynamic programming-based approach. They looked at two situations: (i) minimum failure probability with a given budget, where the failure probability refers to the availability probability provided by vendors, i.e. 99.99%; (ii) maximum data survivability with a given budget, which is achieved by replicate data across different data centres.

Ye, Bouguettaya, and Zhou [227] proposed a QoS-ware Cloud service composition method. They adopted Bayesian networks to build a Cloud economic model. Influence diagram (ID) was employed to model the cloud service composition problem. Dynamic programming algorithm was used to solve such a problem.

Sundareswaran, Squicciarini, and Lin [192] applied a greedy algorithm to find the best combination of service providers that meets the user's service requirements. They proposed a method called CSP-index for indexing Cloud service providers, which used the B^+ tree to encode services and user requirements. They also modelled the relative importance of properties, as ordered by users. After indexes are generated for Cloud services, they do a k-means clustering on the indexes to keep services with similar properties close to each other. An example for the index would be "00010100", where hamming distance was used for comparing indexes during clustering. However, they only experimented on simulated data, as shown in Figure 2.25.

CSP Name	Variable Names								
	Service Type	Sec	QoS	Msrmt	Prcg unts	IS	OS	Prc	Reg
Amazon EC2	1, 2, 3	High	High	1, 2, 3, 4	1, 2, 3, 4	1, 2, 3, 4	1, 2	0.000 - 2.60	Yes
Windows Azure	1, 2, 3	High	High	1, 2, 3, 4	1, 4	1, 2, 3, 4	2	0.04- 0.96	No
Rackspace	1, 2, 3	Low	Medium	1, 3, 4	1, 4	2	1, 2	0.015 - 1.08	No
Salesforce	1, 2, 3	Low	Medium	4	1	N/D	N/D	2 - 260	No
Joynet	1, 2, 3	Low	Medium	4	1, 4	3, 4	1, 2	0.085 -2.80	No
Google Clouds	1, 2, 3	Medium	High	1, 2	1, 4		N/D	0.0057 - 0.0068	No

Figure 2.25: CSP-index Simulation Example [192]

Zheng et al. [240] proposed CloudRank, a QoS ranking prediction framework, Figure 2.26 shows its architecture. This framework uses past service usage experiences of other consumers to do prediction, and to save the cost and time of real time monitoring. A user is called an active user if he/she is requesting ranking prediction

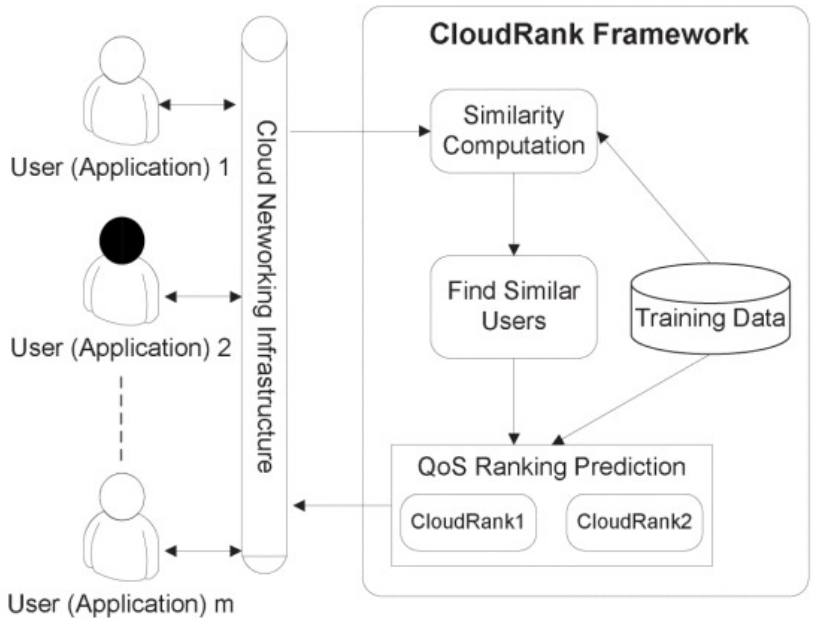


Figure 2.26: System Architecture of CloudRank Framework [240]

from the CloudRank framework. A user can obtain service ranking prediction of all available cloud services from the CloudRank by providing observed QoS values of some cloud services. They calculated the similarity between rankings from different users for the same service using Kendall Rank Correlation Coefficient (KRCC) , defined as:

$$\tau = \frac{(\text{number of concordant pairs}) - (\text{number of discordant pairs})}{\frac{n(n-1)}{2}} \tag{2.9}$$

The denominator is the total number of pair combinations, i.e. $\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$, so the coefficient must be in the range $-1 \leq \tau \leq 1$. Any pair of observations (x_i, y_i) and (x_j, y_j) where $i < j$, are said to be **concordant** if the sort order (rank) by x and by y agree: that is, if both $x_i > x_j$ and $y_i > y_j$ or if both $x_i < x_j$ and $y_i < y_j$. They are said to be **discordant**, if $x_i > x_j$ and $y_i < y_j$; or if $x_i < x_j$ and $y_i > y_j$. If the agreement between the two rankings is perfect (i.e., the two rankings are the same), the coefficient τ has value 1. If the disagreement between the two rankings is perfect (i.e., one ranking is the reverse of the other), the coefficient τ has value -1. If X and Y are independent, then we would expect the coefficient of τ to be approximately zero. Given two rankings on the same set of services, KRCC evaluates the degree of similarity by considering the number of inversions of service pairs which would be needed to transform one rank order into the other.

Within the CloudRank framework, the authors proposed two ranking algorithms: CloudRank1 and CloudRank2. CloudRank1 method calculates user preferences on a pair of services using the difference between the QoS parameter values observed by the end user. The basic idea is that the more often the similar users observe ser-

vice i as higher quality than service j, the stronger the evidence is for the current user to prefer i over j. Based on the preference of each service pair, all the services are sorted via a greedy method. In CloudRank2 method, confidence values are considered when computing the overall preference of a user for the service ranking. Explicit preference has the highest rank, and higher similarity means higher confidence and vice versa. The authors also compared the performance of their algorithms with several collaborative filtering approaches, where they employed the Normalized Discounted Cumulative Gain (NDCG) [100] metric for evaluating ranking results. Experiments show better prediction accuracy of the proposed methods compared with other approaches.

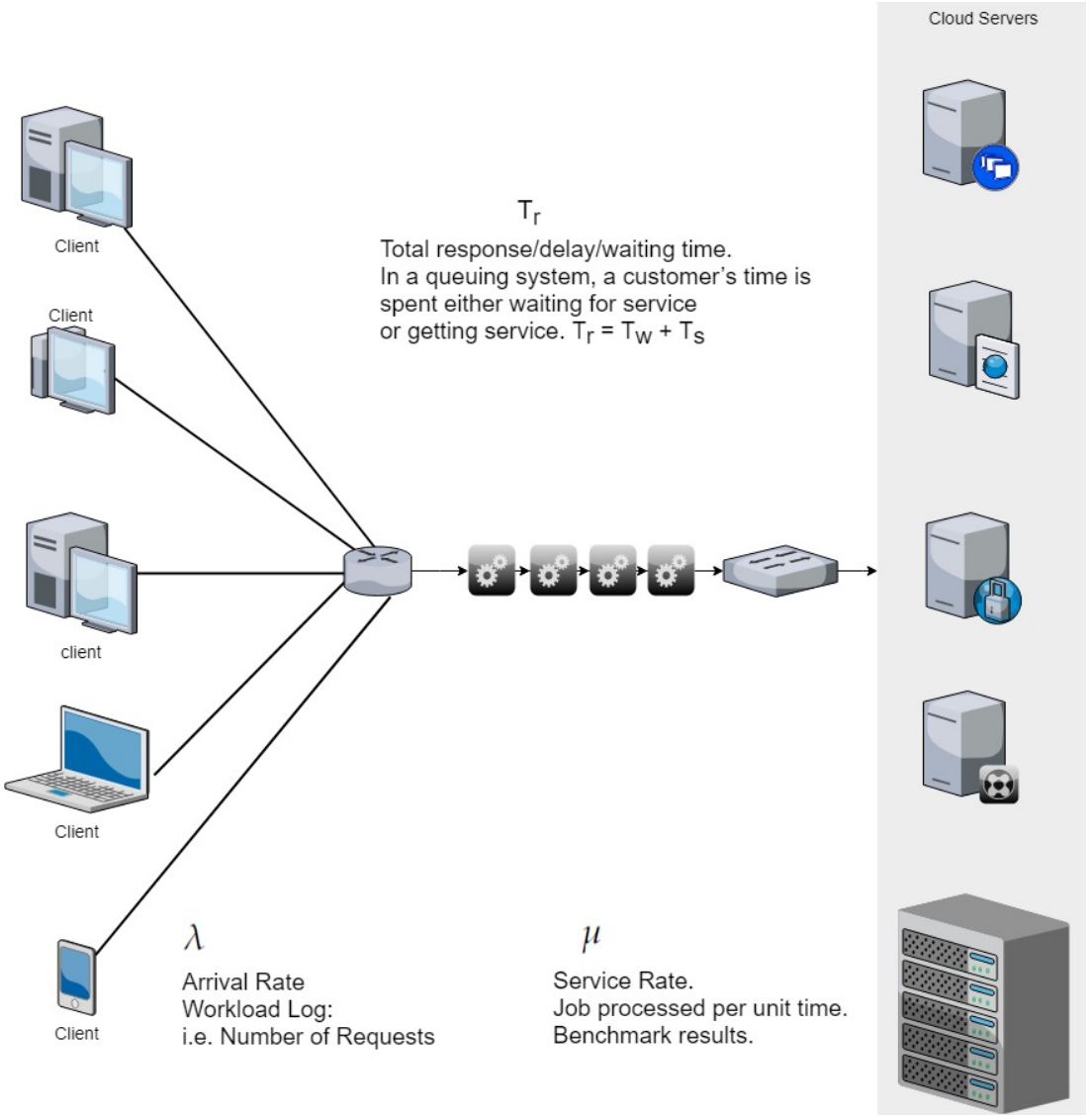


Figure 2.27: Queueing Theory Example

2.4 Queueing Theory

Queueing is quite common in life, for example, in telephone exchange, visa application queues, in computer systems, etc. Queueing theory problems were first raised by early telephone exchange centres at the beginning of the 20th century. Erlang [6] was the one who treated those network congestion problems. His works inspired engineers and mathematicians to deal with queueing problems using probabilistic methods. Queueing theory [78, 194] has become a field of applied probability, and many of its results have been used in operations research, computer science, telecommunication, traffic engineering and reliability theory.

There are many studies [115, 31, 225] which have applied the queuing theory in their Cloud provisioning mechanisms. Since we do not focus on real-time provisioning or load balancing, we only use a queuing theory based method to estimate the required number of VMs for achieving performance goals.

2.4.1 Components of a Queuing System

As illustrated in Figure 2.27, a queuing system consists of one or more servers that provide services of some sort for arriving customers. When all servers are busy, customers generally join one or more queues (lines) in front of the servers.

A queuing system is characterised by three components [164]: arrival process, service mechanism, queue discipline. Arrivals may originate from one or several sources being referred to as the calling population. The calling population can be limited or unlimited. An example of a limited calling population may be that of a fixed number of machines that fail randomly.

The **arrival process** describes how customers arrive at the system. If T_i^A is the inter-arrival time between the arrivals of the (i-1)-th and i-th customers. We shall denote the mean (or expected) inter-arrival time by $E(T^A)$. Moreover, the arrival frequency would be:

$$\lambda = \frac{1}{E(T^A)} \tag{2.10}$$

The **service mechanism** of a queuing system is specified by the number of servers and the probability distribution of customer's service time. Each server may have its queue or share a common queue. Let T_i^S be the service time of the i-th customer. We shall denote the mean service time of a customer by $E(T^S)$. Moreover, the service rate of a server would be:

$$\mu = \frac{1}{E(T^S)} \tag{2.11}$$

The **discipline** of a queuing system means the rule that a server uses to choose the next customer from the queue (if any) when the server completes the service of the current customer. Commonly used queue disciplines are:

- First In First Out (FIFO): Customers who come earlier leaves earlier.

- Last Come First Out (LIFO): Customers who come later leaves earlier.

- Priority: Customers are served in order of their importance based on their service requirements.
- Random Service (RS): Customers are selected randomly.

2.4.2 Kendall's Notation

In queuing theory, Kendall's notation denotes a system by:

$$X_a / X_s / m / K / n / D$$

where

X_a: distribution function of the inter-arrival times

X_s: distribution function of the service times

m: number of servers

K: the capacity of the system, i.e. the maximum number of customers in the system including the one being serviced

n: population size, i.e. number of sources of customers

D: service discipline

Exponentially distributed random variables are notated by **M**, meaning **Markovian** or memoryless. **General distribution** is noted by **G**, meaning any arbitrary distribution. Furthermore, if the population size and the capacity are infinite, the service discipline is FIFO, then they are omitted.

For example M/M/r/K/n stands for a system where the customers arrive from a finite-source with n elements, and they stay for an exponentially distributed time, the service times are exponentially distributed, the service is carried out according to the requests' arrival by r severs, and the system capacity is K.

2.4.3 Different Types of Queues

There are many different kinds of queueing models. In this thesis, we are more interested in the infinite-source models. We assume that requests for services are generated from an infinite population so that the arrival of a request to the system does not influence future arrivals.

Symbols used for describing the queueing models are list in table 2.3.

Table 2.3: Queuing Model Symbols and Formulas

n	Number of servers.
$duration$	Duration of test.
job_count	Number of job send during the test. There are different types of jobs; in web, it could be GET or POST HTTP requests; in analytical application, it would be a specific task, like word count (map reduce); in a SQL or NoSQL database, it would be a query (i.e. SELECT * FROM some_table).
λ	Arrival Rate (Rate of arrival of jobs), calculated by: $\lambda = \frac{job\_count}{duration}$
T_s	Service Time, which is the average time taken to service a job.
μ	Service Rate. Job processed per unit time. For example, to get the number of jobs processed per second $\mu = \frac{\textbf{1 second}}{T_s}$
ρ	Server utilization.
σ	Standard deviation. In this section, σ is specifically refer to the standard deviation of service time distribution.
T_r	Total response/delay/waiting time. In a queuing system, a customer's time is spent either waiting for service or getting service: $T_r = T_w + T_s$

2.4.3.1 M/M/1 Queue

We first look at the simplest non-trivial queue, which is the M/M/1 queue. Its requests arrival is according to a Poisson process with a rate of λ. It means the interarrival times are independent, exponentially distributed random variables with parameter λ. The service times are also assumed to be independent and exponentially distributed with parameter μ. Furthermore, all the involved random variables are supposed to be independent of each other. Average waiting time for the M/M/1 queue can be calculated as:

$$E[W^{M/M/1}] = \frac{\rho}{\mu(1-\rho)} \tag{2.12}$$

where ρ is the server utilization, sometimes also called traffic intensity or agent occupancy. The utilization of the system is the ratio between the rate of arrivals and

the rate of service:

$$\rho = \frac{\lambda}{n\mu} \tag{2.13}$$

where n is the number of servers. In Queuing Theory, ρ is used to describe how busy the system is, or how far away is the system is to its theoretical limit.

2.4.3.2 M/M/n Queue

The M/M/n queue is a variation of the classical queue assuming that the service is provided by n servers operating independently of each other. This modification is natural since if the mean the arrival rate is higher than the service rate. The system will not be stable. This is why the number of servers should be increased. Average delay/waiting time in M/M/n queue can be calculated as:

$$E[W^{M/M/n}] = \frac{C(n, \frac{\lambda}{\mu})}{n\mu - \lambda} + \frac{1}{\mu} \tag{2.14}$$

Erlang's C formula $C(n, \frac{\lambda}{\mu})$ describe the probability that an arriving customer is forced to join the queue (i.e. all servers are occupied):

$$C(n, \frac{\lambda}{\mu}) = \frac{1}{1 + (1-\rho)\frac{n!}{(n\rho)^n}\sum_{k=0}^{n-1}\frac{(n\rho)^n}{n!}} \tag{2.15}$$

where ρ is the server utilization, n is the number of servers.

2.4.3.3 M/G/1 Queue

So far, we have looked at systems with exponentially distributed serviced times. However, in many practical problems, these times are not exponentially distributed. It means that the investigation of queueing systems with generally distributed service times is natural, hence the M/G/1 queue . G means "general" service time distribution, which means any distribution can occur.

Little's theorem (or Little's law, Little's formula) [121] is a theorem by John Little which states that the long-term **average number of customers** ($\bar{L}$) in a stationary system is equal to the long-term **average effective arrival rate** ($\bar{\lambda}$) multiplied by the **average time** ($\bar{W}$) that a customer spends in the system. Algebraically the law is:

$$\bar{L} = \bar{\lambda}\bar{W} \tag{2.16}$$

It states a relation between the mean number of customers, mean arrival rate and the mean response time. A similar version can be stated for the mean queue length, mean arrival rate and mean waiting time. Although it looks intuitively easy, it is quite a remarkable result, as the relationship is "not influenced by the arrival process distribution, the service distribution, the service order, or practically anything else." The result can be applied to any system, and mainly, it can be applied to systems

within systems. Imagine an application that had no easy way to measure response time. If the mean number of jobs in the system and the throughput are known, the average response time can be found using Little's Law:

$$\text{mean response time} = \frac{\text{mean number in system}}{\text{mean throughput}} \tag{2.17}$$

The average waiting time in M/G/1 queue is calculated as:

$$E[W^{M/G/1}] = \frac{\rho^2 + \lambda^2 var[S]}{2\lambda(1-\rho)} \tag{2.18}$$

where var[S] is the variance of the service time. The variance of a data set is calculated by taking the arithmetic mean of the squared differences between each value and the mean value, which is also standard deviation squared, i.e. σ^2.

2.4.3.4 M/G/n Queue

Average delay/waiting time in M/G/n or M/G/k queue is

$$E[W^{M/G/n}] = \frac{CV^2 + 1}{2} E[W^{M/M/n}] \tag{2.19}$$

where CV is the coefficient of variation of the service time distribution:

$$CV = \frac{\sigma}{mean} = \frac{\sigma}{T_s} \tag{2.20}$$

2.4.3.5 Heavy Traffic Approximation

In a system with high occupancy rates (near 1) a heavy traffic approximation can be used to approximate the queueing length process by a reflected Brownian motion [85].

In situations when the system is considered busy or under heavy traffic, different formulas needed to be used for approximation. The ρ value will be between 0 and 1. If it is not less than 1, then the agents are overloaded, and the Erlang's C calculations are not meaningful and may give negative waiting times.

Heavy traffic approximation for wait time:

$$E[W_H^{M/G/1}] = \frac{\lambda(\frac{1}{\lambda^2} + var[S])}{2(1-\rho)} \tag{2.21}$$

The relative error of the heavy traffic approximation:

$$ero_H = \frac{1-\rho^2}{\rho^2 + \lambda^2 var[S]} \tag{2.22}$$

Chapter 3

Cloud Service Ontology

Typically, IaaS providers, including Amazon Web Services (AWS), Microsoft Azure, Google, Alibaba and others, give users the option to deploy their applications over a pool of virtually infinite services with practically no capital investment. Modest operating costs are charged proportionally to the actual use. Elasticity, cost benefits and abundance of resources motivate many organizations to migrate their enterprise applications (e.g. content management system, customer relationship management system and enterprise resource planning system) to the Cloud. Although Cloud offers the opportunity to focus on revenue growth and innovation, decision makers (e.g., CIOs, scientists, developers, engineers, etc.) are faced with the complexity of choosing among private, public, and hybrid Cloud options and selecting the right service delivery and deployment model.

To address the IaaS service discovery problem, we present an OWL-based ontology, namely the Cloud Computing Ontology (CoCoOn), that defines functional and non-functional concepts, attributes and relations of infrastructure services.

From a service discovery point of view, the selection process on the IaaS layer is based on a finite set of functional (e.g. CPU type, memory size and location) and non-functional (e.g. costs, QoS and security) configuration properties that can be satisfied by multiple providers. Similarly, there is a service discovery problem associated with the SaaS and PaaS offerings. However, we are not considering these issues in this thesis. In cloud computing, SaaS services are often developed and provided by third-party service providers who are different from the IaaS providers. We focus on IaaS that is the underpinning layer on which the PaaS services are hosted for creating SaaS applications. There are fewer studies in IaaS selection compared to SaaS selection, for which many service composition techniques are directly applicable.

Although popular search engines (e.g., Google, Bing, etc.) can point users to IaaS providers' web sites, blogs and wikis, they are not designed to compare and reason about the relations among different types of Cloud services and their configurations. Service description models and discovery mechanisms for determining the similarity among Cloud infrastructure services are needed to aid the user in the discovery and selection of the most cost-effective infrastructure services that meet the user's functional and non-functional requirements.

In this chapter, We identify and formalize the domain knowledge of multiple con-

figurations of infrastructure services. The core idea is to formally capture the domain knowledge of services using semantic Web languages like the Resource Description Framework (RDF) and the Web Ontology Language (OWL). The contributions are as follows: i) Identification of the essential concepts and relations of functional and non-functional configuration parameters of infrastructure services and their definitions in an ontology; ii) Modelling of service descriptions published by Cloud providers according to the developed ontology.

3.1 Common Approaches

In relation to the challenges in Cloud selection and comparison, described in Section 1.1, there are 3 common approaches for web services identification:

1. Manually maintains directories by categorizing manually-submitted or collected information. For example, the Yahoo subject directory applied this approach.

2. Use web crawling techniques to create listings automatically, i.e. Google Search index.

3. Combine both automatic crawling and manual maintenance, e.g. using manually-submitted URIs as seeds to generate indexes.

The first approach is the only feasible solution at the moment, as automatic crawling often cannot distinguish between useful and irrelevant links. But extensive research and standardization efforts [174, 132, 83, 48, 136, 152] have been put into developing information representation models, with Resource Description Framework (RDF) [166] and Web Ontology Language (OWL) [150].

3.2 CoCoOn v0.1.0

The initial version of our Cloud service ontology, named CoCoOn v0.1.0, defines the domain model of the IaaS layer. This ontology facilitates the description and discovery of Cloud infrastructure services. This ontology is defined in OWL [150]. Established domain classifications have been used to describe specific aspects of Cloud services, as a guiding reference [229]. For the layering of the ontology on top of Web service models, it draws inspirations and ideas from standard semantic Web service ontologies, i.e., OWL-S [131] and WSMO [224]. Consequently, modellers can use the grounding model and process model of OWL-S in combination with the presented Cloud service ontology to succinctly express common infrastructure Cloud services. We mapped the most prominent set of infrastructure services (i.e. Amazon, Azure, GoGrid, Rackspace, etc.) to CoCoOn. All common metadata fields in the ontology are referenced through standard Web ontologies (i.e. FOAF [62] and Dublin Core [52]), such as The organization, Author, First Name, etc.

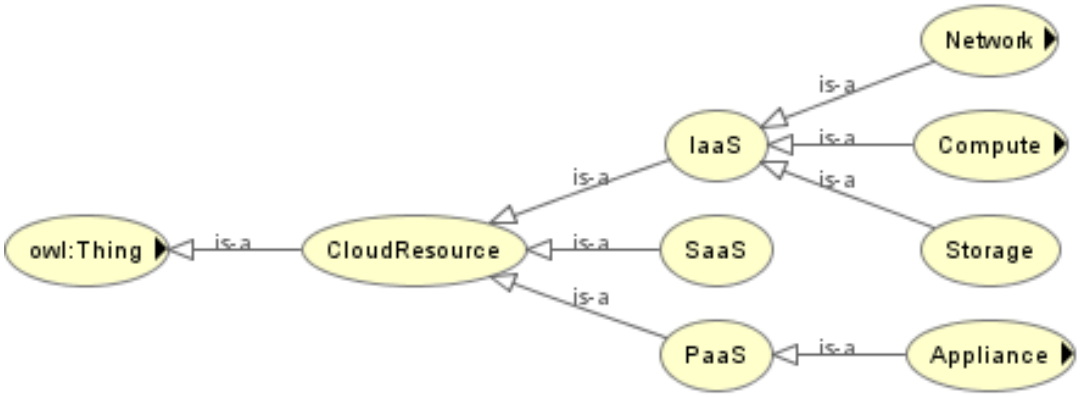


Figure 3.1: CoCoOn v0.1.0 Top Concepts in the IaaS layer

The first version of CoCoOn consists of two parts: functional Cloud service configuration parameters and non-functional service configuration parameters. In the following sections, we detail these two parts. We also present parts of this ontology in a visual form produced by the Cmap Ontology Editor tool [43, 42].

3.2.1 Functional Parameters

The main concept to describe functional Cloud service configurations in CoCoOn v0.1.0 is CloudResource, which can be of one of the three types of services: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) or Software-as-a-Service (SaaS).

Cloud services in the IaaS layer can be categorised into three classes: Compute, Network, and Storage, as shown in Figure 3.1. Compute class models the main concept in infrastructure services. Due to space limitation, a large graph consists of all classes cannot be shown here, but we have made it available online.[1]

The Compute class has the following object properties, hasVirtualization, hasCPU, hasMemoryAddressSize and hasNetworkStorage, which is shown in Figure 3.2. The hasCPU property links a Compute resource to one or many processors which can be of type CPU or ClusteredCPU. A Compute object can be linked to a Storage object by using the top-level object property hasStorage.

There are two different Storage types: i) LocalStorage, which is attached to a Compute resource with the hasLocalStorage property; ii) NetworkStorage, which is attached to a Compute resource with the hasNetworkStorage property. For example, S3 (Simple Storage Service) and EBS (Elastic Block Store) are two file storage services provided by Amazon. The main difference between them is with what they can be used with. EBS is meant explicitly for EC2 instances and is not accessible unless mounted to one. On the other hand, S3 is not limited to EC2. The files within an S3 bucket can be retrieved using HTTP protocols and even with BitTorrent. Many sites use S3 to hold most of their files because of its accessibility to HTTP clients [175].

[1] https://cmapscloud.ihmc.us/viewer/cmap/1SB4SYXQ2-1NW70FR-1MMSKJ

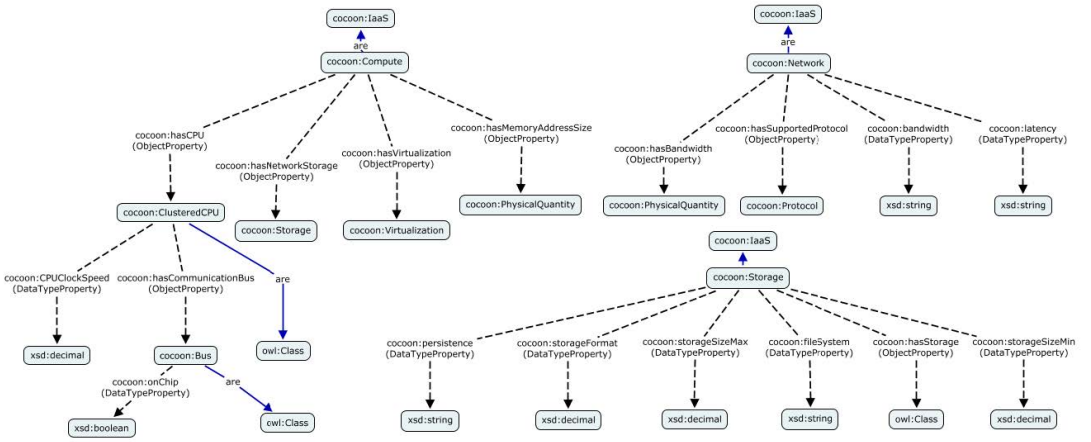


Figure 3.2: CoCoOn v0.1.0 SubClasses and properties for the Compute, Storage and Network classes

The hasNetworkStorage is an inverse (aka owl:inverseOf) property of the isAttached property, which can be used to define that a Storage resource is attached to a Compute resource. There is also an important distinction to be made between Storage resources that are attached to a Compute resource and Storage resources that can be attached. The latter is modeled with the *isAttachable* object property and its inverse property *hasAttachable*. These relations are important for the discovery of infrastructure services based on a user's requirement. For example, in the case of Amazon, we can model that a *BlockStorage* with a *StorageSizeMin* of 1GB and a *StorageSizeMax* of 1TB can be attached to any EC2 instance i.e., Standard, Micro, High-Memory, High-CPUCluster, ComputeCluster and GPUHigh-I/O. Consequently, if a user searches for a specific EC2 instance with, for example, 5GB persistent storage, the target Compute resource, and a relevant BlockStorage will be returned. That is because the isAttached relation in the user request can be matched with the definition of the Amazon EC2 instance with a BlockStorage defined to be isAttachable.

A Network resource can be described with the hasBandwidth and hasProtocol properties. Similar to how Storage resources are attached to Compute resources, we distinguish between the hasSupportedNetwork and hasNetwork property to either express that the specific network types can be used with a Compute resource or that they are in fact, used.

3.2.2 Non-Functional Parameters

For non-functional Cloud service configuration parameters, we distinguish between non-functional properties and QoS attributes. The first are properties of Cloud resources, which are known at design time. For example, PriceStorage, Provider, DeploymentModel. Whereas QoS attributes can only be recorded after at least one execution cycle of a Cloud service, for example, DiskReadOperations, Networkln, NetworkOut, etc. For QoS attributes, we distinguish MeasurableAttributes like the

ones above and UnmeasurableAttributes like Durability or Performance.

ConfigurationParameter and Metric are used in combination to define non-functional properties, e.g. Performance, Cost, etc. For example, ConfigurationParameter can be PriceStorage, PriceCompute, PriceDataTransferIn, PriceDataTransferOut, etc. A Metric can be ProbabilityOfFailureOnDemand or TransactionalThroughput. The resulting ontology is a directed graph where, for example, the property hasMetric (and its inverse property isMetricOf) is the basic link between ConfigurationParameters and *Metric* objects. For the QoS metrics, we used existing QoS ontologies [55]. For the ConfigurationParameter concepts, this ontology defines its independent taxonomy but refers to external ontologies for existing definitions of configuration parameters, such as QUDT [163]. Each configuration parameter has a *Name* and a *Metric* (qualitative or quantitative). The *Metric* itself has UnitOfMeasurement and Value. The type of configuration determines the nature of service using setting a minimum, maximum, or capacity limit or meeting a certain value. For example, the hasMemory configuration parameter of a Compute service can be set to have a value of 2 and a UnitOfMeasurement of GB.

3.3 CoCoOn v1.0.1

As the Web brings people into an age of information overload, presenting only the most relevant personalized and reliable information to customers is crucial to business success. Consumers of Cloud services also need better recommendations to make informed decisions. To fulfil the goal of a smart Cloud service recommendation, a unified model is needed as the foundation for data collection, reasoning and analytics. Due to the Cloud's relatively new emergence compared to traditional Computer Science fields, there is a lack of a well recognized standard ontology model. Semantic technologies are commonly employed to build such a model.

To this end, this section presents our work on cloud service ontology, which consolidates Cloud computing concepts. Built upon the previous work, we present the Cloud Computing Ontology (CoCoOn) v1.0.1: https://w3id.org/cocoon/v1.0.1. The relevant code, data and ontology are made available online as a Github project. Figure 3.3 depicts the IaaS related parts of CoCoOn v1.0.1. The major additions of CoCoOn v1.0.1 compared to its previous version [234, 232] are the Cloud service pricing and QoS modelling features. When CoCoOn was first developed, there were little existing domain ontologies to reuse, e.g. CoCoOn predated the development of PROV-O [160], schema.org, Unit of Measure Ontology (QUDT) [163], SSN [82] and Wikidata [220]. The new CoCoOn makes use of those popular existing ontologies. To improve the reusability, we added more rdfs:comments, metadata, documentation, and use cases. Because our old site on purl.org is hard to maintain and update, we moved the ontology and the documentation to GitHub. Also, we are using w3id.org as the permanent URL service instead, which should lead to better sustainability. More specifically, our model aims to facilitate the publication, discovery and comparison of IaaS, by: i) Providing a schema for constructing and executing complex queries; ii) Defining frequently referenced data as named individuals; iii) Providing

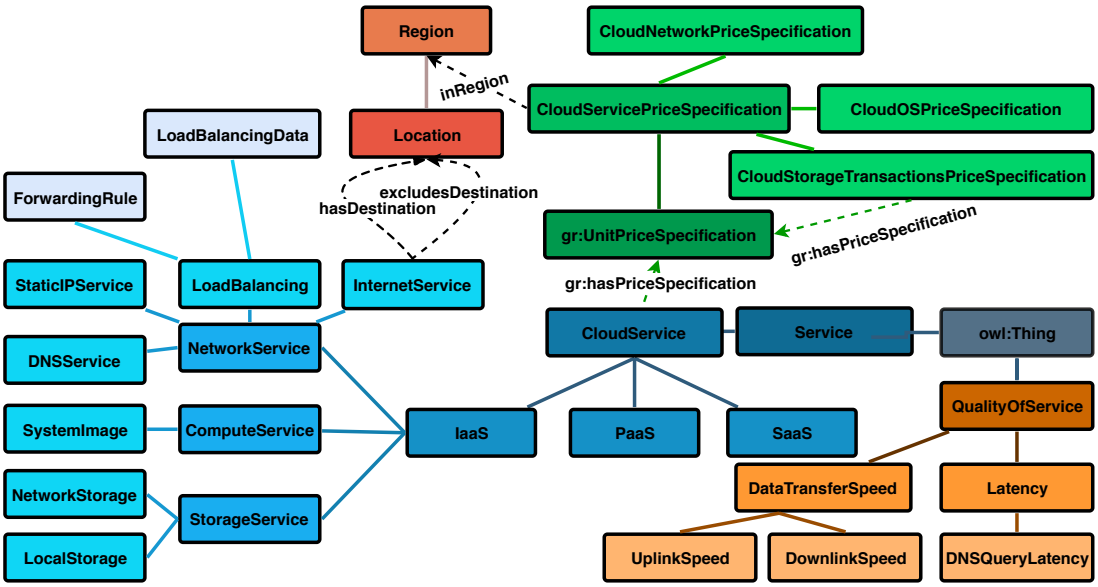


Figure 3.3: CoCoOn v1.0.1: IaaS related parts

a unified machine-readable specification, as opposed to provider-specific APIs and documentation. In addition to these, we demonstrate the capabilities of our model by providing real-life usage datasets. Those datasets include services from the Google Cloud and the Microsoft Azure Cloud, which is detailed in Section 6.1.1.

3.3.1 Major Changes

We have proposed a simple model describing concepts of Cloud infrastructure services (IaaS) in our previous work [235, 234], which is explained in Section 3.2. This work was mostly a taxonomy of IaaS. In this section, we propose an extension focusing on the core parameters for comparing and recommending IaaS services. We revised our CoCoOn ontology, i.e. changes have been made to namespaces, classes, properties, relationships and axioms.

We also introduced versioning from this release, for both ontology and data collected (expressed with the ontology), following the semantic versioning specification. Although this ontology is not in the sense of "in production", this version is a stable release that we intend to keep long term support and availability. We label the collection date of data with schema:dateModified. Thus historical price data can be recorded.

The new model is outlined in Section 3.3.3, with an explanation of its syntax and semantics, design, and formalization.

It has been a long time since we proposed the initial model of CoCoOn, which we retrospectively versioned v0.1.0. The major model changes in CoCoOn v1.0.1 are summarized below:

1. Additional external vocabularies.

2. Additional annotations, including version number, modification dates, metadata like author, web page, etc.
3. Removed classes and properties which are not important, and covered by some other ontologies, e.g. Input, Output, Interface, and User.
4. New class and properties focusing on IaaS and price specifications, i.e. cocoon:CloudOSPriceSpecification, cocoon:LoadBalancing, cocoon:TrafficDirection, etc.

In addition to the changes as mentioned above, we have also developed a full mapping service between CoCoOn v1.0.1 and Cloud vendor APIs for the Google Cloud and the Microsoft Azure Cloud. These mapping services demonstrate the usability and strength of the ontology we have developed. More details can be found in Section 6.1.

We have used existing ontologies whenever fits, such as QUDT for defining price with currency values. For the full list of ontologies we have referenced, see the online documentation.[2]. In this section, we have significantly extended the capabilities of our initial model, i.e. changes have been made to classes, properties, relationships and axioms, with a strong focus on flexibility and extensibility.

3.3.2 Reusing Existing Vocabularies

A set of well-known vocabularies has evolved in the Semantic Web community. The set of established vocabularies we reused is listed in Table 3.1.

Names	Details	Namespaces	URIs
OWL2	Web Ontology Language Schema 2.	owl	http://www.w3.org/2002/07/owl#
RDF	Resource Description Framework Concepts Vocabulary.	rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
	RDF Schema vocabulary.	rdfs	http://www.w3.org/2000/01/rdf-schema#
XML Schema Definition	Data types for elements and attributes.	xsd	http://www.w3.org/2001/XMLSchema#

[2] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/vocabularies.md

DCMI Metadata Terms	Defines general metadata attributes.	dcterms	http://purl.org/dc/terms/
VANN	Vocabulary for annotating descriptions of vocabularies with examples and usage notes.	vann	http://purl.org/vocab/vann/
Friend of a Friend	People and relationship.	foaf	http://xmlns.com/foaf/0.1/
Creative Commons	License terms.	cc	http://creativecommons.org/ns#
Geo Names	Geographic places.	gn	http://www.geonames.org/ontology#
Schema.org	Vocabulary for common concepts, i.e. TypeAndQuantityNode, GeoCoordinates, e.t.c.	schema	https://schema.org/
Good Relations	Product, price and company data.	gr	http://purl.org/goodrelations/v1#
QUDT	Units and measurements schema. Unit vocabulary.	qudt unit	http://qudt.org/schema/qudt# http://qudt.org/vocab/unit#
Semantic Sensor Network	SSN is an ontology for describing sensors and their observations, the studied features of interest, the samples used to do so, and the observed properties.	ssn	http://www.w3.org/ns/ssn/

System capabilities terms of SSN ontology.	ssn-system	http://www.w3.org/ns/ssn/systems/
Sample Observation and Actuator.	sosa	http://www.w3.org/ns/sosa/

Table 3.1: Vocabularies

Apart from vocabularies in Table 3.1 which we have used, we have considered the following vocabularies, but decided that they are not suitable for our work for the following reasons:

1. OWL-S [151]: It is useful when defining the input, output and interactions between web services, but CoCoOn is not modelling those.
2. Unified Code for Units of Measure (UCUM) [198] and Custom Datatypes [47]: These ontologies do not include the units needed for our use cases, which are presented in Section 3.3.3.8.1. Note that we have used another similar vocabulary QUDT.

3.3.3 Concepts and Design

In this section, we describe we describe the main components of CoCoOn v1.0.1 and explain its syntax, semantics, design and formalization, and the rationale behind CoCoOn's design, and some usages.

The classes and properties are arranged according to subsumption hierarchies, which represent the skeleton of the model and establish the basic relationships between the components. Following the principle of minimal commitment [79], we use guarded restrictions (i.e. owl:someValuesFrom) instead of domain range restrictions (rdfs:domain, rdfs:range). As such, the domain and ranges are more permissive, keeping the model more flexible and extensible. We also use qualified cardinality restrictions (e.g., exactly, owl:qualifiedCardinality; max, owl:maxQualifiedCardinality) when there is a known cardinality restriction.

Most building blocks of IaaS services naturally correspond to OWL2 classes (e.g., cocoon:CloudService, cocoon:ComputeService, and cocoon:StorageService), object properties (e.g., cocoon:hasMemory, cocoon:hasStorage, and cocoon:inRegion) and data properties (e.g., cocoon:numberOfCores). The more challenging part is to capture constraints posed by the possible combination of services in IaaS in the models using ontological axioms. We next describe how this can be accomplished using a combination of OWL 2 axioms and integrity constraints.

We use Turtle syntax [167] throughout our examples, and use Manchester OWL Syntax [128] when explaining the ontology specifications. We will show the prefix definitions once, then omit the duplicated definitions in the following examples.

3.3.3.1 Cloud Service

The class cocoon:CloudService is the main class hosting our Cloud feature vocabularies. We define a top level class cocoon:Service to be its parent, and make it the union of schema:Service and sosa:FeatureOfInterest. So our Cloud service definitions are compatible with the schema.org vocabulary [80] and the SOSA ontology [99] from which we reuse terms.

Cloud services are usually classified into three categories: cocoon:IaaS, cocoon:PaaS and cocoon:SaaS. Some examples of cocoon:SaaS are *database as a service*, *machine learning as a service*, Google Cloud Composer, etc. Some examples of cocoon:PaaS are the Google App Engine, Heroku, etc.

The following properties are defined for the class cocoon:CloudService:

gr:hasPriceSpecification some gr:UnitPriceSpecification

cocoon:hasProvider exactly 1 gr:BusinessEntity

cocoon:inRegion exactly 1 cocoon:Region

cocoon:inZone max 1 xsd:string

We use gr:UnitPriceSpecification and its associated object property gr:hasPriceSpecification to model price (see Section 3.3.3.5.1 for more details about price specification). Existential quantifiers (i.e., some, owl:someValuesFrom) are used on gr:hasPriceSpecification.

Note that, although some is the same as min 1, it is not the same as database integrity constraints. We can still define valid Cloud services without a price specification. Under the open world assumption, missing information is just not known but may exist, whereas, in databases (closed world assumption), absence of information often assumes that information does not exist. This open world assumption serves us well because we cannot guarantee that every service will have a price specification. There are services available upon requests, but the price is negotiated later. For example, we may want to specify that secure data centres for governmental use are available, but detailed price information is probably not disclosed publicly.

We assume each service can belong to exactly one provider. A qualified cardinality restriction exactly (owl:qualifiedCardinality) is used to define this type of assumption. We reuse gr:BusinessEntity to define a provider (see Section 3.3.3.8 for more details).

Infrastructure as a Service can be classified into 3 categories: cocoon:ComputeService (see Section 3.3.3.2), cocoon:StorageService (see Section 3.3.3.3), and cocoon:NetworkService (see Section 3.3.3.4).

3.3.3.2 Compute Service

We define the following properties for class cocoon:ComputeService:

gr:hasPriceSpecification max 1 cocoon:CloudStorageTransactionsPriceSpecification

cocoon:hasCPUcapacity max 1 schema:TypeAndQuantityNode

cocoon:hasMaxNumberOfDisks max 1 schema:TypeAndQuantityNode

cocoon:hasMaxStorageSize max 1 schema:TypeAndQuantityNode

cocoon:hasMemory exactly 1 schema:TypeAndQuantityNode

cocoon:hasStorage some schema:TypeAndQuantityNode

cocoon:numberOfCores max 1 xsd:decimal

The data property cocoon:numberOfCores defines the number of cores available on a virtual machine (VM). Because it can have a non-integer value, we define its datatype as xsd:decimal. For Google Cloud, cores and vCPU refer to the same thing. The performance power of the CPU can be described by cocoon:hasCPUcapacity. The memory size of a VM is specified by cocoon:hasMemory.

The cocoon:LocalStorage available on a VM can be specified with cocoon:hasStorage. We use an existential quantification (i.e. some) on this property, so that it is possible to define more cocoon:NetworkStorage later. Google has a limit for the maximum number of disks that can be attached to a VM, which we model with the object property cocoon:hasMaxNumberOfDisks. Additionally, Google also has a limit for the maximum total disk size that can be attached to a VM, which is modelled with cocoon:hasMaxStorageSize.

We use schema:TypeAndQuantityNode to describe the quantity of things. So value, unit, and type of an object can all be captured (see Section 3.3.3.8 for more details).

Note that cocoon:ComputeService also inherits properties from its super classes, e.g. the following property is inherited from cocoon:CloudService:

gr:hasPriceSpecification some gr:UnitPriceSpecification

There are data access fees on local disks of the Azure VM [49]. To model this we use gr:hasPriceSpecification max 1 cocoon:StorageTransactionsPriceSpecification. For a short example of cocoon:ComputeService, see Listing 3.1.

Listing 3.1: Virtual Machine

```
@prefix schema: <https://schema.org/> .
@prefix unit: <http://qudt.org/vocab/unit#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix gr: <http://purl.org/goodrelations/v1#> .
@prefix cocoon: <https://w3id.org/cocoon/v1.0.1#> .
@base <https://w3id.org/cocoon/data/v1.0.1/> .
<2019-02-12/ComputeService/Gcloud/CP-COMPUTEENGINE-VMIMAGE-N1-HIGHCPU-96-PREEMPTIBLE>
       a cocoon:ComputeService ;
       rdfs:label "CP-COMPUTEENGINE-VMIMAGE-N1-HIGHCPU-96-PREEMPTIBLE" ;
       gr:hasPriceSpecification [ a cocoon:CloudServicePriceSpecification ;
                                  gr:hasCurrency "USD" ;
                                  gr:hasCurrencyValue 0.72 ;
                                  gr:hasUnitOfMeasurement unit:Hour ;
                                  cocoon:inRegion <Region/Gcloud/us-east1>
                                ] ;
```

```
        cocoon:hasMemory [ a schema:TypeAndQuantityNode ;
                           schema:amountOfThisGood 86.4 ;
                           schema:unitCode cocoon:GB
                         ] ;
        cocoon:hasProvider cocoon:Gcloud ;
        cocoon:numberOfCores "96"^^xsd:decimal ;
        schema:dateModified "2019-02-12"^^xsd:date .
```

3.3.3.3 Storage Service

We define the following properties for the class cocoon:StorageService:

cocoon:canHaveSnapshot some cocoon:StorageService

cocoon:hasStorageIOMax max 1 schema:TypeAndQuantityNode

cocoon:hasStorageSize max 1 schema:TypeAndQuantityNode

cocoon:hasStorageThroughputMax max 1 schema:TypeAndQuantityNode

Two subclasses for cocoon:StorageService have been defined: cocoon:LocalStorage and cocoon:NetworkStorage.

On the Azure Cloud, snapshot options are available for storage, which is modelled with the object property cocoon:canHaveSnapshot. This information is manually interpreted from the documentation.[3] There are also caps on input/output operations per sec (IOPS) and throughput, which are modeled with cocoon:hasStorageIOMax and cocoon:hasStorageThroughputMax. We have also defined corresponding units, which is explained in Section 3.3.3.8.

In Listing 3.2, we show a cocoon:NetworkStorage service from cocoon:Azure, which is a Cloud provider we have pre-defined as a named instance. More details on its corresponding storage transaction prices can be found in Section 3.3.3.5.3.

Next, an example of the Azure provisional Ultra SSD storage service is presented. It has configurable IOPS and throughput. Prices are based on provisioned storage size, IOPS and throughput. There is also a reservation charge imposed if you enable Ultra SSD capability on the VM without connecting an Ultra SSD disk, whose rate is provisioned at per vcpu/hour.

Listing 3.2: Storage

```
@base <https://w3id.org/cocoon/data/v1.0.1/> .
<2019-03-07/NetworkStorage/Azure/premiumssd-p30>
        a cocoon:NetworkStorage ;
        rdfs:label "premiumssd-p30" ;
        gr:hasPriceSpecification <CloudStorageTransactionsPriceSpecification/Azure/managed_disk/
            ↪ transactions-ssd> ;
        gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                   gr:hasCurrency "USD" ;
                                   gr:hasCurrencyValue 0.13200195133686066 ;
                                   gr:hasUnitOfMeasurement cocoon:GBPerMonth ;
                                   cocoon:inRegion <Region/Azure/australia-east>
```

[3] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/azure/storage.md#disk-snapshots

```
                                ] ;
        cocoon:canHaveSnapshot <NetworkStorage/Azure/standardssd-snapshot> , <NetworkStorage/
             ↪ Azure/standardhdd-snapshot-zrs> , <NetworkStorage/Azure/premiumssd-snapshot> , </
             ↪ NetworkStorage/Azure/standardhdd-snapshot-lrs> ;
        cocoon:hasProvider cocoon:Azure ;
        cocoon:hasStorageIOMax [ a schema:TypeAndQuantityNode ;
                                 schema:amountOfThisGood "5000"^^xsd:nonNegativeInteger ;
                                 schema:unitCode cocoon:IOPs
                                 ] ;
        cocoon:hasStorageSize [ a schema:TypeAndQuantityNode ;
                                 schema:amountOfThisGood "1024"^^xsd:nonNegativeInteger ;
                                 schema:unitCode cocoon:GB
                                 ] ;
        cocoon:hasStorageThroughputMax [ a schema:TypeAndQuantityNode ;
                                 schema:amountOfThisGood "200"^^xsd:nonNegativeInteger ;
                                 schema:unitCode unit:MegabitsPerSecond
                                 ].

<2019-03-07/NetworkStorage/Azure/ultrassd>
        a cocoon:NetworkStorage ;
        rdfs:label "ultrassd" ;
        gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                   rdfs:label "vcpu" ;
                                   gr:hasCurrency "USD" ;
                                   gr:hasCurrencyValue 0.003 ;
                                   gr:hasUnitOfMeasurement cocoon:VcpuPerHour ;
                                   cocoon:inRegion <Region/Azure/us-east-2>
                                 ] ;
        gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                   rdfs:label "throughput" ;
                                   gr:hasCurrency "USD" ;
                                   gr:hasCurrencyValue 0.000685 ;
                                   gr:hasUnitOfMeasurement cocoon:MegabitsPerSecondPerHour ;
                                   cocoon:inRegion <Region/Azure/us-east-2>
                                 ] ;
        gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                   rdfs:label "stored" ;
                                   gr:hasCurrency "USD" ;
                                   gr:hasCurrencyValue 0.000082 ;
                                   gr:hasUnitOfMeasurement cocoon:GBPerHour ;
                                   cocoon:inRegion <Region/Azure/us-east-2>
                                 ] ;
        gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                   rdfs:label "iops" ;
                                   gr:hasCurrency "USD" ;
                                   gr:hasCurrencyValue 0.000034 ;
                                   gr:hasUnitOfMeasurement cocoon:IOPsPerHour ;
                                   cocoon:inRegion <Region/Azure/us-east-2>
                                 ] .
```

3.3.3.4 Network Service

We classify network services into the following categories: cocoon:InternetService, cocoon:LoadBalancing, cocoon:StaticIPService and cocoon:DNSService.

3.3.3.4.1 Internet Service

We define the following properties for the class cocoon:InternetService:

cocoon:excludesDestination some cocoon:Location

cocoon:hasDestination some cocoon:Location

cocoon:hasDirection exactly 1 cocoon:TrafficDirection

There is generally no charge to ingress cocoon:InternetService, unless there is a load balancer used. We use the cocoon:hasDirection object property to indicate the direction of traffic. A class cocoon:TrafficDirection is also defined with two disjoint subclasses, cocoon:Egress and cocoon:Ingress. Those can be used to indicate the direction of traffic.

Internet egress rates are based on usage and destination. For example, Google Cloud has three destination categories[4]: Australia, China (excluding Hong Kong) and Worldwide (excluding China and Australia, but including Hong Kong). In this case, the object properties cocoon:hasDestination and cocoon:excludesDestination can be used to specify destination ranges. Because Cloud service regions do not constrain traffic destinations, cocoon:Location is used, which has more explanations in Section 3.3.3.7.

The internet egress traffic rates can be modelled by cocoon:CloudNetworkPriceSpecification. For more details, see Section 3.3.3.5.4.[5]

3.3.3.4.2 Load Balancing

Both hardware and software-based load balancing solutions exist. Here we consider load balancing as a hardware feature unless it is known otherwise. We create a class cocoon:LoadBalancing to represent such a service. It is further broken down into two subclasses: cocoon:LoadBalancingData and cocoon:ForwardingRule.

Ingress data processed by a load balancer is charged (per GB) based on its region. Listing 3.3 models such cases with cocoon:LoadBalancingData.

Listing 3.3: Load Balancing Data Price Specification

```
@base <https://w3id.org/cocoon/data/v1.0.1/2019-02-12/> .
<LoadBalancingData/Gcloud>
        a cocoon:LoadBalancingData ;
        gr:hasPriceSpecification [ a gr:CloudServicePriceSpecification ;
                                   gr:hasCurrency "USD" ;
                                   gr:hasCurrencyValue 0.008 ;
                                   gr:hasUnitOfMeasurement cocoon:GB ;
                                   cocoon:inRegion <Region/Gcloud/us>
                                 ] ;
        cocoon:hasDirection cocoon:Ingress ;
        cocoon:hasProvider cocoon:Gcloud ;
        schema:dateModified "2019-02-12"^^xsd:date .
```

Forwarding rules that are created for load balancing are also charged on an hourly base, regardless of how many forwards. This can be modelled by co-

[4]Effective until end of June 2019, when this paper has been submitted, after that new pricing takes effect based on not only the destination but also the sources.

[5]Example for Google Cloud internet services prices: https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#internet-service

coon:ForwardingRule and cocoon:CloudNetworkPriceSpecification.[6] For example, from Google Cloud documentation, the pricing for rules is:

> Up to 5 forwarding rules are charged at 0.025 USD/hour. That means, if you create one forwarding rule, you will be charged 0.025/hour. If you have three forwarding rules, you will still be charged 0.025/hour. However, if you have ten forwarding rules, you will be charged:
>
> 5 forwarding rules = 0.025/hour
>
> Each additional forwarding rule = 0.01/hour
>
> 0.025/hour for 5 rules + (5 additional rules * 0.01/hour) = 0.075/hour

3.3.3.4.3 Static IP Address

The IP address of a VM instance usually is not guaranteed to stay the same between reboots/resets. So you may want to reserve a static external IP address for your customers or users to have reliable access. An example from Google:

> No charge for an IP address in use, i.e. used it with a Compute Engine resource or a forwarding rule. However, Google will charge 0.010 USD/Hour if you reserve a static external IP address but do not use it.

This can be modelled with cocoon:StaticIPService and cocoon:CloudServicePriceSpecification.

3.3.3.5 Cloud Service Price

For price modelling, we extend the GoodRelations vocabulary [87]. GoodRelations is a Web Ontology Language-compliant ontology for Semantic Web online data, dealing with business-related goods and services. In November 2012, it was integrated into the Schema.org ontology.

3.3.3.5.1 Cloud Service Price Specification

We define cocoon:CloudServicePriceSpecification as a subclass of gr:UnitPriceSpecification. As one service can be offered in multiple regions, we extend our specialized class with: cocoon:inRegion some cocoon:Region. For more details on region, see Section 3.3.3.7.

In GoodRelations, there is a gr:hasCurrencyValue property taking a xsd:float as range. However, floats can introduce cumulative rounding errors. So we extend the existing class to allow xsd:decimal, which can represent exact monetary

[6] Forwarding rules example: https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#forwarding-rule

values: cocoon:hasCurrencyValue exactly 1 xsd:decimal.[7] For more usages, see Section 3.3.3.2. We also define specialized subclasses to handle the following scenarios: price of a VM image (cocoon:CloudOSPriceSpecification), price of storage transactions (cocoon:CloudStorageTransactionsPriceSpecification), and price of network services (cocoon:CloudNetworkPriceSpecification). These sub-classes are owl:disjointWith each other. Because each case has very different requirements, it is clearer to model them with different subclasses rather than define all properties in the base class cocoon:CloudServicePriceSpecification.

3.3.3.5.2 Price of Virtual Machine Images

cocoon:CloudOSPriceSpecification is defined with the following properties:

cocoon:chargedPerCore exactly 1 xsd:boolean

cocoon:forCoresLessEqual max 1 xsd:decimal

cocoon:forCoresMoreThan max 1 xsd:decimal

For example, from the Google Cloud documentation, the following charges apply for images:

> The price for a premium image is different depending on which machine type you use. For example, a standard SUSE image costs $0.02 per hour to run on an f1-micro instance, but the same image costs $0.11 per hour to run on an n1-standard-8 instance. The prices for premium images are the same worldwide and **do not differ based on zones or regions**. All prices for premium images are in addition to charges for using a machine type. For example, the total price to use an n1-standard-8 instance with a SUSE image would be the sum of the machine type cost and the image cost: n1-standard-8 cost + SUSE image cost = $0.3800 + $0.11 = $0.49 per hour

Under the class cocoon:CloudOSPriceSpecification, the data property cocoon:chargedPerCore specifies if the price is charged per core. For instance, Windows Server images on some machine types from Google Cloud are charged based on the number of CPUs available [75], i.e., n1-standard-4, n1-highcpu-4, and n1-highmem-4 are machine-types with four vCPUs, and are charged at $0.16 USD/hour (4 $\times$ $0.04 USD/hour).

The data property cocoon:forCoresMoreThan is used to describe a price for machines with more than the specified number of cores. Similarly, cocoon:forCoresLessEqual is used to describe a price for machines with less than or equal to the specified number of cores. They can be used together to quantify a range. Listing 3.4 presents an example for OS Price Specification.

[7] Example modelling of the price of Azure Compute service: https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#cloud-service-price-specification

Listing 3.4: OS Price Specification

```
@base <https://w3id.org/cocoon/data/v1.0.1/2019-02-12/> .
<SystemImage/Gcloud/suse-sap>
        a cocoon:SystemImage ;
        rdfs:label "suse-sap" ;
        gr:hasPriceSpecification [ a cocoon:CloudOSPriceSpecification ;
                                   gr:hasCurrency "USD" ;
                                   gr:hasCurrencyValue 0.41 ;
                                   cocoon:chargedPerCore false ;
                                   cocoon:forCoresMoreThan "4"^^xsd:decimal
                                 ] ;
        gr:hasPriceSpecification [ a cocoon:CloudOSPriceSpecification ;
                                   gr:hasCurrency "USD" ;
                                   gr:hasCurrencyValue 0.34 ;
                                   cocoon:chargedPerCore false ;
                                   cocoon:forCoresLessEqual "4"^^xsd:decimal ;
                                   cocoon:forCoresMoreThan "2"^^xsd:decimal
                                 ] ;
        gr:hasPriceSpecification [ a cocoon:CloudOSPriceSpecification ;
                                   gr:hasCurrency "USD" ;
                                   gr:hasCurrencyValue 0.17 ;
                                   cocoon:chargedPerCore false ;
                                   cocoon:forCoresLessEqual "2"^^xsd:decimal
                                 ] .
```

3.3.3.5.3 Price of Storage Transactions

For storage transactions, we use the class cocoon:CloudStorageTransactionsPriceSpecification to define the price. There are different prices in different regions, but there is a common transaction price specification for a group of cloud storage offers.[8]

An example from Microsoft Azure [49] on data access fees for local disks: "Every single block access incurs a transaction. The default block size is 4 Megabytes, meaning uploading a 32Mb file will incur 8 Storage Transactions. Deleting and updating the file will also each incur eight transactions, the same goes for any other time the file is touched. The transactions are charged at the cost of around $0.00036 AUD per 10,000 transactions. As such, a 32Mb file will cost $0.000000368 AUD. The only exception to Storage Transactions is when Premium Storage (persistent SSD storage) is used. That is when you provision a P10, P20 or a P30 disk for your virtual machine. Those disks are exempt from Storage Transactions."

3.3.3.5.4 Price of Network Services

cocoon:CloudNetworkPriceSpecification is defined with the following properties:

cocoon:forUsageLessEqual exactly 1 schema:TypeAndQuantityNode

cocoon:forUsageMoreThan exactly 1 schema:TypeAndQuantityNode

cocoon:specialRateType exactly 1 xsd:string

[8] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#cloud-storage-transactions-price-specification

cocoon:CloudNetworkPriceSpecification can be used to model network services prices, including internet egress traffic and load balancing forwarding rules.

For instance, there are three (monthly) usage tiers for Google Internet egress traffic price: 0-1 TB, 1-10 TB and 10+ TB. Properties cocoon:forUsageLessEqual and cocoon:forUsageMoreThan can be used to specify the upper/lower usage limits. We combine this with schema:TypeAndQuantityNode to define the values with their units.

There are also some special rates, e.g., for Google Cloud Internet Traffic:

1. Egress between zones in the same region (per GB) is 0.01.
2. Egress between regions within the US (per GB) is 0.01.
3. Egress to Google products (such as YouTube, Maps amd Drive), whether from a VM in GCP with an external IP address or an internal IP address is no charge.

The property cocoon:specialRateType can be used to model those situations. See an online example for price of Google internet egress between zones in the same region.[9]

3.3.3.6 Cloud Service Performance

In the area of QoS modelling, some papers have proposed QoS ontologies (i.e. QoSOnt [55] and OWL-QoS [237]). However, they did not publish the actual specifications, and only figures/graphs were given. In this section, we provide formal modelling of QoS parameters and make it readily available for general use. Our classes to model QoS parameters are grouped under cocoon:QualityOfService. We also use terms from many ontologies when modelling QoS, such as SSN [82] and SOSA [99]. The Semantic Sensor Network (SSN) ontology is an ontology for describing sensors and their observations, involved procedures, studied features of interest, samples, and observed properties, as well as actuators. SSN includes a lightweight but self-contained core ontology called SOSA (Sensor, Observation, Sample, and Actuator) for its elementary classes and properties. "SSN System" contains the terms defined for system capabilities, operating ranges, and survival ranges.

3.3.3.6.1 Quality Of Service Property

QoS parameters are grouped under cocoon:QualityOfService. We define cocoon:QualityOfService to be an equivalent class of ssn-system:SystemProperty. Then we extend it with the subclass cocoon:DataTransferSpeed.

3.3.3.6.1.1 Data Transfer Speed cocoon:DataTransferSpeed is measured multiple times with different file sizes, for both the uplink and downlink, which are represented by subclasses cocoon:DownlinkSpeed and cocoon:UplinkSpeed. The common properties for cocoon:DataTransferSpeed are defined as follows:

[9] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#cloud-network-price-specification

cocoon:transferedFileSizeMax exactly 1 schema:TypeAndQuantityNode

cocoon:transferedFileSizeMin exactly 1 schema:TypeAndQuantityNode

See an example online.[10]

Listing 3.5 shows some examples. For instance, the downlink speed when transferring a file with a size between 1 KB and 128 KB, the uplink speed when transferring a file with size between 256 KB and 10240 KB, etc. Note that, this only describes the properties which are being measured. The actual values are modeled with cocoon:Measurement, which is explained in Section 3.3.3.6.2.

Listing 3.5: QoS Properties

```
@base <https://w3id.org/cocoon/data/v1.0.1/> .
<1-KB> a schema:TypeAndQuantityNode;
    schema:amountOfThisGood "1"^^xsd:interger;
    schema:unitText "KB";
    schema:unitCode "2P".

<128-KB> a schema:TypeAndQuantityNode;
    schema:amountOfThisGood "128"^^xsd:interger;
    schema:unitText "KB";
    schema:unitCode "2P".

<256-KB> a schema:TypeAndQuantityNode;
    schema:amountOfThisGood "256"^^xsd:interger;
    schema:unitText "KB";
    schema:unitCode "2P".

<10240-KB> a schema:TypeAndQuantityNode;
    schema:amountOfThisGood "10240"^^xsd:interger;
    schema:unitText "KB";
    schema:unitCode "2P".

<QualityOfService/DownlinkSpeed-1-128-KB> a cocoon:DownlinkSpeed;
    cocoon:transferedFileSizeMin <1-KB>;
    cocoon:transferedFileSizeMax <128-KB>.

<QualityOfService/DownlinkSpeed-256-10240-KB> a cocoon:DownlinkSpeed;
    cocoon:transferedFileSizeMin <256-KB>;
    cocoon:transferedFileSizeMax <10240-KB>.

<QualityOfService/UplinkSpeed-1-128-KB> a cocoon:UplinkSpeed;
    cocoon:transferedFileSizeMin <1-KB>;
    cocoon:transferedFileSizeMax <128-KB>.

<QualityOfService/UplinkSpeed-256-10240-KB> a cocoon:UplinkSpeed;
    cocoon:transferedFileSizeMin <256-KB>;
    cocoon:transferedFileSizeMax <10240-KB>.
```

3.3.3.6.1.2 Latency There is an existing ssn-system:Latency class, which we can use. We extend this class with a specialized subclass cocoon:DNSQueryLatency, which is the latency for completing the DNS query. The term latency is most commonly referred to as the round-trip delay time, which is the one-way latency for the request

[10] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#downlink-speed

to travel from a source to a destination plus the one-way latency for the response to travel back.

3.3.3.6.2 Measurement

QoS measurements are modeled with cocoon:Measurement, which is an equivalent class to sosa:Observation. The cocoon:Measurement can use sosa:hasFeatureOfInterest to specify which feature it measures. Since cocoon:Service is equivalent to sosa:FeatureOfInterest, all its subclasses can be used to describe features, and we have some examples can be viewed online.[11]

3.3.3.6.3 Device

We extend sosa:Sensor with a subclass cocoon:Device to describe computers used to measure QoS. The additional properties we define are:

cocoon:inPhysicalLocation max 1 cocoon:Location

cocoon:ipv4 some xsd:string

Listing 3.6 shows an example for the device.

Listing 3.6: Device

```
@base <https://w3id.org/cocoon/data/v1.0.1/> .
<Device/150.203.213.249/lat=-35.271475/long=149.121434>
       a cocoon:Device ;
       rdfs:comment "The computer used to conduct the tests, belongs to Australian National
            ↪ University College of Engineering & Computer Science."@en ;
       rdfs:label "CECS-030929"@en ;
       cocoon:inPhysicalLocation [ a schema:Place ;
                                   schema:geo [ a schema:GeoCoordinates ;
                                                schema:address "Hanna Neumann Building #145,
                                                     ↪ Science Road, Canberra ACT 2601" ;
                                                schema:latitude -35.271475 ;
                                                schema:longitude 149.121434
                                              ]
                                 ] ;
       cocoon:ipv4 "150.203.213.249" .
```

3.3.3.7 Location and Region

cocoon:Location is a permissible class that can be used to represent any location, i.e. Worldwide, Australia and Hong Kong. In comparison, cocoon:Region, the subclass of cocoon:Location, is more specialized to represent known/predefined cloud service regions.

We link regions from each Cloud provider to GeoNames data [68], and at the same time make it compatible with Schema.org. So we define it as the union of gn:Feature and schema:Place.

cocoon:Region is defined with the following properties:

[11] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#measurement

cocoon:city max 1 xsd:string

cocoon:continent max 1 xsd:string

cocoon:hasProvider exactly 1 gr:businessEntity

cocoon:inJurisdiction some cocoon:Location

cocoon:inPhysicalLocation max 1 cocoon:Location

cocoon:state max 1 xsd:string

If a specific location or address is known, a physical location can be set with cocoon:inPhysicalLocation. Otherwise, we only describe the approximate location with cocoon:inJurisdiction. Some regions can be in multiple jurisdictions, i.e. `nam-eur-asia1` belongs to North America, Europe, and Asia. Typically, a region cannot be in more than one physical location. An example for region is shown in Listing 3.7.

Listing 3.7: Regions

```
@base <https://w3id.org/cocoon/data/v1.0.1/> .
<Region/Gcloud/asia-southeast1>
       a cocoon:Region ;
       rdfs:label "asia-southeast1" ;
       cocoon:city "Singapore" ;
       cocoon:hasProvider cocoon:Gcloud ;
       schema:dateModified "2019-02-12"^^xsd:date ;
       schema:geo [ a schema:GeoCoordinates ;
                    schema:addressCountry "SG" ;
                    schema:latitude 1.3521 ;
                    schema:longitude 103.8198
                  ] .
```

Each region can also specify which cocoon:continent it is in, which provider it belongs to (with cocoon:hasProvider), and a human readable name with rdfs:label. Currently, there is a simple script written for matching a region to a gn:Feature, but it can be further optimized in future work.

We have also obtained some geographic coordinates from the QoS measurements, and modelled such information with schema:geo and schema:GeoCoordinates. More examples are available online.[12]

3.3.3.8 Named Individuals

We define a number of useful named individuals to be included in this ontology.

3.3.3.8.1 Unit

We define cocoon:UnitOfMeasure as an owl:equivalentClass of qudt:Unit, and then use the instances from the unit vocabulary, i.e. unit:Hour and

[12] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/quickstart.md#location-and-region

unit:MegabitsPerSecond. We also define a number of custom units with reference to qudt:InformationEntropyUnit and qudt:DataRateUnit, i.e., cocoon:GB, cocoon:GBPerHour, cocoon:GBPerMonth, cocoon:GCEU (which is the Google Compute Engine Unit), cocoon:IOPs, cocoon:IOPsPerHour, cocoon:MegabitsPerSecondPerHour, cocoon:TB, and cocoon:VcpuPerHour. See examples in Listing 3.8.

Listing 3.8: Units

```
### https://w3id.org/cocoon/v1.0.1#GB
cocoon:GB rdf:type owl:NamedIndividual ,
                  qudt:InformationEntropyUnit ,
                  cocoon:UnitOfMeasure ;
          rdfs:comment "Gigabyte (GB): There are 1024MB in one gigabyte."@en ;
          rdfs:label "Gigabyte"@en .


### https://w3id.org/cocoon/v1.0.1#GBPerHour
cocoon:GBPerHour rdf:type owl:NamedIndividual ,
                         qudt:DataRateUnit ,
                         cocoon:UnitOfMeasure ;
                 rdfs:comment "Often describe 1 GB per Hour usage."@en ;
                 rdfs:label "Gigabyte per Hour"@en .


### https://w3id.org/cocoon/v1.0.1#GBPerMonth
cocoon:GBPerMonth rdf:type owl:NamedIndividual ,
                          qudt:DataRateUnit ,
                          cocoon:UnitOfMeasure ;
                  rdfs:comment "Often describe 1 GB per Month usage."@en ;
                  rdfs:label "Gigabyte per Month"@en .


### https://w3id.org/cocoon/v1.0.1#GCEU
cocoon:GCEU rdf:type owl:NamedIndividual ,
                    cocoon:UnitOfMeasure ;
            rdfs:comment "GCEU (Google Compute Engine Unit), or GQ for short, is a unit of CPU
                 ↪ capacity that we use to describe the compute power of our instance types. We
                 ↪ chose 2.75 GQ's to represent the minimum power of one logical core (a hardware
                 ↪  hyper-thread) on our Sandy Bridge platform."@en ;
            rdfs:label "Google Compute Engine Unit"@en .


### https://w3id.org/cocoon/v1.0.1#Gcloud
cocoon:Gcloud rdf:type owl:NamedIndividual ,
                      gr:BusinessEntity ;
              gr:name "Google Cloud" ;
              rdfs:label "Gcloud"@en ;
              foaf:page <https://cloud.google.com/> .


### https://w3id.org/cocoon/v1.0.1#IOPs
cocoon:IOPs rdf:type owl:NamedIndividual ,
                    qudt:DataRateUnit ,
                    cocoon:UnitOfMeasure ;
            rdfs:comment "Azure Managed Disks provide different input/output operations per sec (
                 ↪ IOPs)"@en ;
            rdfs:label "Input/output operations per sec"@en .


### https://w3id.org/cocoon/v1.0.1#IOPsPerHour
cocoon:IOPsPerHour rdf:type owl:NamedIndividual ,
```

```
                        qudt:DataRateUnit ,
                        cocoon:UnitOfMeasure ;
                rdfs:comment "Azure Ultra SSD Managed Disks come in different sizes that
                    ↪ provide a configurable range of input/output operations per sec (IOPS),
                    ↪  and are billed on an hourly rate."@en ;
                rdfs:label "IOPS/hour"@en .


### https://w3id.org/cocoon/v1.0.1#MegabitsPerSecondPerHour
cocoon:MegabitsPerSecondPerHour rdf:type owl:NamedIndividual ,
                                        qudt:DataRateUnit ,
                                        cocoon:UnitOfMeasure ;
                                rdfs:comment "Often describe the throughput usage measured at MB/s
                                    ↪ within an hour."@en ;
                                rdfs:label "MB/s/hour"@en ..

### https://w3id.org/cocoon/v1.0.1#TB
cocoon:TB rdf:type owl:NamedIndividual ,
                qudt:InformationEntropyUnit ,
                cocoon:UnitOfMeasure ;
        rdfs:comment "Terabyte is more precisely defined as 1,024 gigabytes (GB)"@en ;
        rdfs:label "Terabyte"@en .


### https://w3id.org/cocoon/v1.0.1#VcpuPerHour
cocoon:VcpuPerHour rdf:type owl:NamedIndividual ,
                        cocoon:UnitOfMeasure ;
                rdfs:comment "Azure Ultra SSD storage is in preview in East US 2 and is billed
                    ↪ on vcpu/hour provisioned as reservation charge. This reservation charge
                    ↪  is only imposed if you enable Ultra SSD capability on the VM without
                    ↪ connecting an Ultra SSD disk."@en ;
                rdfs:label "vcpu/hour"@en .
```

3.3.3.8.2 Provider

We define providers as a gr:BusinessEntity, such as cocoon:Gcloud and cocoon:Azure, which are shown in Listing 3.9.

Listing 3.9: Provider

```
cocoon:Azure rdf:type owl:NamedIndividual ,
                    gr:BusinessEntity ;
            gr:name "Microsoft Azure Cloud" ;
            rdfs:label "Azure"@en ;
            foaf:page <https://azure.microsoft.com/> .

cocoon:Gcloud rdf:type owl:NamedIndividual ,
                     gr:BusinessEntity ;
             gr:name "Google Cloud" ;
             rdfs:label "Gcloud"@en ;
             foaf:page <https://cloud.google.com/> .
```

3.3.3.8.3 Quantity and Type

We define some frequently used quantities as named individuals, using schema:TypeAndQuantityNode, i.e. cocoon:1TB. This will save us from redefining each value whenever we use it. Since schema:unitCode can take schema:URL, it means we can pass in any external defined units, i.e. cocoon:UnitOfMeasure.

3.4 Summary

Since the semantic web is a relatively new technology, existing ontologies only broadly define a service and does not cover the IaaS specifics. To fill in the gap, we proposed CoCoOn that formalizes the domain knowledge of cloud infrastructure services based on RDF. There are four main modules in CoCoOn: IaaS, QoS, Pricing and Region. IaaS module models Compute, Storage and Network Cloud services. Pricing module models price and unit information. Region module models datacenter location. QoS module models the quality of service information.

Since existing cloud providers can incorporate RDF annotation into their web pages to make them indexable by search engines. We converted unstructured text data on commonly available infrastructure services (e.g. Amazon, Microsoft Azure, and GoGrid) to structured RDF data, which is queryable with SPARQL. In doing so, CoCoOn provides cloud service ontology to simplify the presentation, sharing, searching and comparison of cloud services and their QoS and Pricing features.

Chapter 4

Ranking and Recommendation

In Chapter 3, we have developed a unified domain model capable of adequately describing infrastructure services in Cloud computing. In this chapter, we build upon this previous work to further investigate how to achieve high automation on service discovery.

4.1 Motivation

While the elastic nature of Cloud services makes it suitable for provisioning all kinds of applications, the heterogeneity of Cloud service configurations and their distributed nature raise some serious technical challenges. The Cloud computing landscape is evolving with multiple and diverse options for compute (also known as virtual machines) and storage services. Hence, application owners are facing a daunting task when trying to select Cloud services that can meet their constraints. As we mentioned in the introduction, Amazon Web Service (AWS) had 674 different offerings differentiated by price, features and location (in Burstorm's 2013 study [29]). Our evaluation in May 2019 found 17871 Azure offerings in 38 regions and 1467 Google Cloud offers in 26 regions. Application owners must simultaneously consider and optimize complex dependencies and different sets of criteria (price, features, location, QoS) when selecting the best mix of service offerings from an abundance of possibilities. For instance, it is not enough to just consider the suitability of storage services, but also essential to guarantee that the corresponding computing capabilities are enough to process data as fast as possible while minimizing cost.

There are methods proposed for network-aware service composition[231, 24, 239] considering generic web services, i.e., at the SaaS and PaaS levels. However, the compatibility constraints at the IaaS level are different from web services. For example, generic web services are distinguished by their features, QoS, and prices. It does not make sense to include two same services in one composition as one job does not need to be done twice, but using multiple quantities of an IaaS offering is perfectly valid.

4.1.1 The Problem of Varied Pricing Model

Varied pricing models of different Cloud providers is one of the problems. Another example of disparity is the model of "on-demand instances". Although GoGrid's plan has a similar concept to Amazon's on-demand and reserved instance, it gives very little importance to what type or how many of compute services a user is deploying. GoGrid charges users based on what they call RAM hours – 1 GB RAM compute service deployed for 1 hour consumes 1 RAM Hour. This means a 2 GB RAM compute service deployed for 1 hour consumes 2 RAM Hour. It is worthwhile mentioning that only Azure clearly states that one month is considered to have 31 days. This is important as the key advantage of the fine-grained pay-as-you-go price model which, for example, should charge a user the same when they use 2GB for half a month or 1 GB for a whole month. Other vendors merely give a GB-month price without clarifying how short term usage is handled. It is neither reflected in their usage calculator.

Regarding storage services, providers charge for every operation that an application program or a user undertakes. These operations can also be generated via RESTful APIs or Simple Object Access Protocol (SOAP) API. Furthermore, Cloud providers refer to the same set of operations with different names; for example, Azure refers to storage service operations as transactions. Nevertheless, the operations are categorized into upload and download categories as shown in Table 4.1. Red means an access fee is charged; Green means the service is free, and yellow means access fees are not specified, and can usually be treated as green/free of charge. To facilitate our calculation of similar and equivalent requests across multiple providers, we analyzed and pre-processed the price data. We recorded it in our domain model and used a homogenized value in the CloudRecommender. For example, Windows Azure Storage charges a flat price per transaction. It is considered as transaction whenever there is a "touch" operation, i.e. Create, Read, Update, Delete (CRUD) operation over the RESTful service interface, on any component (Blobs, Tables or Queues) of Windows Azure Storage.

We collected service configuration information from many public Cloud providers: Windows Azure, Amazon, GoGrid, RackSpace, Nirvanix, Ninefold, SoftLayer, AT and T Synaptic, Cloud Central, etc. It is show in Table 4.2. This table depicts the different pricing schemes and varied terminologies of providers.

4.1.2 The Need to Incorporate Quality of Service Constraints

Cloud computing embraces an elastic paradigm where applications establish on-demand interactions with services to satisfy QoS requirements such as response time, throughput, availability and reliability. QoS targets are encoded in the Legal Service Level Agreement (SLA) documents, which state the nature and scope of the QoS parameters. However, selecting and composing the right services to meet application requirements are challenging problems.

For different application deployment scenarios (e.g., multimedia, eResearch, and enterprise applications), the question is: How to satisfy QoS requirements across

Table 4.1: Heterogeneities in Request Types Across Storage Services

Provider	Storage	Requests		
		Upload	Download	Other
Windows Azure	Azure Storage	storage transactions	storage transactions	
Amazon	S3	PUT, COPY, POST, or LIST Reque	GET and all other Requests	Delete
GoGrid	Cloud Storage	Transfer protocols such as SCP, SAMBA/CIFS, and RSYNC		
RackSpace	Cloud Files	PUT, POST, LIST Requests	HEAD, GET, DELETE Requests	
Nirvanix	CSN		Search	
Ninefold	Cloud Storage	GET, PUT, POST, COPY, LIST and all other transactions		
SoftLayer	Object Storage	Not Specified/Unknow		
AT and T Synaptic	Storage as a Service	Not Specified/Unknow		
Cloudcentral	Object Storage	GET, PUT		

Red means an access fee is charged.
green means the service is free.
yellow means access fees are not specified, and can usually be treated as free of charge.

the layers? Notably, QoS aware service selection problem [98] is a multi-criteria optimization problem. In order to solve it, we can employ a multi-criteria decision-making technique, which will be explained in Section 4.3.1.

4.1.3 Applications Use Case Examples

We next provide a few examples to demonstrate different types of applications with the needs to cater for real-time QoS requirements during their deployment lifecycle.

Interactive Online Games: In the gaming industry, World of Warcraft counts over six million unique players daily. The operating infrastructure of this Massively Multiplayer Online Role-Playing Game (MMORPG) comprises more than 10,000 computers [142]. Depending on the game, typical response times to ensure fluent play must remain below 100 milliseconds for online First Person Shooter (FPS) action games [22] and below 1-2 seconds for Role-Playing Games (RPGs). The excellent game experience is critical for keeping the players engaged, and has an immediate consequence on the earnings and the popularity of the game operators. Failing to deliver timely simulation updates leads to degraded game experience and triggers player departure and account closures [185]. Startup gaming company with no existing infrastructure could launch a new game platform using a public Cloud infrastructure. Cloud services offer the flexibility to scale on demand with no upfront investment. Using Cloud services, the game application services can be dynamically allocated or de-allocated according to demand fluctuations. Game companies can also better serve diverse international users with the global presence of data centres owned by

Table 4.2: Pricing and Terminology Heterogeneities in Compute and Storage Services Across Providers

Provider	Compute	Pay As You Go	Other Plans*	Storage	Pay As You Go	Other Plans*	Trail
	Terminology	Unit		Terminology	Unit		Period or Value
Windows Azure	Virtual Server	/hr	1	Azure Storage	/GB month	1	90 day
Amazon	EC2 Instance	/hr	2	S3	/GB month	2	1 year
GoGrid	Cloud Servers	/RAM hr	1	Cloud Storage	/GB month		
RackSpace	Cloud Servers	/RAM hr		Cloud Files	/GB month		
Nirvanix				CSN	/GB month		
Ninefold	Virtual Server	/hr		Cloud Storage	/GB month	1	50 AUD
SoftLayer	Cloud Servers	/hr	1	Object Storage	/GB		
AT and T Synaptic	Compute as a Service	vCPU per hour +/RAM hr		Storage as a Service	/GB month		
Cloudcentral	Cloud Servers	/hr					

* Other Plans includes Monthly/Quarterly/Yearly Plan, Reserve and Bidding Price Option.
Red blank cells in the table mean that a configuration parameter is not supported. Some providers offer their services under a different pricing scheme than pay-as-you-go, and we refer to these schemes as other plans (e.g. Amazon Reduced redundancy, reserved price plans, GoGrid Pre-Paid plans).

Cloud providers.

Real-time Mobile applications: There is an explosion of (primarily mobile based) communication apps. For example, WhatsApp, acquired by Facebook, has 450 million users [196]; Viber, acquired by Rakuten, has 200 million users [207]; and WeChat, a Chinese rival, has 270 million users [217]. For these apps, low latency is significant for real-time collaboration experience. For example, video conferencing has a limit of about 200 to 250 milliseconds delay for a conversation to appear natural [213]. These apps have similar requirements as game apps. They require a large number of servers to support millions of users and need optimization on latency, speed and throughput. It is worth mentioning that even for a generic web application, there are experiments with delaying a page in increments of 100 milliseconds, and the results show that even minimal delays would result in substantial and costly drops in revenue [213].

Big Data, IoT (Internet of Things) and eScience: IDC estimates that the amount of digital data generated annually will grow from 33 ZB in 2018 to 175 ZB by 2025 [2]. These data are resulting from an internet search, social media, business transactions, and content distribution. Similarly, scientific disciplines increasingly produce, process, and visualize data sets gathered from sensors [209]. If the prediction holds, then

the Square Kilometer Array (SKA) radio telescopes will transmit 400,000 petabytes (∼400 exabytes) per month or a whopping 155.7 terabytes per second [186]. Furthermore, European Space Agency (ESA) will launch several satellites in the next few years [60], which will collect data about the environment, such as air temperatures and soil conditions, and stream that data back in real time for analysis. Similarly, in the finance industry, the New York Stock Exchange creates one terabyte of market and reference data per day covering the use and exchange of financial instruments. On the other hand, Twitter feeds generate eight terabytes of data per day of social interactions [26]. Such "Data Explosions" has led to research issues such as: how to effectively and optimally manage and analyze a large amount of data. The issue is also known as the "Big Data" problem [88]. "Big data" is a field that treats ways to analyze, systematically extract information from, or otherwise deal with data sets that are too large or complex to be dealt with by traditional data-processing application software [25]. As both storing and analyzing the data requires a massive amount of storage capacity and processing power, companies and institutions may want to offload the complexity of managing hardware infrastructure to Cloud providers who are specialized in that, while eliminating the need to wait for facilities to be built.

Other: Apart from the scenarios as mentioned above, there are many more cases where our proposed solution would be useful. System administrators and developers may need a lot of simulated clients from all around the world for a website load testing before its official release. An automatic selection system would facilitate optimal auto deployments. Similarly, a bitcoin [21] (or some other similar cryptocurrencies [46]) miner may decide to invest in some additional resources in mining when the price of the currency is high, and stop the mining when the profit does not justify the expense anymore. In such a situation, a Cloud service selection system, which is price sensitive, would be beneficial as well.

4.2 Limitations of Existing Approaches

The network QoS (data transfer latency) varies mainly due to distance and traffic conditions. This variation should show some consistency, which depends on the location of the data centre and the location of input data. This raises a research question: **how to optimize the process of choosing the best compute and storage services, which are not only optimized in terms of price, availability, processing speed but also offers good QoS (e.g. network throughput and response delivery latency)?**

For research problems described in Section 1.1, many research [118] and commercial projects provide simple cost calculation or benchmarking and status monitoring, but none is capable of consolidating all aspects and providing a comprehensive ranking of infrastructure services. For instance, CloudHarmony [40] provides up-to-date benchmark results without considering cost, and Cloudorado [41] calculates the price of IaaS-level compute services based on static features (e.g., processor type, processor speed, I/O capacity, etc.) while ignoring dynamic QoS features (e.g., latency, throughput, load, utilization). Yuruware once had features for comparing Cloud ser-

vices, but it is no longer available. Swinburne University has a research project called Smart Cloud Broker Service [36]. From the screen-cast they released, we can tell that their benchmarking is done in real time, which means that users have to wait for the results to come back. We have considered this kind of situation but decided to collect the benchmarking result beforehand. This is because this way, no matter how many cloud providers users want to compare against, they can still get the result with a minimum (or no) waiting time. Another reason we choose to do it this way is that, at any particular point in time, the network benchmark result is not conclusive as performance fluctuates as time passes. Thus, we use an aggregated average, which is a more reliable overall indication. Table 4.3 shows a brief comparison of the CloudRecommender with other existing products we mentioned previously. We need to clarify that cloud management is not in the scope of this work.

To further distinguish ourselves from others, we offer the following two innovative features when ranking, selecting, and comparing various cloud services: 1) allow users to choose to include the QoS requirements during comparison; 2) when users want to take into account mixed qualitative (e.g. hosting region, operating system type) and quantitative criteria, we apply the Analytic Hierarchy Process to aggregate numerical measurements and non-numerical evaluation. Results are personalized according to each user's preferences because AHP takes users' perceived relative importance of criteria (pair-wise comparisons) as input.

Table 4.3: A Brief Comparison of The Cloud Recommender With Other Existing Solutions

Feature / Product	QoS Benchmark	Single Criteria Comparison	Aggregate Ranking Comparison	Cloud Management
Broker@Cloud	No evidence on progress of project			
Yuruware	No	No	No	Yes
CloudHarmony	Adjustable	No	No	No
Cloudorado	No	Yes	No	No
CloudBroker	Adjustable	Yes	No	No
CloudRecommender	Fixed	Yes	Yes	No

4.3 Proposed Approaches

The selection of Cloud services involves weighing the pros and cons of of multiple criteria, which is fundamentally a multi-criteria decision-making problem. We surveyed some well-known MCDM methods in Section 2.3.1. We chose AHP because it handles multiple criteria, i.e. capable of optimizing mixed qualitative and quantitative criteria. Moreover, AHP does not involve advanced mathematics and is intuitive to use. In comparison, SAW is stronger in single dimensional problems, but does not perform well on multidimensional problems; WPM has a problem of dealing with the equal weights of decision matrices; PROMETHEE requires generalized criteria to be defined, and such input is ambiguous for an inexperienced user to provide; the mechanism of VIKOR does not fit our problem. Furthermore, one problem encountered in most MCDM methods (e.g. TOPSIS) is the computation of the weight of criteria. This problem can be tackled in various ways like AHP, cross-entropy, fuzzy preference programming, etc. [53]. This also makes AHP stand out as a stand-alone solution.

4.3.1 Ranking with Analytic Hierarchy Process

We now present how we build a decision-making framework based on Analytic Hierarchy Process (AHP). It not only allows users to compare and select a Cloud service based on a single criterion (e.g. total cost, the max size limit for storage, and memory size for compute instance), but also supports a utility function that combines multiple selection criteria about storage, compute, and network services. In the rest of this section, we focus on the following topics:

1. **Problem Formulation.** We first provide formulations of the research problem by identifying the most important Cloud service selection criteria relevant to specific real-time QoS-driven applications, selection objectives, and Cloud service alternatives.

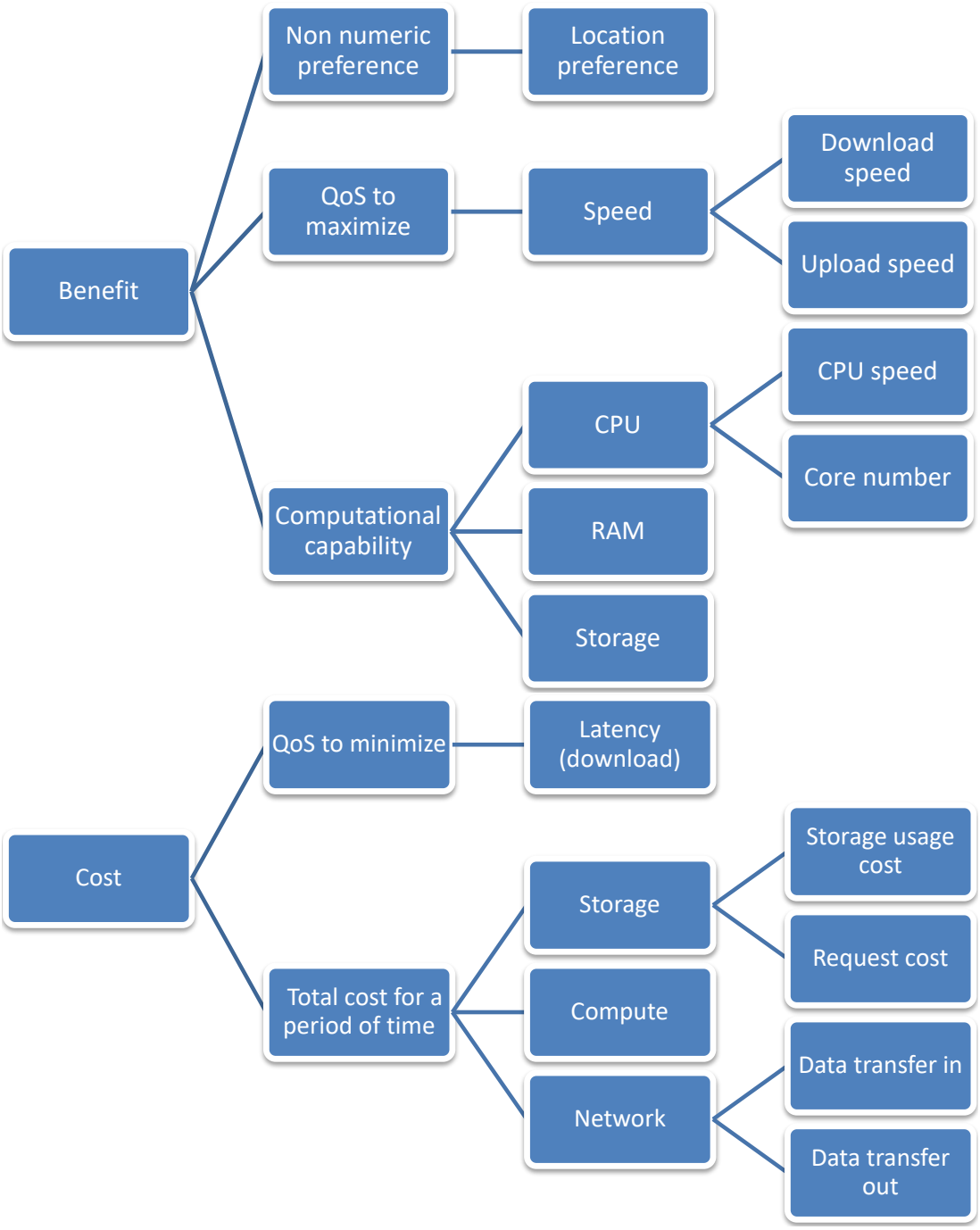
2. **Multi-criteria Optimization.** Then we discuss how AHP based decision (service selection) making techniques are designed. Examples are provided on how to handle multiple quantitative (i.e. numeric) as well as qualitative (descriptive, non-numeric, like location, CPU architecture: 32 or 64 bit, operating system) criteria. How pair-wise comparisons can be conducted to determine the relative importance of criteria is also described.

We also developed a decision-supporting tool with the proposed techniques. It can automate and map users' specified application requirements to specific Cloud service configurations. The details and evaluations (conducted in a real-world context) are presented in Chapter 6.

4.3.1.1 Cost Benefit Ratio

Generally, criteria can be divided into tow categories: benefit and cost. Benefit criteria are the good criteria which are meant to be maximized. Conversely, cost criteria

Figure 4.1: Criteria for Comparison



are the bad criteria to be minimized. The actual values to be collected and stored are at the leaf (i.e. Node/criterion with no children) of the tree. For example, as shown in Figure 4.1, under benefit criteria, numeric values are collected for "Download speed" "Upload Speed", "CPU Speed" and "Core number". "Speed" is the

parent of "Download speed" "Upload Speed", and there is no value stored for this node. Benefit criteria are criteria to be maximized, and cost criteria are criteria to be minimized. Table 4.5 explains the symbols used. Note that location preference is an example of qualitative criteria. Such preference can be converted into numerical scale by pairwise comparison explained in Section 4.3.1.3.

Each Cloud service has a number of criteria $\mathbb{C} = \{C_1, ..., C_k, ..., C_n\}$ where $\mathbb{C}^{\mathbb{B}} = \{C_1, ..., C_k\}$ and $\mathbb{C}^{\mathbb{C}} = \{C_{k+1}, ..., C_n\}$ are the set of benefit criteria and the set of cost criteria in $\mathbb{C}$. Let a_{ij} indicate the value of criteria C_j w.r.t. to Cloud service A_i. Suppose we have m Cloud services. Then we can normalize the criterion values a_{ij} as:

$$\widetilde{a_{ij}} = \frac{a_{ij}}{\sum_{z=1}^{m} a_{zj}} \tag{4.1}$$

Criteria normalization is carried out by dividing each value with the sum of all values for that criteria from different Cloud service providers. For each A_i, we calculate the cost/benefit ratio as shown in Equation (4.2), which is the sum of weighted cost criteria divided by the sum of weighted benefit criteria.

$$\frac{\sum_{j=k+1}^{n} (w_j \widetilde{a_{ij}})}{\sum_{j=1}^{k} (w_j \widetilde{a_{ij}})} \tag{4.2}$$

where w_j represents the weight for each criterion C_j, which measures users' perceived importance on the criterion. We will seek a smaller ratio as a better option. Note that the previous definition let there be k benefit criteria and n-k cost criteria.

Symbol	**Meaning**
r	Resource, e.g. GoGrid XX - Large Instance, S3 Storage Serive, EC2 instance.
c	Cloud Provider, e.g. Amazon,Rackspace, GoGrid.
l	A datacenter location, e.g. Sydney, Tokyo.
P	Unit price for a resource.
T	Period of time the resource is used.
u	Usage of a resource behave like a decision variable.

Table 4.4: Usage Expense Estimation Symbols

Note that denominator for Equations (4.1) and (4.2) can not be zero under normal circumstances. Since all criteria are non negative, cost is the only criteria can possibly be zero, but it is in the nominator. All benefit criteria are positive, e.g. it doesn't not make sense to purchase zero resource.

4.3.1.2 Usage Expense Estimation

The corresponding expense for resource usage is one of those important cost criteria. Due to the high complexity of calculating the expenses related to different resource types, providers and locations, we dedicate this section to explain how to estimate usage expenses.

Let "u" be the intended usage of a resource from a data center location of a Cloud provider. More specifically, $u_{r,c,l} \in \{0, 1, \ldots, n\}$ means the usage of the compute resource r from provider c at location l. For example, when the user have a rough estimate of how much resources they might need, we can use $u_{storage,any,any} = 50GB$ to represent the user's need to store 50 GB of data in the Cloud, regardless of providers and locations.

To calculate the expense of $u_{r,c,l}$, we multiply its usage with the corresponding unit price ($P_{r,c,l}$) as:

$$u_{r,c,l} P_{r,c,l} \tag{4.3}$$

Then we can calculate the total price per unit time for desired resource(s) (assume constant resource usage pattern throughout the time) as in Equation (4.4). We assume users will choose the duration/time period ($T_{r,c,l}$) they want to estimate price for, e.g. 1 hour, 30 days.

$$u_{r,c,l} P_{r,c,l} T_{r,c,l} \tag{4.4}$$

Symbol	Meaning	Type of Criteria
$C_{downlink}$	Downlink speed.	Benefit
$C_{latency}$	Downlink latency.	Cost
C_{memory}	Memory size.	Benefit
$C_{storage}$	Storage size.	Benefit
C_{uplink}	Uplink speed.	Benefit
$C_{expense}$	Usage expense.	Cost
C_{CPU}	CPU speed.	Benefit

Table 4.5: Symbols for Common Criteria

The symbols used for usage expense estimation are summarized in Table 4.4. Expense calculation is not new, but existing calculators most only include service from a single provider. Here we build upon the generic CoCoOn model and collected data from multiple providers. So calculation can be applied across providers.

Suppose we have seven criteria $\{C_{expense}, C_{latency}, C_{uplink}, C_{downlink}, C_{memory}, C_{storage}, C_{CPU}\}$ as shown in Table 4.5, five of them are benefit criteria and two are cost criteria. After substitute the a_{ij} with our actual criteria, assuming we have calculated the preference weights, the cost-benefit ratio formula can be expanded to:

$$\frac{w_{expense}\widetilde{a_{expense}} + w_{latency}\widetilde{a_{latency}}}{w_{uplink}\widetilde{a_{uplink}} + w_{downlink}\widetilde{a_{downlink}} + w_{memory}\widetilde{a_{memory}} + w_{storage}\widetilde{a_{storage}} + w_{CPU}\widetilde{a_{CPU}}} \tag{4.5}$$

where $a_{expense}$ is calculated as discussed previously:

$$a_{expense} = \sum U_{c,l,r} P_{c,l,r} T_{c,l,r} \tag{4.6}$$

When calculating the expense we need to sum over all resources. If user only select one type of resource, all resources is just one type. $a_{latency}$ is the average of all latency QoS stats collected over time on the resource r from provider c and location l, a_{uplink} is the average uplink speed, $a_{downlink}$ is the average downlink speed, a_{memory} is the total memory size, i.e. if an alternative option consists of 2 VM of 8GB each, the total memory should be 16 GB, $a_{storage}$ is the total storage size, and a_{CPU} is the CPU speed.

Table 4.6: Absolute Value and Corresponding Descriptive Scale Representing Relative Importance

Scale	**Value**	**Reciprocals**∗
equal	1	1
moderate	3	1/3
strong	5	1/5
very strong	7	1/7
extreme	9	1/9

∗**If activity i has one of the above nonzero numbers assigned to it when compared with activity j, then j has the reciprocal value when compared with i.**

In our decision-making framework, we consider the following QoS statistics: download latency, download speed and upload speed. Those characteristics are important for end-users' experience and satisfaction. It is possible to have options that have small price differences, or when having a high-quality service is more important than saving money. Note that the network QoS statistics of Compute and Storage ser-

vices are both collected then separately stored, since a user may be only interested in one of the services. For example, transferring files from (and to) a compute instance storage is different from downloading or uploading files from/to dedicated network storage service (like AWS S3 [16]). If a user selects both, we use the average of data downlink speed from virtual machines and storage services as the average download speed.

4.3.1.3 Weight Computed by Pairwise Comparison

The weights of criteria are calculated based on AHP's pairwise comparison method. We choose the commonly used scale [69, 81] shown in Table 4.6. If a user does not specify any preference, by default all weights are equal.

To illustrate, if we have n (=4) criteria, we can build the PWC matrix as:

$$
\begin{array}{c}
M \\
\begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ C_4 \end{array}
\begin{array}{c}
\begin{array}{cccc} C_1 & C_2 & C_3 & C_4 \end{array} \\
\begin{pmatrix} 1 & a_{12} & a_{13} & a_{14} \\ a_{12}^{-1} & 1 & a_{23} & a_{24} \\ a_{13}^{-1} & a_{23}^{-1} & 1 & a_{34} \\ a_{14}^{-1} & a_{24}^{-1} & a_{34}^{-1} & 1 \end{pmatrix}
\end{array} \\
\text{column sum} \downarrow \\
\begin{bmatrix} S_{c1} & S_{c2} & S_{c3} & S_{c4} \end{bmatrix}
\end{array}
\tag{4.7}
$$

In Equation (4.7), a_{ij} is the preference of criterion C_i compared to C_j for $1 \leq i, j \leq n$. When $i = j$, we always get 1 as it is compared to itself. And $\forall i, j \quad a_{ji} = a_{ij}^{-1}$. A numeric example is shown in Table 4.7. Then we normalize M and calculate

Table 4.7: Example User Preference

	Uplink Speed	Downlink Speed	Ram	Storage
Uplink Speed	1	1/3	1/5	1/5
Downlink Speed		1	3	5
Ram			1	3
Storage				1

normalized principal eigenvector x_1.

$$\begin{matrix} \widetilde{M} \\ \begin{bmatrix} \frac{1}{S_{c1}} & \frac{a_{12}}{S_{c2}} & \frac{a_{13}}{S_{c3}} & \frac{a_{14}}{S_{c4}} \\ \frac{a_{12}^{-1}}{S_{c1}} & \frac{1}{S_{c2}} & \frac{a_{23}}{S_{c3}} & \frac{a_{24}}{S_{c4}} \\ \frac{a_{13}^{-1}}{S_{c1}} & \frac{a_{23}^{-1}}{S_{c2}} & \frac{1}{S_{c3}} & \frac{a_{34}}{S_{c4}} \\ \frac{a_{14}^{-1}}{S_{c1}} & \frac{a_{24}^{-1}}{S_{c2}} & \frac{a_{34}^{-1}}{S_{c3}} & \frac{1}{S_{c4}} \end{bmatrix} \end{matrix} \xrightarrow{\frac{\sum row}{n}} \begin{matrix} x_1 \\ \begin{bmatrix} \frac{\frac{1}{S_{c1}} + \frac{a_{12}}{S_{c2}} + \frac{a_{13}}{S_{c3}} + \frac{a_{14}}{S_{c4}}}{n} \\ \frac{\frac{a_{12}^{-1}}{S_{c1}} + \frac{1}{S_{c2}} + \frac{a_{23}}{S_{c3}} + \frac{a_{24}}{S_{c4}}}{n} \\ \frac{\frac{a_{13}^{-1}}{S_{c1}} + \frac{a_{23}^{-1}}{S_{c2}} + \frac{1}{S_{c3}} + \frac{a_{34}}{S_{c4}}}{n} \\ \frac{\frac{a_{14}^{-1}}{S_{c1}} + \frac{a_{24}^{-1}}{S_{c2}} + \frac{a_{34}^{-1}}{S_{c3}} + \frac{1}{S_{c4}}}{n} \end{bmatrix} \end{matrix} \tag{4.8}$$

If we use the same example from Table 4.7. It will produce the preference matrix M_1:

$$\begin{matrix} M_1 \\ \begin{bmatrix} 1 & 1/3 & 1/5 & 1/5 \\ 3 & 1 & 3 & 5 \\ 5 & 1/3 & 1 & 3 \\ 5 & 1/5 & 1/3 & 1 \end{bmatrix} \end{matrix} \tag{4.9}$$

$$\text{column sum} \downarrow$$

$$\begin{bmatrix} 14 & 1\frac{13}{15} & 4\frac{8}{15} & 9\frac{1}{5} \end{bmatrix}$$

Then we can find $\widetilde{M_1}$, which is the normalized M_1, by dividing each value in M_1 with the sum of its column. Then we calculate normalized principal eigenvector, round result to 4 decimal places to get v_1:

$$\begin{matrix} \widetilde{M_1} \\ \begin{bmatrix} \frac{1}{14} & \frac{5}{28} & \frac{3}{68} & \frac{1}{46} \\ \frac{3}{14} & \frac{15}{28} & \frac{45}{68} & \frac{25}{46} \\ \frac{5}{14} & \frac{5}{28} & \frac{15}{68} & \frac{15}{46} \\ \frac{5}{14} & \frac{3}{28} & \frac{5}{68} & \frac{5}{46} \end{bmatrix} \end{matrix} \xrightarrow{\frac{\text{row sum}}{4}} \begin{bmatrix} \frac{247}{782} * \frac{1}{4} \\ \frac{1529}{782} * \frac{1}{4} \\ \frac{5925}{5474} * \frac{1}{4} \\ \frac{3539}{5474} * \frac{1}{4} \end{bmatrix} \approx \begin{matrix} v_1 \\ \begin{bmatrix} 0.0790 \\ 0.4888 \\ 0.2706 \\ 0.1616 \end{bmatrix} \end{matrix} \tag{4.10}$$

Finally, we square normalized matrix $\widetilde{M}$ and calculate the next iteration of eigenvector, e.g. x_2 is calculated from $\widetilde{M}^2$, until the difference $x_{k+1} - x_k$ is neglectable. This process is supposed to be repeated until no big enough difference can be observed. In our case, we noticed that the improvement is small after the first squaring, so we relaxed the rule only to do one matrix squaring. If high precision is needed, one can define the threshold of $x_{k+1} - x_k$. e.g. 0.001 means stop when difference is less than 0.001.

Continued with the previous example, v_1 is calculate from $\widetilde{M_1}$ before matrix squaring. If we square the matrix we get:

$$\widetilde{M_1} \times \widetilde{M_1} = \begin{bmatrix} 0.06688762 & 0.11862571 & 0.13265383 & 0.11535162 \\ 0.56054596 & 0.50165718 & 0.54990973 & 0.57067402 \\ 0.25901665 & 0.23376742 & 0.20656472 & 0.21218885 \\ 0.11354977 & 0.14594968 & 0.11087171 & 0.10178552 \end{bmatrix} \tag{4.11}$$

If we keep doing this matrix squaring, the results are shown in Table 4.8. $\mathrm{E}(M)$ denotes the eigenvector of matrix M, so the eigenvector calculated from (4.11) is $\mathrm{E}(\widetilde{M_1^2})$, and so on. This example only shows the benefit criteria, as cost criteria need to be calculated separately. Once all the weights are computed, we can calculate the

Table 4.8: Eigenvector Calculation: Matrix Squaring

Criteria	$\mathrm{E}(\widetilde{M_1})$	$\mathrm{E}(\widetilde{M_1^2})$	$\mathrm{E}(\widetilde{M_1^4})$	$\mathrm{E}(\widetilde{M_1^8})$
Uplink Speed	0.07896419	0.1083797	0.11582867	0.11542847
Downlink Speed	0.48881074	0.54569672	0.52718223	0.52830939
Ram	0.27059737	0.22788441	0.22775774	0.22771419
Storage	0.16162769	0.11803917	0.12923137	0.12854795
Change		0.0294155	0.00744897	-4.00196647e-04
		0.05688598	-0.0185145	1.12716376e-03
		-0.04271296	-0.00012667	-4.35480523e-05
		-0.04358852	0.0111922	-6.83419065e-04

cost-benefit ratio with Equation (4.5).

Remark. *We published the work of applying AHP on Cloud service recommendation back in 2015. Later, we learnt that many researchers criticised its normalisation method, which leads to rank reversal. Rank reversal is the irregularity that may appear in a ranking when there are two alternatives with similar values, or when adding an alternative inferior to non-optimal alternatives in the ranking. Previous studies [97, 112, 18, 56, 201] suggested to use Multiplicative Additive Weightings (MAW) instead of Simple Additive Weightings (SAW).*

4.3.2 Algorithm

Our approach is described in Algorithm 1. Symbols are explained in Table 4.9. which mostly are the relational algebra and set operations symbols, please pay attention to operation G as it has multiple inputs represented by superscript and subscripts.

Algorithm 1 only depicts one everyday use case, while other scenarios exist, they can be solved with this algorithm with a small modification. We will explain these situations in the following paragraphs.

As shown in Algorithm 1, a user can provide us with the following parameters as input:

1. ℓ is the set of locations that a user wants to consider, and by default, we consider all locations.

2. ρ is the set of Cloud service providers that a user wants to consider, and by default, we consider all providers.

3. $\mathfrak{M}_{\text{min}}$ is the minimal memory requirements for the VMs, and by default 0 denotes no memory requirements.

4. $price_{\text{max}}$ is the maximum budget a user is willing to spend, where 0 indicates they are only interested in free services, and -1 represents infinity which means there is no budget constraint.

5. U is a tuple representing the estimated usages provided by user, which contains the following: $(u_{storage}, u_{data_{out}}, u_{VM})$. $u_{storage}$ represents the amount of storage (GB) that will be used. $u_{data_{out}}$ is the amount of outward data transfer in GB from Cloud provider to end devices/users. Since normally data ingress data transfer to the Cloud is free, we are more concerned with the egress data transfer usage. u_{VM} is the number of instances needed. All the previously mentioned usage estimations are monthly based by default, but other lengths can be used, such as daily or hourly. As long as all resource is calculated based on the same standard, there should be no effect on the final comparison and ordering.

6. W is a tuple representing the preference/weight given to each component by the user, which consists of the following: $(w_{cpu}, w_{memory}, w_{storage}, w_{network}, w_{download}, w_{upload}, w_{latency}, w_{expense})$, details are explained in Sections 4.3.1.1 and 4.3.1.3.

Table 4.9: Symbols Used In Algorithm: Relational Algebra and Set Operations.

Symbol	Meaning
G	Aggregation operation over a schema, like a **group by** clause in SQL. It follows the format: ${}_{a_g}G_{agg\_op(attri)}(r)$ where a_g is the grouping attribute. $agg\_op(attri)$ is the aggregation operation over attribute (attri). There are five aggregate functions that are included with most relational database systems. These operations are Sum, Count, Average, Maximum and Minimum. **r** is an arbitrary relation. See relationa algebra wiki page [181] for more details.
σ	Selection, see wiki page [181].
$\bowtie$	Natural join: depends on the condition can be either θ-join or equijoin. For example, $\bowtie\ (Provider, Location)$ means equijoin where the condition is join only under the same provider and location
$\cup$	Set union operation.
$\mapsto$	Ordered pair, here we use it to denote a new record being formed. The reason we used it is because $storageCostByQuota$ maps storage options to a set of costs, i.e. for the same kind of storage option it maps to multiple storage tiers, with different unit prices; thus $storageCostByQuota : \Phi_{storage} \to \mathcal{P}(\mathbb{R})$ where $\mathcal{P}(\mathbb{R})$ means the power set of real numbers, for example for a value x, we can have $storageCostByQuota(x) = \{1, 2\}$, and the total cost for x is 1+2=3.

First, we filter on the static characteristics on Line 1, such as provider, location, memory, and CPU speed. $\Re_{compute}$ is the table containing all data collected on Cloud Compute resources. Similarly, $\Re_{storage}$ is the table containing data for storage services and $\Re_{network}$ is the table for network services. Line 2 shows the filtering step on the storage options, and Line 3 filters on the network options.

Firstly we identified the options satisfying user requirements, then calculate the total price according to different models. For example, various pricing models include free, flat-rate, two-part tariffs (like the AWS reserved instance), block-declining (S3 storage), and bidding (AWS spot instance). Our model can handle all of them except for the bidding price model. One provider often has multiple offers within the same type of services, for example, different kinds of instances for a compute service and different storage options. We combine them to get a combinatorial num-

Algorithm 1: Rank Cloud services based on requirements

input : Requirements such as locations (ℓ), providers (ρ), minimal memory ($\mathfrak{M}_{\text{min}}$), maximum budget ($price_{\text{max}}$), Estimated resources usages (U), weights (w))

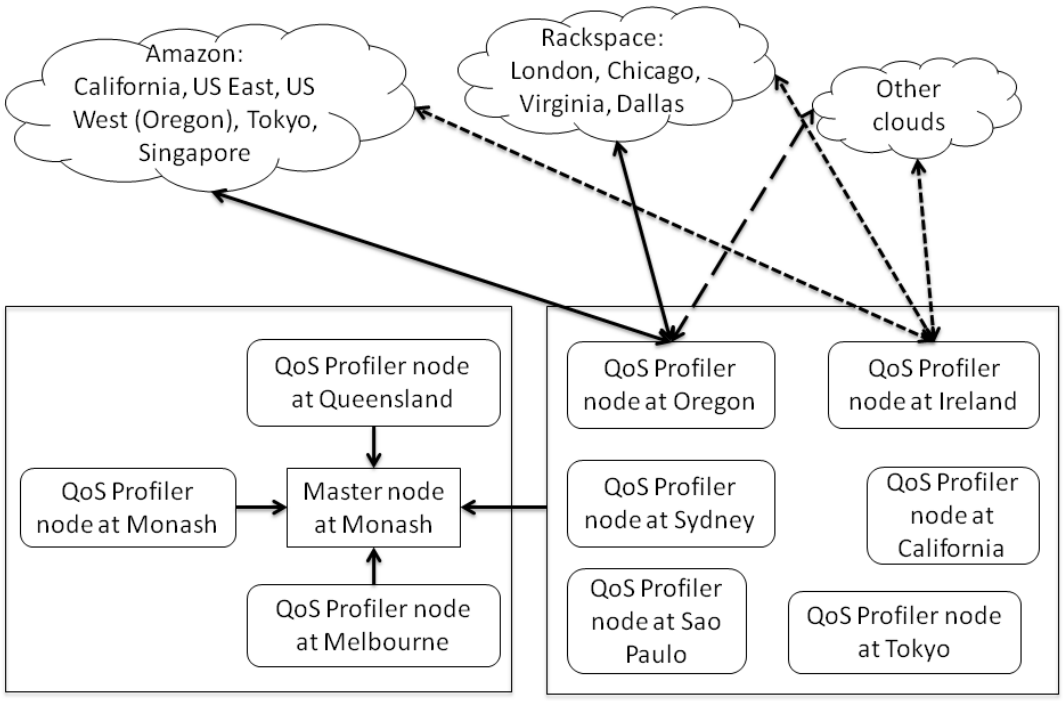
output: Cloud service offerings recommendation, sorted by rank from low (better) to high.

1 $\Phi_{compute} \leftarrow \sigma_{provider \in \rho \wedge location \in \ell \wedge memory \geq \mathfrak{M}_{\text{min}}} \left(\Re_{compute}\right)$;
2 $\Phi_{storage} \leftarrow \sigma_{provider \in \rho \wedge location \in \ell \wedge quota_{low} < u_{storage}} \left(\Re_{storage}\right)$;
3 $\Phi_{network} \leftarrow \sigma_{provider \in \rho \ \wedge \ location \in \ell \ \wedge \ quota_{low} < u_{data_{out}}} \left(\Re_{network}\right)$;
4 $storageCostByQuota \leftarrow \emptyset$;
5 **foreach** $x_s \in \Phi_{storage}$ **do**
6 **if** $u_{storage} > quota_{max}(x_s)$ **then**
7 $storageCostByQuota \cup \{x_s \mapsto (quota_{max}(x_s) - quota_{min}(x_s)) \cdot unitPrice(x_s)\}$;
8 **else**
9 **if** $u_{storage} > quota_{min}(x_s)$ **then**
10 $storageCostByQuota \cup \{x_s \mapsto (u_{storage} - quota_{min}(x_s)) \cdot unitPrice(x_s)\}$;
11 **end**
12 **end**
13 **end**
14 $storageCost \leftarrow {}_{service\_name \ \& \ provider}G_{sum(storage\_cost)}(storageCostByQuota)$;
15 $\varphi_{compute} \leftarrow \Phi_{compute} \bowtie_{provider,location,serviceName} AvgQoS$;
16 $\varphi_{storage} := storageCost \bowtie_{provider,location,serviceName} AvgQoS$;
17 $\varphi \leftarrow \varphi_{compute} \bowtie_{provider,\ location} \varphi_{storage} \bowtie_{provider,\ location} \Phi_{network}$;
18 $totalCost \leftarrow \emptyset$;
19 **foreach** $x \in \varphi$ **do**
20 $totalCost(x) \leftarrow \sum (u_r P_r)$
21 **end**
22 $rank \leftarrow \emptyset$;
23 **foreach** $x \in \varphi$ **do**
24 $rank(x) =$ Equation (4.5);
25 **end**
26 **return** sortOnRankDescending(rank)

ber of choices. We do that for all providers, and then calculate the summed cost and the cost-benefit ratio for each combined option. Not all users need all three types of resources. If they do not need a service, its usage will be zero. However, a network service is always needed, thus cannot be zero. If a user wants to avoid costly data transfer, he may want to keep data within a single region, as inter region data transfer is often free. Others may need to use multiple providers to achieve disaster resilience.

In the for loop starting from Line 5, we go through each price option of storage

Figure 4.2: QoS Monitoring Service Network Topology



services. Storage services are charged with block declined model. For example, AWS S3 price in US East (N. Virginia) for the first 50 TB per month is 0.023 USD per GB. Next, 450 TB per Month is 0.022 USD per GB. So for the usage of 51 TB, the total cost is calculated as: $50 * 1024 * 0.023 + 1 * 1024 * 0.022 \approx 1200.13$ We first calculate the storage price for each tier. Assume units are normalized. Line 14 sums up costs in different tiers for each storage service.

Next on Line 15, we link the results $\Phi_{compute}$ with QoS statistics. "AvgQoS" is the table containing the QoS data collected. Similarly, on Line 16, we link storage data with QoS.

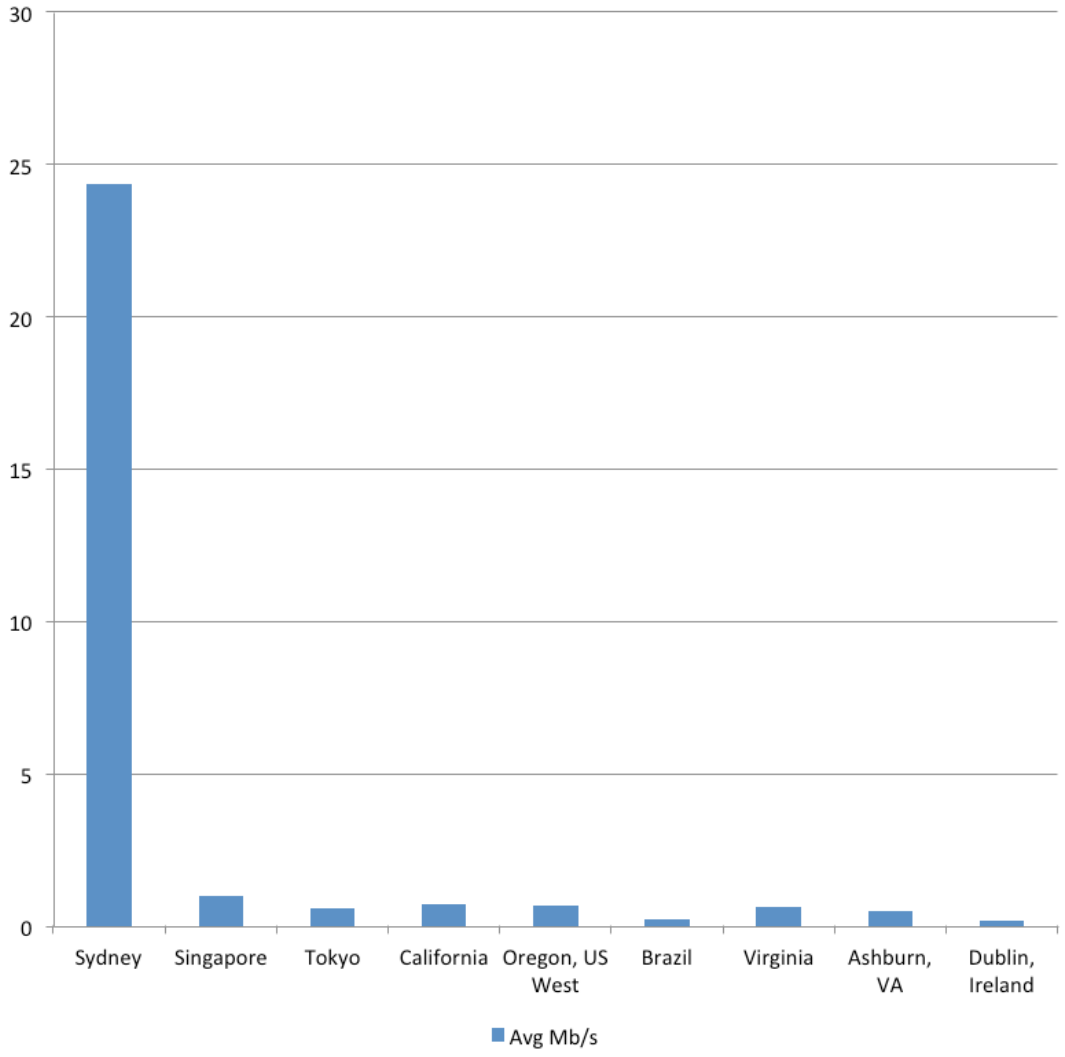
Then, we match appropriate Compute, Storage and Network options on Line 17. After this, in the for loop on Line 19 we calculate the total cost by aggregating cost for all resources.

Before calculating the final rank, we first find the unit values for normalization. Then from Line 23 we calculate the rank of each option according to Equation (4.5). The average download speed is calculated as: $\frac{1}{2}\left(D_{compute} + D_{storage}\right)$ where $D_{compute}$ is the average download speed from the compute instances, and $D_{storage}$ is the average download speed from storage service in the same combined option.

At last, Line 26 sorts the results in accenting order of rank before they are returned.

4.3.3 QoS Profiling

Figure 4.3: Average download speed from Amazon data centers to Melbourne



We implemented a QoS Profiler that helps in collecting network QoS values from different points on the Internet (modelling big data source locations) to the Cloud data centres. We collected the statistics using the "speedtest" service provided by CloudHarmony.

Klein et al. [110] proposed a highly theoretical model based on Euclidean distance for estimating latency, which we believe their theoretical perfect distance assumption cannot be practically accurate. However, we can use this model to estimate latency when QoS data is not available for a new client location.

Our QoS profiling system consists of multiple agents at geographically dispersed locations to collect and process data, shown in Fig 4.2. Initially, QoS data was col-

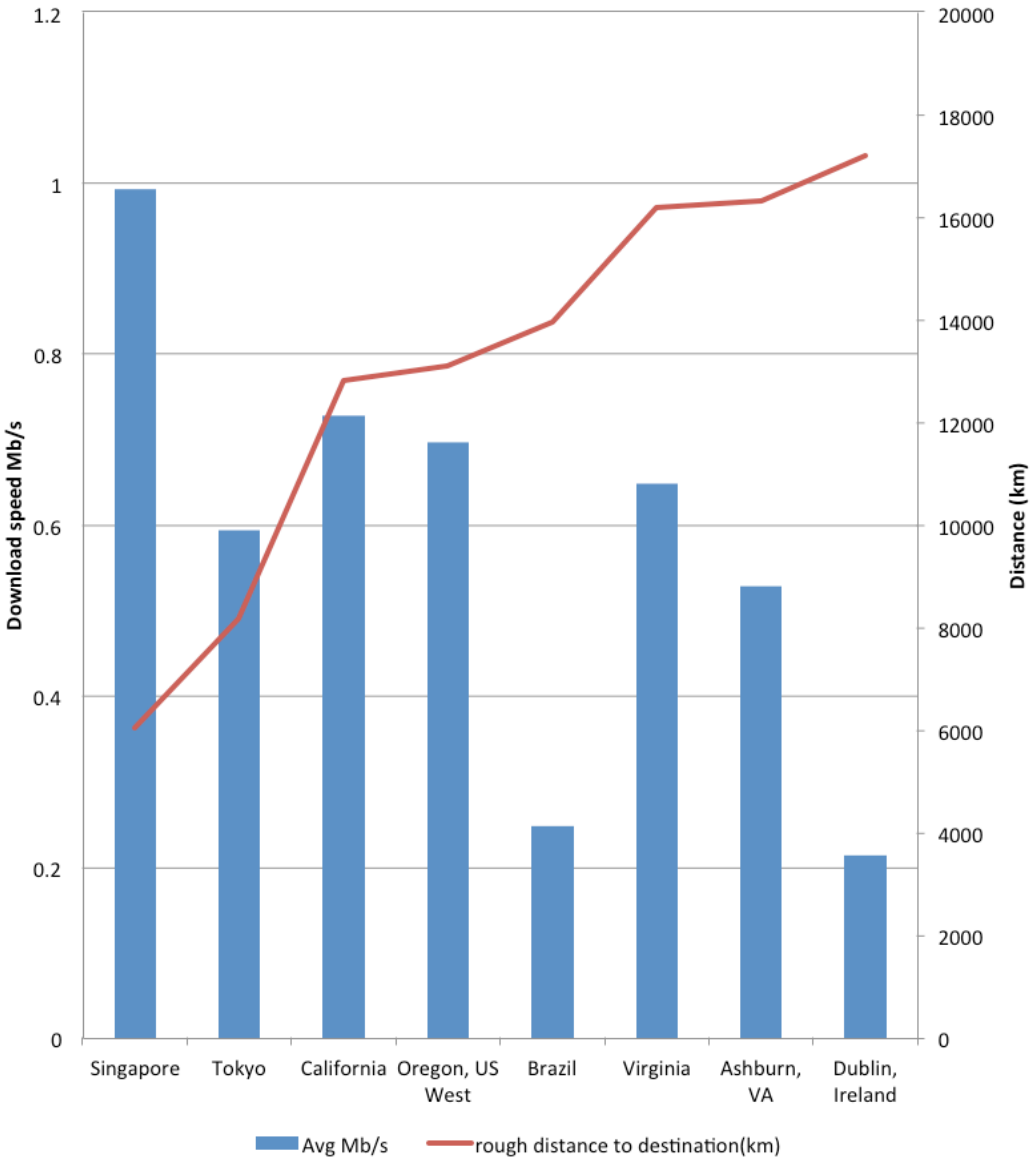


Figure 4.4: Download Speed Against Distance

lected on slave nodes every 2 hours by running the "speedtest" service of CloudHarmony. A single run takes more than an hour to finish, and hence we collect it once every two hours. We have used two Clouds, namely: Nectar Research Cloud and Amazon Web Service. Since Nectar Cloud is free for researchers, we decided to put the master node in Nectar. There is a limit of quota in Nectar and Amazon have more excellent geographical coverage in terms of data-centre locations, so we used an additional spot instance from Amazon as slave data crawlers. A QoS monitoring node records download speed, latency and upload speed from different locations to each data-centre in various Clouds. If we look at individual slave nodes, we can see every

node records the QoS statistics to various Clouds from each location. Bashed scripts are written to export data from each node. Master node pulls data from its children nodes, and access keys are required for this operation. Then the CSV formatted data is imported to the master database, where appropriated merge operation is performed. Data was collected over a period of two weeks. For more implementation details, please refer to Chapter 6.

Later by analyzing the data, we conclude that such high frequency is not necessary, because the average QoS from a particular location to a particular data centre mostly has very fluctuations. That means the average would be pretty stable. Note that differences between data-centres and various locations are still vast as expected, which is shown in Figure 4.3. In the future, we may allow a combination of real-time and off-line values to be used if necessary.

Figure 4.3 shows that geographically close data-centres have (as high as 25 times) better network performance. Hence, this validates the fact that location is one of the important criteria which should be considered during the selection process. Our measurements also indicate that distance is not the only factor that affects network performance, as shown in Figure 4.4. Data-centres are ordered from closest to furthest, and from left to right, Tokyo and Brazil perform worse than expected. Hence, we consider the need for active probing and profiling of network QoS from a user's endpoint to the Cloud data-centres. By doing so, we get a clear picture of the data centre's network QoS from the users' device that may be deployed across topologically distributed network locations. Note that we have left out Sydney on purpose.

Figure 4.3 shows the exponential increase in speed between Sydney and Melbourne compare to overseas locations, while Figure 4.4 shows the linear relationship between downloading speed and distance among overseas locations. We are aware that while it is generally true that the geographical distance between any pair of servers (or users) on the Internet affects the route trip time (RTT), the bandwidth between them is not necessarily determined by the distance, many other aspects can affect the user end QoS, like the last-mile home-connecting technology, and local Internet traffic condition. Our measurements only provide a suggestive base for further optimisation; user's experience may vary due to unforeseen conditions of Internet traffic.

4.4 Experiments

4.4.1 Setup

We run our system and proposed algorithmic techniques across a range of hardware systems, see Table 4.10.

To summarise, Environment 1 is the local machine used during the development of the program, which is capable of running the database and other system modules. Environment 2 is the server from the National eResearch Collaboration Tools and Resources (NeCTAR) Cloud [90] where our system can be deployed as a service which is easily accessible over the Internet. It is a virtualised environment, so the CPU

Table 4.10: Experiment Environments

Environment	Description	Processor Speed	Memory	Processor Name	Role
1	MacBook Air Physical machine	1.4 GHz	2 GB	Intel Core 2 Duo	Master
2	Ubuntu 12.04.3 LTS instance ina virtualized environment	2.4 GHz (1vCPU)	4 GB	AMD Opteron (TM) Processor 6234	Master / Profiler
3	Standard Small (m1.small)Linux / UNIX EC2 Spot Instance	1.79 GHz (1ECU / vCPU)	1.7 GB	Intel(R) Xeon(R)CPU E5-2650	Profiler
4	Compute Optimized(c3.8xlarge) Linux/UNIX EC2Spot Instance	2.8 GHz (32 vCPU 1081 ECU)	60 GB	Intel(R) Xeon(R)CPU E5-2680v2	Performance Testing

speed labelled may not accurately reflect the actual allocation. NeCTAR's infrastructures are located at eight different organisations (node sites) around Australia. It operates as one Cloud system under the OpenStack framework. It has different UIs and APIs compared to AWS. Being a collaborative research Cloud, it is only open to affiliated members (i.e. Australian researchers, and students from participating university). Although the access is free, there is a limitation of 2 instances per member and a cap on the total resource usage. Environment 3 is the spot instance type (from Amazon) we used to collect QoS statistics from additional locations. Environment 4 is the compute optimised spot instance type we used to test program performance under a powerful CPU or vertical scalability in short.

4.4.2 Case Study

4.4.2.1 Input Parameters

Table 4.11 shows the primary configurable parameters of our algorithm. Everyone's requirements regarding the compulsory parameters usually vary. So we choose a range of values to mimic different selection scenarios. In future work, we may conduct a user survey to understand the most concerned factors for different types of users, for example, we can expose all possible constrainable parameters via the API,

Table 4.11: Input Parameters

Compulsory	**Example Value**
Storage(GB/30 Days)	20
Outbound Data Transfer(GB/30 Days)	50
Min RAM(GB)	4
Optional	**Default Value**
Provider Brand	Consider All
Display Currency	AUD
Number of Hours to run (per Month)	720
Number of Instance needed (per Month)	1
Inbound Data Transfer(GB/30 Days)	1
Weight of Compute Cost(percentile)	35%
Weight of Storage Cost(percentile)	25%
Weight of Network Cost(percentile)	35%
Weight of Latency(percentile)	5%
Weight of Download Speed(percentile)	70%
Weight of Upload Speed(percentile)	30%
Max RAM(GB)	20

but it may not be necessary. It can overwhelm the users who only use the visual interface, especially for users with a less technical background. Default value column shows what we use when users do not specify it.

4.4.2.2 Results

Figure 4.5 shows the top 5% of the result we get from the inputs in Table 4.11. It is in ascending order of ratio (cost over benefit) as indicated by the dotted (blue) line

because lower cost over higher benefit gives us a smaller ratio, which represents a better choice. If we look at ranking by considering only the cost, as illustrated by the solid (red) line, the Go-Grid offers to dominate over Windows offerings. If ordering results in ascending price order (means network QoS constraints are not considered), shown in Figure 4.6, Azure disappears from the top 10% of choices. Similarly, we can see that although the price change is small in solutions, their overall rankings are significantly different (dotted blue line). What this means to users is that while we can save money by ignoring network QoS but then they should be ready for degraded network performance. Note that, although we tried out best in using real-world data, sometimes Cloud providers vary their prices as frequent as weekly. However, in future work, we intend to implement a price crawler service that will automatically parse the provider's web pages and update our system's database.

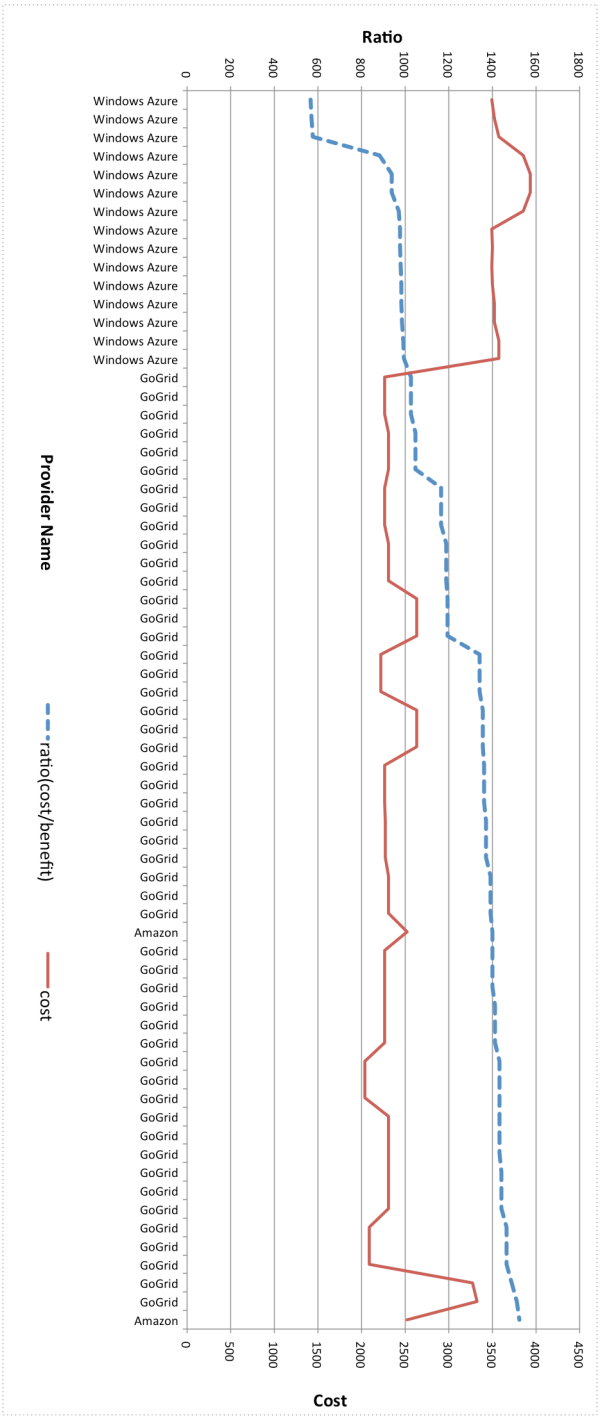


Figure 4.5: Results in ascending order by (cost / benefit) ratio

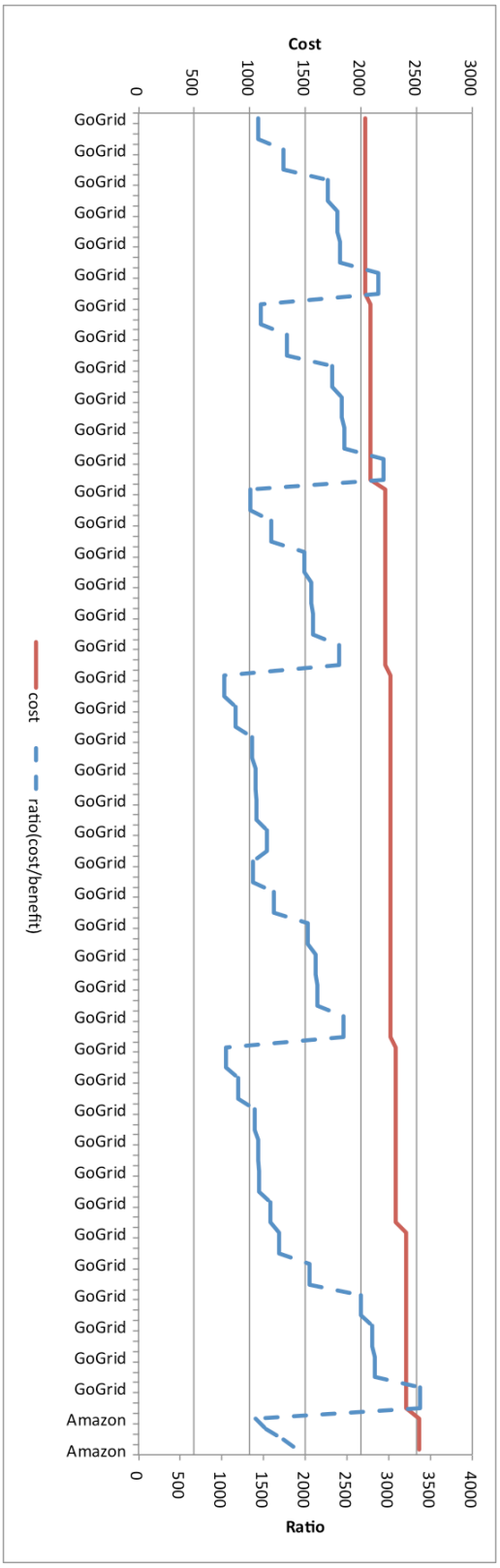


Figure 4.6: Results in ascending order by cost

Table 4.12: Average Runtime.

Test Number	Storage (GB/30 Days)	Outbound Data Transfer (GB/30 Days)	Min RAM (GB)	Row(s)	Enviroment 1	Enviroment 2	Enviroment 4
1	20	10	0	3808	12.04	11.07	10.96
2	40	15	0	3808	11.913	11.59	7.81
3	10	2	0	3808	11.169	10.76	7.05
4	20	2	0	3808	11.744	11.15	7.57
5	200	200	0	3808	11.894	11.72	7.49
6	200	200	0	3808	11.912	10.85	6.76
7	200	200	16	552	9.15	7.7	4.97
8	200	200	8	1524	9.644	9.69	5.53
9	200	200	4	2095	10.25	8.72	5.58
10	20	20	0	3808	12.06	11.51	7.03
Average					11.1776	10.476	7.075

4.4.2.3 Performance

The average run time for a query is about 11 seconds, as shown in Table 4.12. With cache being turned on in MySQL, we get up to 9% improvement on the same query. As the constraints become stricter than previous, the solution space increasingly reduces as a result of the processing time decreases (to as low as 4.97 seconds).

We observe that the performance increases when we move from Environment 1 to Environment 2. Environment 4 has the shortest processing time as a result of increased hardware capacity, hence the idea of "scale up". There is a limit to the amount of processing power one core can have, but our solution is single threaded at the moment. There is still room for improvement by utilizing all cores. In the future, we will explore the option of configuring MySQL/InnoDB to use multithreading (Default is 4, and the maximum is 64 since MySQL 5.1.38). Then we will decide

whether we need to "scale out".

4.4.3 Computational Complexity

We define the upper bound computational complexity of our optimization approach as shown in (4.12).

$$O\left(|R| \times |C| \times |L| + \left(\binom{k}{2} + \binom{n-k}{2}\right)\right) \tag{4.12}$$

In cost estimation, we have to calculate prices for $|R|$ resources, $|C|$ providers, and $|L|$ geographical locations. The computational complexity may increase with a more complex utilization function. For example, if the estimated usage has periodic bursts, more during the day (10 servers) and less at night (2 servers). The utilization function can follow a sine curve.

Since we have n criteria, and from 1 to k is the benefit criteria, and from k+1 to n is the cost criteria. We need to do $\binom{k}{2} + \binom{n-k}{2}$ pair-wise comparisons to compute the preference weight. In our current model, we have k = 5 and n = 7, hence $\binom{k}{2} + \binom{n-k}{2} = \binom{5}{2} + \binom{2}{2} = \frac{5!}{2!(5-2)!} + 1 = 11$.

4.5 Summary

In this chapter, we proposed an AHP based QoS-aware technique for Cloud infrastructure services selection. Our approach takes into account real-time, variable network QoS constraints, as it is necessary to guarantee the performance of applications. It not only allows users to compare and select a Cloud service based on a single criterion (e.g. total cost, the max size limit for storage, and memory size for computing instance) but also supports a utility function that combines multiple selection criteria about storage, compute, and network services.

AHP is selected because it allows both actual measurements and subjective opinions to be inputs. If there is a preference on the brand, technology or location of a service, it can be turned into weights by pairwise comparison.

Chapter 5

Resource Usage Estimation

In the previous chapters, we have proposed techniques for selecting Infrastructure as a Service (IaaS) Cloud offers, which allow users to define multiple design-time and run-time requirements. These requirements are then matched against our knowledge base to compute possible best fit combinations of Cloud services at the IaaS layer. We have also investigated QoS-aware Cloud service selection techniques, which compares the network speed and latency. In this chapter, we will investigate some high level (i.e. application level) metrics e.g. benchmarks for web servers. Application level metrics can give a more precise measurement of how good an overall system completes assigned tasks. Our approach applies queueing theory [78] model to estimate the best fit resource allocation for achieving target QoS.

5.1 Problem Description

Decisions to migrate existing systems to public IaaS clouds can be complicated as evaluating the benefits, risks and costs of using Cloud computing is far from straightforward. Furthermore, given the large number of Cloud instance types even for a single data-center (such as AWS), it is not easy to select the type of server instance. Choosing different instance types has different impacts on cost and performance, but such decision making information is not readily available in a succinct and organized manner.

For example, when application developers/companies are considering renting Cloud Infrastructure, it would be beneficial for them to estimate how many servers they need to achieve certain performance, and associated total costs of such setting. There are different kinds of servers/VMs that are available from different Cloud providers with various costs. There are also different locations for data-centers that one can choose from each provider. Each kind of server will have different RAM capacity and CPU speed, which would result in different speeds for processing the same job.

We would like to estimate the system performance, give benchmarking results (servicing time) for certain tasks, and simulate system behaviors under different constraints, e.g. total cost and maximum wait time. For example, a company may be operating 10 servers with a configuration of 4 GB memory and 1.8 GHz CPU, but

they need to expand their business in a new country and support the double number of customers they currently support. They want to evaluate the option to move the in-house infrastructure to Cloud. First, they could run a benchmark software on their servers to get the service time for their business operations, e.g., run Apache JMeter to get the number of web requests they can handle per second for the whole cluster, i.e. 40 requests per 10 servers per second. Then they could run the same benchmark software on one of the Cloud servers (maybe with 16 GB RAM 3.6 GHz CPU). Suppose that they receive a 8 request per second result on a single VM. The simplest way to estimate the number of Cloud servers needed could be to divide the target throughput with the rate of processing, i.e. get 40/8=5 of this kind of Cloud servers to achieve the same requests per second (average/peak time) performance. However real world situations are often more complex and full of uncertainties, and thus queueing theory can serve as a robust mathematical model for probabilistic simulation.

In this work, we model systems to be migrated to a public Cloud as queueing systems. In an commercial web application scenario, customers would send requests (i.e. jobs) to the server. When the workload is heavy, such jobs would be queued (in a buffer). Assuming that the job arrival is according to a Poisson process with rate λ, i.e. the probability of the next job arrival only depends on the last one, but not anything before the last. This is also called *memorylessness*. Then we can model this process with the M/M/n queue. If the job arrivals are not following any pattern, thus any kind of distribution is possible and we can apply the M/G/n queueing model. In situations when the system is considered busy or under heavy traffic, heavy traffic approximation models can be applied.

5.2 Queueing theory based models

Let n be the **number of Cloud servers**, and *job_count* be **the number of jobs** during a test. There are different types of jobs. For web workload, a job can be a GET or POST HTTP request. Here, *job_count* refers to the number of requests received per second. For analytical applications, a job can be a specific task, like word count with map-reduce. For SQL or NoSQL databases, a job can be a query (e.g. SELECT, INSERT). The **average arrival rate** (λ), i.e. rate of arrival of jobs, is calculated by:

$$\lambda = \frac{job\_count}{duration} \tag{5.1}$$

where *duration* refers to the duration of a test. For example, 10000 calls came in over the course of one day and you want to calculate the arrival rate per minute. By Equation (5.1), we have $\frac{10000}{24\times 60} \approx 6.94444$, and the arrival rate is just about 7 calls per minute.

Let T_s be the **service time**, which is the average time taken to service a job. Then the **service rate** (μ), which is the number of jobs processed per unit time, can be

calculated as:

$$\mu = \frac{1 \ unit \ time}{T_s} \tag{5.2}$$

where **unit time** can be one second, one minute, one hour, etc. **Server utilization** (ρ) is the ratio defined as:

$$\rho = \frac{\lambda}{n\mu} \tag{5.3}$$

where λ can be calculated by Equation (5.1), and μ can be calculated by Equation (5.2). ρ sometimes is also referred to as the **traffic intensity** or the **agent occupancy**, as it represents the average proportion of time which each of the servers is occupied, assuming that jobs choose their servers randomly.

In a queueing system, a customer's time is spent on either waiting for services or getting services. The **average response time** (T_r) is the total amount of time a job spends both in the queue waiting for services and getting services:

$$T_r = T_w + T_s \tag{5.4}$$

where T_w is the average time spent on waiting to be served, and T_s is the average time spent on getting a service.

We assume that jobs are generated from an infinite population. So we will only look at those infinite-source queueing models.

5.2.1 Waiting time calculation

5.2.1.1 M/M/1 queue

For a system where its job arrival follows a Poisson distribution, and service times follows an exponential distribution, the simplest queue we can have is the M/M/1 queue with only one server, and its job arrival is according to a Poisson process with rate λ. The average waiting time can be computed as:

$$E(T_w^{M/M/1}) = \frac{\rho}{\mu(1-\rho)} \tag{5.5}$$

where $E(T_w^{M/M/1})$ can be read as the expected value of T_w for the M/M/1 queue.

5.2.1.2 M/M/n queue

Similar to the M/M/1 queue, when we have n servers, it becomes the M/M/n queue [126]. Its average job arrival rate can be calculated according to Equation (5.1). For example, if there are 360 jobs per minute, λ = 360/60 = 6 jobs per second.

In a system with n servers, if there are less than n jobs, some of the servers will be idle. If there are more than n jobs, the jobs will be queued in a buffer. Server utilization is calculated with Equation (5.3). For a queue to be stable $\rho < 1$ is required. The probability that an arriving job is forced to join the queue (all servers

are occupied) is denoted as:

$$C(n, \frac{\lambda}{\mu}) \tag{5.6}$$

which is best known as the Erlang's C formula [6, 59]:

$$C(n, \frac{\lambda}{\mu}) = \frac{1}{1 + (1-\rho)\frac{n!}{(n\rho)^n}\sum_{k=0}^{n-1}\frac{(n\rho)^n}{n!}} \tag{5.7}$$

where ρ is the server utilization and can be calculated by Equation (5.3). The average waiting time in the M/M/n queue can be calculated as:

$$E(T_w^{M/M/n}) = \frac{C(n, \frac{\lambda}{\mu})}{n\mu - \lambda} + \frac{1}{\mu} \tag{5.8}$$

Then, we can calculate the percentage of waiting times that are within a certain value (z):

$$B(T_w <= z) = (\frac{\lambda}{\mu})^n (n!)^{-1}[\sum_{k=1}^{n-1}(\frac{\lambda}{\mu})^k (k!)^{-1} + (\frac{\lambda}{\mu})^n (n!)^{-1}(1 - \frac{\lambda}{n\mu})^{-1}]^{-1}$$
$$n\mu(n - 1 - \frac{\lambda}{\mu})^{-1} e^{-\mu z}(1 - e^{-\mu(n-1-\frac{\lambda}{\mu})z}) \tag{5.9}$$

where e is the mathematical constant approximately equal to 2.71828, and z is the upper bound of waiting times we want to investigate. For example, if $B(T_w < 1\ sec) = 0.95$, it means 95 percent of the waiting times are within 1 second. We also call this the **error bound estimation (B)** of the waiting time [194].

5.2.1.3 M/G/1 queue

So far, we intestigated how waiting times can be calculate for systems with exponentially distributed service times. In many practical problems service times are not exponentially distributed, so we need to investigate queueing systems with generally distributed service times [124]. "Generally distributed" means that service times can follow any arbitrary distribution. The M/G/1 queue models a 1 server system with generally distributed service times. Its average waiting time is calculated with:

$$E(T_w^{M/G/1}) = \frac{\rho^2 + \lambda^2 var(T_s)}{2\lambda(1-\rho)} \tag{5.10}$$

where $var(T_s)$ is the variance of the service times. The variance of the service times is calculated by taking the arithmetic mean of the squared differences between each value and the mean value, which is also standard deviation squared, i.e. σ^2.

5.2.1.4 M/G/n queue

Generally distributed service time with n servers can be modeled by the M/G/n queue [125], and its average waiting time can be calculated by:

$$E(T_w^{M/G/n}) = \frac{1+CV^2}{2} E[T_w^{M/M/n}] \tag{5.11}$$

where CV^2 is the squared coefficient of variation of the service time distribution, which can be computed as:

$$CV = \frac{\sigma}{\bar{x}} \tag{5.12}$$

where σ is the standard deviation and $\bar{x}$ is the mean. The mean of the service time distribution is the average of service time, which is T_s.

The error bound estimation of a M/G/n queue waiting time can be calculated by applying the Markov's inequality [130]:

$$B(T_w \geq z) \leq \frac{E(T_w)}{z} \tag{5.13}$$

where $E(T_w)$ is the average waiting time. For example, the upper bound percentage of waiting times which are greater than 2 seconds would be $\frac{E(T_w^{M/G/n})}{2}$.

5.2.1.5 Heavy Traffic Approximation

In a system with high occupancy rates (ρ approaching 1), a heavy traffic approximation [86, 85] can be used to approximate waiting time, e.g. for the M/G/1 queue:

$$E(T_w^{M/G/1,H}) = \frac{\lambda(\frac{1}{\lambda^2} + var[T_s])}{2(1-\rho)} \tag{5.14}$$

The relative error of the heavy traffic approximation is defined as:

$$\frac{E(T_w^{M/G/1}) - E(T_w^{M/G/1,H})}{E(T_w^{M/G/1})} \tag{5.15}$$

which measures the relative difference of the heavy traffic approximation compared to the normal estimation of average waiting time in a queue, and can be calculated by:

$$\frac{1-\rho^2}{\rho^2 + \lambda^2 var(T_s)} \tag{5.16}$$

5.2.2 Number of servers needed

If we have a target response time to achieve, assuming all relevant information is known, we can estimate the number of servers needed.

Let R_{max} be the **response time maximum limit** specified by a user. If we use the M/M/n queueing model, we can let $E(T_w^{M/M/n}) = R_{max}$ to calculate the number of

minimum servers required. $C(n, \frac{\lambda}{\mu})$ is the probability that an arriving job is forced to join the queue. To simplify the calculation, we assume this value to be 1. Thus, we have:

$$R_{max} = \frac{1}{n\mu - \lambda} + \frac{1}{\mu} \tag{5.17}$$

Rearrange this equation leads to:

$$n = \frac{1}{\mu R_{max} - 1} + \frac{\lambda}{\mu} \tag{5.18}$$

Similarly, for cases when the M/G/n queue is the model, we have the following by applying Equation (5.11):

$$n = [\,(\frac{2R_{max}}{1+CV^2} - \frac{1}{\mu})^{-1} + \lambda]\,\frac{1}{\mu} \tag{5.19}$$

5.2.3 Throughput

According to Little's law, as shown in Equations (2.16) and (2.17), λ is also the mean throughput of the system. So when the response time constraint R_{max} is satisfied, the system achieves a throughput λ' which is equal to the arrival rate. Thus, we can deduce its calculation by rearranging Equation (5.18) as follows:

$$\lambda' = n\mu - \frac{\mu}{\mu R_{max} - 1} \tag{5.20}$$

Similarly, we have a M/G/n queue, Equation (5.19) should be used:

$$\lambda' = n\mu - (\frac{2R_{max}}{1+CV^2} - \frac{1}{\mu})^{-1} \tag{5.21}$$

5.2.4 Constraint satisfaction

If a customer has requirements on estimating the VM usage in the Cloud. It can be formulated as a constraint satisfaction problem and solved by linear optimization.

Firstly, among all available options, we filter out the ones that satisfy users' static requirements, such as location, provider, VM types, CPU speed, number of cores, memory size, etc. Then we calculate the total cost among those selected options. The cost of each **resource** (r) can be calculated by multiplying the **resource usage** (n_r) with its corresponding **price** (Co_r) and how long it is estimated to be required (T_r). Then we sum up the costs for VMs, storage and networks:

$$TotalCo = \sum_{r \in R} (n_r Co_r T_r) \tag{5.22}$$

where R is the set of all resource types. More details on cost calculation can be found in Chapter 4, and this section only represents it very abstractly.

When QoS constraints are given, such as **minimum throughput** (Th_{min}) and **maximum response time** (R_{max}). The objective is to minimize the cost. For a M/M/n queue, after substituting in Equations (5.8) and (5.20), our problem becomes:

$$\begin{array}{ll} \min & TotalCo \\ s.t. & \frac{1}{n\mu-\lambda} + \frac{1}{\mu} <= R_{max} \\ & n\mu - \frac{\mu}{\mu R_{max}-1} >= Th_{min} \end{array}$$

Alternatively, when the objective is to minimize waiting time, when satisfying user's **budget** (B_{max}) limitation, and not breaking performance promise on throughput at the same time, we get the following:

$$\begin{array}{ll} \min & \frac{1}{n\mu-\lambda} + \frac{1}{\mu} \\ s.t. & TotalCo <= B_{max} \\ & n\mu - \frac{\mu}{\mu R_{max}-1} >= Th_{min} \end{array}$$

Similarly, for a M/G/n queue, after substituting in Equations (5.11) and (5.21), we get the following respectively:

$$\begin{array}{ll} \min & TotalCo \\ s.t. & (\frac{1}{n\mu-\lambda} + \frac{1}{\mu})(1+CV^2)\frac{1}{2} <= R_{max} \\ & n\mu - (\frac{2R_{max}}{1+CV^2} - \frac{1}{\mu})^{-1} >= Th_{min} \end{array}$$

$$\begin{array}{ll} \min & (\frac{1}{n\mu-\lambda} + \frac{1}{\mu})(1+CV^2)\frac{1}{2} \\ s.t. & TotalCo <= B_{max} \\ & n\mu - (\frac{2R_{max}}{1+CV^2} - \frac{1}{\mu})^{-1} >= Th_{min} \end{array}$$

The form of QoS calculating functions (5.8), (5.11), (5.20) and (5.21) has a deciding influence on the nature of the optimization problem we are ending up with. With the assumption of the linearity of these functions as given above, our optimization problems fall in the class of mixed integer programming (MIP) [149]. The reason why they fall in the MIP class and not the linear programming class is that our variables can take only integer values, e.g. we cannot allocate 0.4 VMs of some type. Though MIP is not in the polynomial class of problems, there are industry implementations of efficient algorithms (up to an extent) to solve them. But in our work we are approximating the MIP to the linear program, allowing the relaxation that variables can take non integer values. After the solution is computed, we approximate the variables to the nearest integers (by taking the ceiling value).

5.3 Case Study

In this section, we give case studies on web applications and databases, about how workload can be modeled and measured in each situation.

5.3.1 Web application

Figure 5.1 shows the workload of the cluster which supports Wikimedia [221]. This figure includes one week's data, monitored by Ganglia [64]. Job arrival rate in this workload varies depending on the day of the week and the hour of the day, which is labeled as "Loads/Proces" on the y-axis. In this dataset, there is a 12-hour difference between the peak (at noon) and the trough (at midnight) in number of arriving jobs.

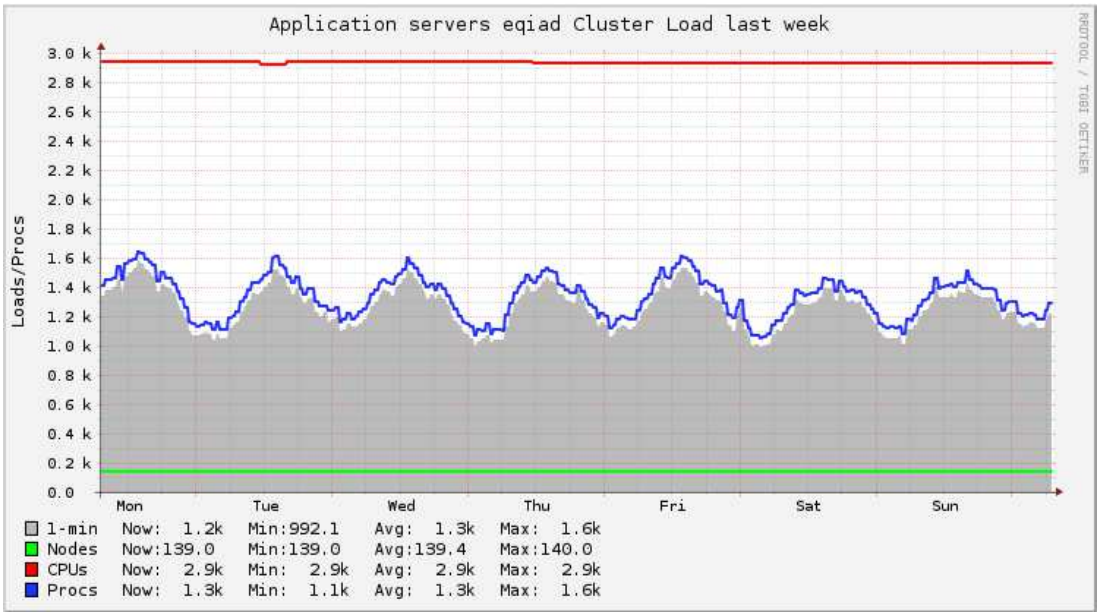


Figure 5.1: Wikimedia cluster's load between 18th to 24th August 2014 [65].

Overall, the workload follows a sine wave pattern. Similar patterns are also shown in the ClarkNet-HTTP trace dataset [38]. ClarkNet-HTTP contains two weeks of HTTP logs from the ClarkNet WWW server in July 1995. Figure 5.2 shows the number of GET requests received daily. The source and data for generating this histogram can be viewed online via plot.ly.[1]

When simulating such web workload, we can generate random requests accord-

[1] https://plot.ly/~miranda.zhang.q/28

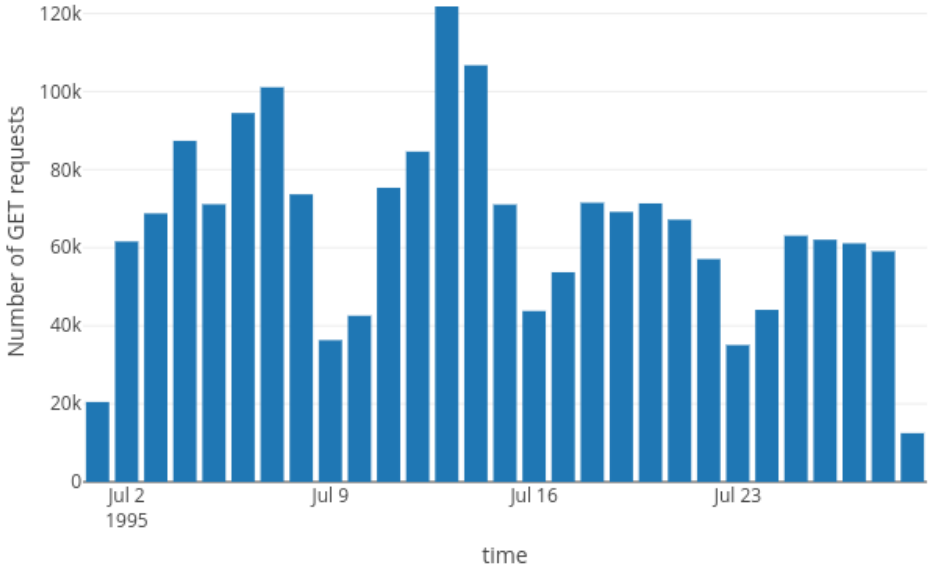


Figure 5.2: Number of GET requests received daily, plotted from ClarkNet-HTTP trace dataset.

ing to:

$$r = \gamma_{min} + 0.5(\gamma_{max} - \gamma_{min}) + 0.5(\gamma_{max} - \gamma_{min})sin(\frac{\pi t}{86400}) \tag{5.23}$$

where 86400 is the number of seconds in a day (24 hours), γ_{max} is the maximum number of requests per second on that day, γ_{min} is the minimum number of requests occurs on that day, and t is time (i.e. timestamp).

A range of tools can be used to benchmark websites or monitor web traffic and server workloads. For example, Jmeter can generate web workloads, and monitor the performance of the tested websites. In the end, Jmeter produces a report on a range of attributes, such as latency and throughput. The "latency" or "elapsed" column in a Jmeter results table is the response time measurements. Throughput is the number of requests sent to a server per unit of time (i.e. per second, per minute, etc.):

$$throughput = \frac{\text{number of requests}}{T_{end} - T_{start}} \tag{5.24}$$

Another tool is Apache Benchmark [3]. It can carry out tests similar to Jmeter. There are also web services for general network probing, such as RIPE Atlas [172], and cloud network monitoring service such as CloudHarmony.

5.3.2 Database

An example for the database benchmarking software is mysql-bench [139]. Despite having "mysql" in its name, it supports other databases as well, including Access, Adabas, AdabasD, Empress, Oracle, Informix, DB2, mSQL, MS-SQL, MySQL, Pg, Solid, Sybase, SAPdb, and SQLite. Mysql-bench benchmark suite is designed to test how a given SQL implementation performs. This benchmark is single-threaded, and measures the minimum time for the operations performed. It also measures the average time (in wall clock seconds) it takes for all of the mysql-bench benchmarks, i.e. alter-table, ATIS, big-tables, connect, create, insert, select, transactions and Wisconsin. For example, we could focus on two of the most important query types – SELECT (READ) and INSERT (WRITE). The mysql-insert benchmark issues 300,000 INSERT queries (if configured to run under default conditions) and calculates the total wall clock seconds to execute all queries. Thus, if the wall clock seconds is t, on a particular system, we can deduce that the service rate of that server to process insert queries is $\mu_w = \frac{300000}{t_w}$ queries/second. Similarly, we can deduce the service rate for SELECT queries as $\mu_r = \frac{1{,}194{,}631}{t_r}$ where the SELECT queries include the WHERE clause and thus SELECT queries with range parameters have been taken into account. This implies that, if the fraction of READ/WRITE requests is given as wr and ww, we can determine the average service rate as:

$$\mu = wr\mu_r + ww\mu_w \tag{5.25}$$

Thus, using benchmark results and the READ/WRITE ratio, the service rate of different servers for performing MySQL queries can be computed.

There are also other benchmarking softwares, such as pgbench [155] for PostgreSQL database and tpcc-mysql [202] for MySQL database.

5.4 Experiment

Our experiments are carried out with Apache JMeter and later imported to BlazeMeter [101]. BlazeMeter is a commercial, self-service load testing platform as a service that is compatible with JMeter. I have set up a simple web application on a machine with 16 GB RAM and 3.2 GHz CPU.

In the first test, we sent 581 identical GET requests to the web server. During the test, the server has an average of around 30% CPU utilization and 45% memory usage (16 GB * 0.45 ≈ 7.2 GB). Without tuning the default web server settings, the application is not exhausting the full potential of the hardware. The average processing time is 164 ms with a standard deviation of 0.028 seconds. We use those as the base values for the service time distribution, i.e. $T_s = 0.164$ and $\sigma = 0.028$.

Next we simulated a Poisson arrival workload. We used a timer called "Poisson Random Timer" to give a random pause to requests before they are sent to a server. Between consecutive requests, there is a random pause between 0 and 300 milliseconds. We sent 194 requests over 2 minutes (120 seconds).[2] So the arrival rate can be calculated using Equation (5.1), i.e. $\lambda \approx 1.6167$ requests per second. According to Equation (5.2), we have $\mu \approx 6.0976$ requests per second. Then by applying Equation (5.3), we have $\rho \approx 0.26513$. After applying Equation (5.10), we get $E(T_w^{M/G/1}) \approx 0.030447$. The estimated response time, according to Equation (5.4) is $T_r = T_w + T_s \approx 0.19445$ seconds. This is close to the measured average response time of 217 ms.

The Octave script for the above calculations is available online.[3]

5.5 Summary

This chapter proposed a technique to estimate the impact of underlying cloud resources on the performance of user applications/services. An informed user (i.e. infrastructure architects, CIOs, and system admins) may have benchmarking results in local environments and want to estimate application-level performances in the Cloud. The model introduced in this chapter simulates the application in the Cloud as a queueing system. So given the information on how fast one unit of resource can process one job, queuing formulas in the proposed technique can estimate the time taken for the whole system (made up of many resources) to process some workload (made up of many jobs) in the Cloud.

[2] https://github.com/miranda-zhang/Cloud-Infrastructure-Services-Selection-and-Evaluation/blob/master/blazemeter.com/sense-report-663269.pdf

[3] https://octave-online.net/bucket~NkdpWMdmmrFJq2VJ4AZYj3

Chapter 6

System Implementation and Evaluation

6.1 CoCoOn Websites and Services

This data model has evolved with the Cloud Service landscape. We started the model with taxonomy definitions of Cloud terminologies, which is described in Section 3.2. Then we extended part of the ontology and developed it into the current version, which is explained in Section 3.3. This section will explain how to use the ontology, including: i) How are the ontologies, documentation, datasets hosted? ii) How can data be collected and transformed with this ontology? iii) How to query the results with SPARQL? We also provide some other usage cases.

6.1.1 Ontology Usage Cases

The general workflow of CoCoOn is illustrated in Figure 6.1. A possible visualization of Azure's Compute service offers and regions is shown in Figure 6.2, with offers in green and regions in purple. Regions with more offers are big.

6.1.2 Mapping Data to Ontology

For converting data from various sources to semantic data, many methods were explored. Initially, we tried to transform the JSON file from the Provider APIs into a JSON-LD file by adding a `@context` to it. After experimenting with JSON-LD Macros, we realized there are many limitations. It only works well for simple cases, but it is not sufficient to cover more complex scenarios. So we moved to use SPARQL-Generate [117, 189] for defining the mappings.

6.1.2.1 Data Clean Up

Various data can be obtained from APIs of providers, in JSON or JS format. We then clean up/transform such data with jq. Next, we map the cleaned data to ontologies, and get results in the RDF turtle format.

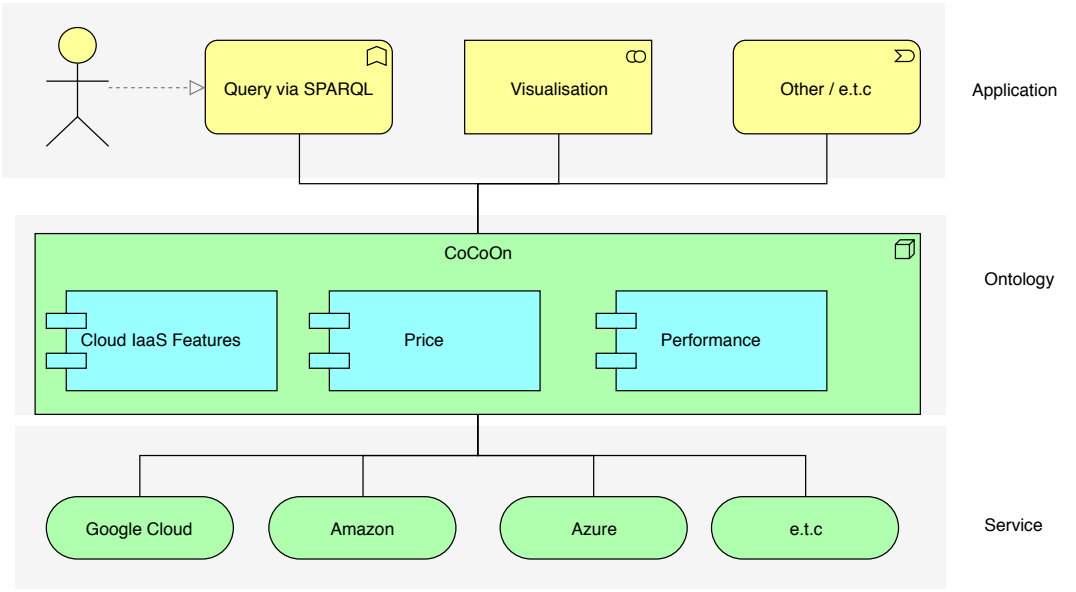


Figure 6.1: CoCoOn Data Integration Workflow

Listing 6.1 illustrates an example jq script transforms json data from Google API. The complete process with input and output for each step is documented online with more details.[1]

Listing 6.1: jq script transforms data from Google API

```
.gcp_price_list | . |=with_entries(select(.key| contains("VMIMAGE") )) |
[ to_entries[] |
    {
        "name": .key,
        "cores":(
            if (.key|contains("F1-MICRO")) then
                0.2
            elif (.key|contains("G1-SMALL")) then
                0.5
            else .value.cores end
        ),
        "memory": .value.memory,
        "gceu": (
            if .value.gceu == "Shared CPU, not guaranteed" then
                null
            else .value.gceu end
        ),
        "maxNumberOfPd": .value.maxNumberOfPd,
        "maxPdSize": .value.maxPdSize,
        "price":
         [
            .value | del(
                .cores, .memory, .gceu,
                .fixed, .maxNumberOfPd, .maxPdSize, .ssd)
            | to_entries[] | { "region": .key, "price": .value }
         ]
```

[1] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/gcloud/compute.md

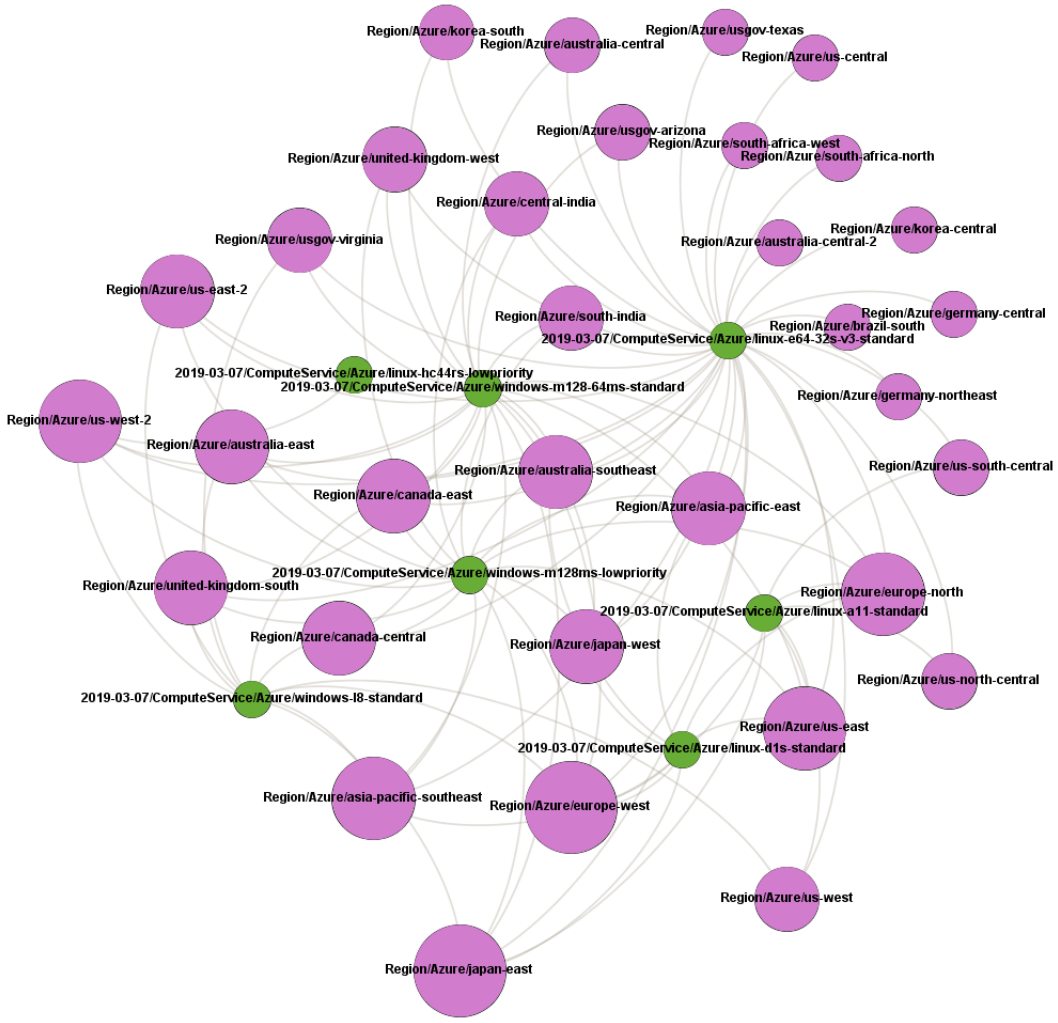


Figure 6.2: Azure Services Regions

```
    }
]
```

6.1.2.2 Annotating Plain Data with CoCoOn

For converting data from various sources to semantic data, we used SPARQL-Generate [117, 189] for defining the mappings. We developed many SPARQL-Generate scripts for this process. An example SPARQL-Generate script that maps json data from Azure API about managed disks to CoCoOn v1.0.1 is illustrated in Listing 6.2. The complete process with input and output for each step is documented

online with more details.[2]

Listing 6.2: SPARQL-Generate script maps json data to ontologies

```
BASE <https://w3id.org/cocoon/data/v1.0.1/>
PREFIX iter: <http://w3id.org/sparql-generate/iter/>
PREFIX fun: <http://w3id.org/sparql-generate/fn/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gr: <http://purl.org/goodrelations/v1#>
PREFIX cocoon: <http://w3id.org/cocoon/v1.0.1#>
PREFIX schema: <https://schema.org/>
PREFIX unit: <http://qudt.org/1.1/vocab/unit#>

GENERATE {
  ?iri a cocoon:NetworkStorage;
    rdfs:label ?name;
    schema:dateModified ?date;
    cocoon:hasProvider ?provider;

    GENERATE {
        ?iri gr:hasPriceSpecification [
            a gr:CloudServicePriceSpecification ;
                gr:hasCurrency "USD"^^xsd:string;
                cocoon:hasCurrencyValue ?value_normalized;
                gr:hasUnitOfMeasurement cocoon:GBPerMonth ;
                cocoon:inRegion <Region/{?provider_slug}/{?region}>;
        ]
    }
      ITERATOR iter:JSONListKeys(?prices) AS ?region
    WHERE {
        FILTER( !CONTAINS(?managed_disk,"ultrassd") )
        BIND ( xsd:float( fun:JSONPath(?prices,"$.['{?region}'].value") ) AS ?value )
        BIND ( xsd:nonNegativeInteger( fun:JSONPath(?managed_disk_json,"$.size") ) AS ?size)
        BIND ( xsd:decimal( IF( BOUND(?size) , ?value/?size , ?value ) ) AS ?value_normalized )
    } .

    GENERATE {
        ?iri
        cocoon:hasStorageIOMax [
            a schema:TypeAndQuantityNode;
                schema:amountOfThisGood ?iops;
                schema:unitCode cocoon:IOPs;
        ];
        cocoon:hasStorageSize [
            a schema:TypeAndQuantityNode;
                schema:amountOfThisGood ?size;
                schema:unitCode cocoon:GB;
        ];
        cocoon:hasStorageThroughputMax [
            a schema:TypeAndQuantityNode;
                schema:amountOfThisGood ?speed;
                schema:unitCode unit:MegabitsPerSecond ;
        ];
    } WHERE {
        BIND (xsd:nonNegativeInteger( fun:JSONPath(?managed_disk_json,"$.iops") ) AS ?iops )
        BIND (xsd:nonNegativeInteger( fun:JSONPath(?managed_disk_json,"$.size") ) AS ?size)
        BIND (xsd:nonNegativeInteger( fun:JSONPath(?managed_disk_json,"$.speed") ) AS ?speed)
        FILTER( BOUND(?iops) )
        # either ?iops ?size ?speed are all bound or non is bound
```

[2] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/azure/storage.md

```
    } .

    GENERATE {
        ?iri
            gr:hasPriceSpecification <{?updated}/CloudStorageTransactionsPriceSpecification/{?
                ↪ provider_slug}/managed_disk/transactions-{?type}>;
    } WHERE {
        FILTER( ! CONTAINS(?managed_disk,"snapshot") && ! CONTAINS(?managed_disk,"ultrassd"))
        FILTER( CONTAINS(?managed_disk,"hdd") || CONTAINS(?managed_disk,"ssd") )
        BIND ( IF( CONTAINS(?managed_disk,"hdd") , "hdd" ,"ssd" ) AS ?type )
    } .

    GENERATE {
        ?iri
            cocoon:canHaveSnapshot <{?updated}/NetworkStorage/{?provider_slug}/standardssd-
                ↪ snapshot>;
            cocoon:canHaveSnapshot <{?updated}/NetworkStorage/{?provider_slug}/standardhdd-
                ↪ snapshot-lrs>;
            cocoon:canHaveSnapshot <{?updated}/NetworkStorage/{?provider_slug}/standardhdd-
                ↪ snapshot-zrs>;
            cocoon:canHaveSnapshot <{?updated}/NetworkStorage/{?provider_slug}/{?
                ↪ premiumssd_snapshot}>;
    } WHERE {
        FILTER( !CONTAINS(?managed_disk,"snapshot") )
        FILTER( CONTAINS(?managed_disk,"premiumssd") || CONTAINS(?managed_disk,"standardssd"))
        BIND ( IF( CONTAINS(?managed_disk, "premiumssd") , "premiumssd-snapshot" , ?undefined )
            ↪ AS ?premiumssd_snapshot )
    } .

    GENERATE {
        ?iri gr:hasPriceSpecification [
            a gr:CloudServicePriceSpecification ;
                gr:hasCurrency "USD"^^xsd:string;
                cocoon:hasCurrencyValue ?value;
                gr:hasUnitOfMeasurement ?unit ;
                cocoon:inRegion <Region/{?provider_slug}/{?region}>;
                rdfs:label ?label;
        ].
    }
      ITERATOR iter:JSONListKeys(?prices) AS ?region
    WHERE {
        FILTER( CONTAINS(?managed_disk,"ultrassd") )
        BIND (xsd:decimal( fun:JSONPath(?prices,"$.['{?region}'].value") ) AS ?value )
        BIND (
            COALESCE(
                IF(CONTAINS(?managed_disk,"iops"), cocoon:IOPsPerHour, 1/0),
                IF(CONTAINS(?managed_disk,"stored"), cocoon:GBPerHour, 1/0),
                IF(CONTAINS(?managed_disk,"throughput"), cocoon:MegabitsPerSecondPerHour, 1/0),
                cocoon:VcpuPerHour # assume "vcpu"
            ) AS ?unit
        )
        BIND ( STRAFTER(?managed_disk,"-") AS ?label)
    } .
}
SOURCE <https://raw.githubusercontent.com/miranda-zhang/cloud-computing-schema/master/example/jq
    ↪ /azure/2019-03-07/managed-disks.json> AS ?source
ITERATOR iter:JSONListKeys(?source) AS ?managed_disk
WHERE {
    BIND (fun:JSONPath(?source,"$.['{?managed_disk}']") AS ?managed_disk_json)
    BIND (fun:JSONPath(?managed_disk_json,"$.prices") AS ?prices)
    # having single point of change for the following
    BIND ( "Azure" as ?provider_slug )
```

```
    BIND ( cocoon:Azure as ?provider )
    BIND ( "2019-03-07" as ?updated) # yr-month-day
    BIND ( xsd:date(?updated) as ?date )
    BIND ( IF (CONTAINS(?managed_disk,"ultrassd"), "ultrassd", ?managed_disk) AS ?name )
    BIND ( <{?updated}/NetworkStorage/{?provider_slug}/{?name}> AS ?iri )
}
```

6.1.2.3 Dataset

We have also made the complete datasets (132,282 triples) available at github. It is recommended to download the data and investigate with a triplestore. For example, you can run a query as shown in Listing 6.3, and the results are shown in Table 6.1.

Table 6.1: Instance counts of the classes

Class	Count
cocoon:NetworkStorage	45
cocoon:ComputeService	1021
cocoon:Region	55
cocoon:StorageService	161
cocoon:Location	5
cocoon:InternetService	6
cocoon:SystemImage	10

Listing 6.3: A SPARQL query

```
PREFIX cocoon: <https://w3id.org/cocoon/
    ↪ v1.0.1#>
PREFIX gr: <http://purl.org/goodrelations
    ↪ /v1#>
SELECT ?cls (COUNT(?s) AS ?count)
{
    VALUES ?cls {cocoon:ComputeService
        ↪ cocoon:SystemImage cocoon:
        ↪ StorageService cocoon:
        ↪ NetworkStorage cocoon:
        ↪ NetworkService cocoon:
        ↪ InternetService cocoon:Region
        ↪  cocoon:Location gr:
        ↪ BusinessEntity
    } ?s a ?cls
} GROUP BY ?cls
```

Data can also be hosted with a Linked Data Fragments Server. Example project setup is also documented in the github directory.

6.2 QoS Profiler

In this section, we describe the implementation of the QoS profiler proposed in Section 4.1.2.

6.2.1 QoS Data Collection Scripts

Initially we used the HtmlUnit library [92], to collect QoS data from CloudHarmony. Later, as CloudHarmony evolved, we also upgraded our script, as shown in a live demos for measuring downlink speed and latency for Google Cloud services.[3] Up-

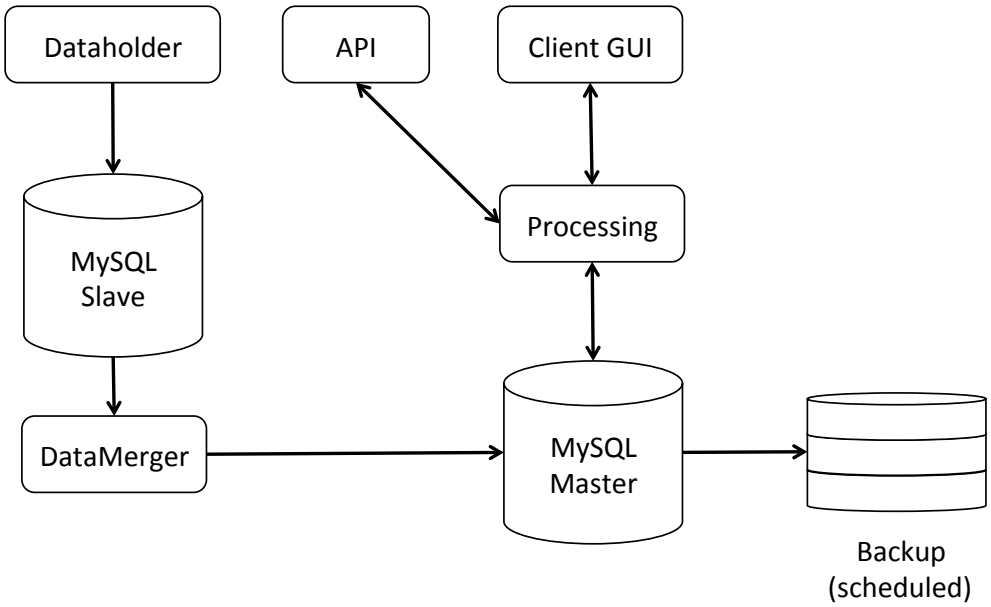
[3] https://miranda-zhang.github.io/cloud-computing-schema/cloudharmony/google/test.html

link tests script[4] is written in Python as selenium[5] is required. Additional details on using cloudharmony for measuring QoS are documented online.[6]

6.2.2 Distributed Architecture

Figure 6.3 shows the top level dataflow of our QoS profiler. It is better to look into this figure together with Figure 4.2. We have used several (slave) servers to collect data from different locations. Then we transferred them to a central server for processing and backup. Data on this server was also archived and cleared manually every time after we imported the newly collected data into the local offline system for post-processing and cleaning up. We only used the (summarised) average QoS data for real-time querying via API and web GUI, as this allows us to respond faster.

Figure 6.3: System Dataflow.



6.3 CloudRecommender

Despite the popularity of Cloud Computing, existing Cloud Service manipulations (e.g. select, start, stop, configure, delete, scale and de-scale) techniques require human familiarity with different Cloud service types and typically rely on procedural programming or scripting languages. The interaction with services is performed

[4] https://github.com/miranda-zhang/cloud-computing-schema/blob/master/example/cloudharmony/selenium/cloudharmony.py

[5] https://github.com/miranda-zhang/cloud-computing-schema/tree/master/example/cloudharmony/selenium

[6] https://github.com/miranda-zhang/cloud-computing-schema/tree/master/example/cloudharmony

through low-level application programming interfaces (APIs) and command line interfaces. As a result, accessibility to Cloud Computing is limited to decision makers with high IT expertise. This is inadequate, given the proliferation of new providers offering services at different layers (e.g. SaaS, PaaS, and IaaS). This raises a set of research questions: How to develop interfaces that can transform low, system-level programming to easy-to-use drag and drop operations? Will such interfaces improve and simplify the process of Cloud Service Selection and Comparison?

Therefore, we present CloudRecommender, a decision support system, using transactional SQL semantics, procedures and views. The benefits to users of CloudRecommender include, for example, the ability to estimate costs, compute cost savings across multiple providers with possible trade-offs, and provide a visual aid in the selection of Cloud services.

Before CloudRecommender, there have been a variety of systems that use declarative logic-based techniques for managing resources in distributed computing systems. The focus of the authors in the work [122] is to provide a distributed platform that enables Cloud providers to automate the process of service orchestration via the use of declarative policy languages. The authors in [28] present an SQL-based decision query language for providing a high-level abstraction for intuitively expressing decision guidance problems so that database programmers can use mathematical programming techniques without prior experience. We also draw inspiration from the work in [129], which proposes a data-centric (declarative) framework to improve SLA fulfilment ability of Cloud service providers by dynamically relocating infrastructure services. COOLDAID [35] presents a declarative approach to managing configurations of network devices and adopts a relational data model and Datalog-style query language. NetDB [30] uses a relational database to manage the configurations of network devices. However, NetDB is a data warehouse, not designed for Cloud service selection or comparison.

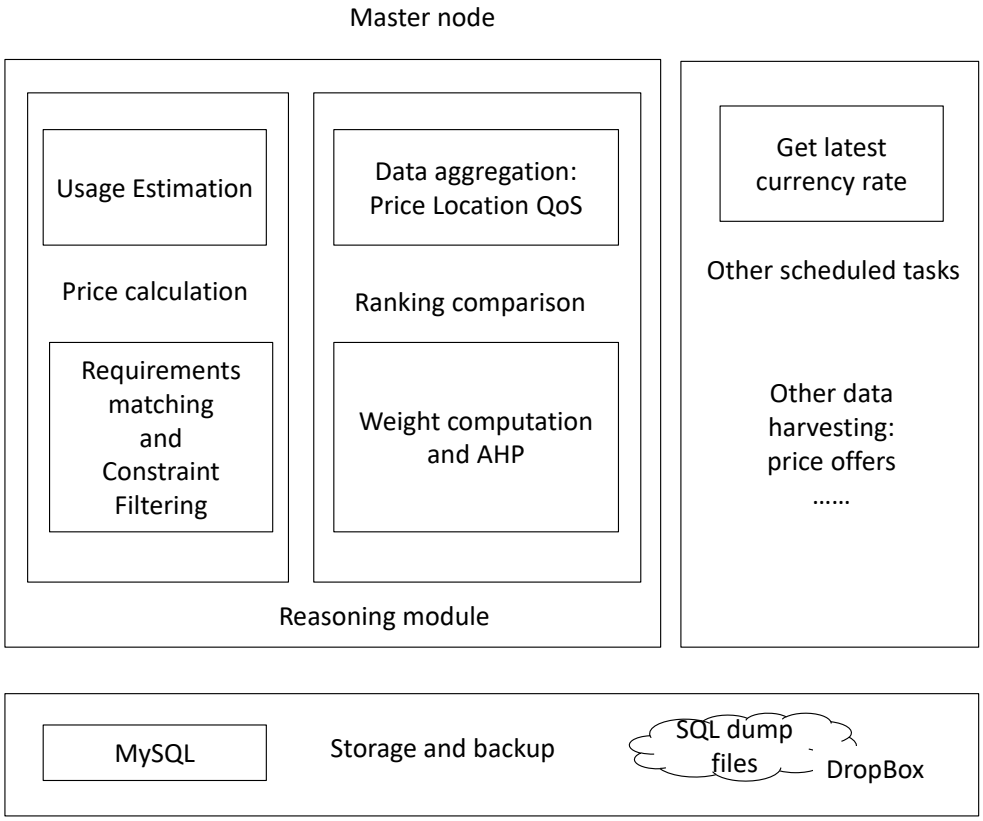
In contrast to the approaches as mentioned above, CloudRecommender is designed for solving the new challenges of handling heterogeneous service configuration and naming conventions in Cloud computing. It is designed with a different application domain – one that aims to apply declarative and widget programming techniques for solving the Cloud service selection problem.

6.3.1 System Architecture

The CloudRecomender works together with the QoS profiler, and they can be connected through a master-slave architecture. For example, Figure 6.4 shows our prototype design for the master node. We used Dropbox as the backup service for this implementation to demonstrate the feasibility. As long as data is properly backed up in a separate location, other mechanisms can be used, i.e. git repository.

The price data is collected from providers' websites. The problem with automatic data collection can be solved if providers release more structured data with sufficient metadata descriptions, using the ontology we have proposed in Chapter 3. The CloudRecommender system architecture consists of three layers: the configuration

Figure 6.4: CloudRecommender System Architecture.



In the reasoning module, main functions and operations are broken down into different blocks. Some other tasks cannot be strictly categorized into existing modules and are put into the other tasks section. While it is possible to back up the whole server, it is not necessary. Data stored in the MySQL database can be backed much easier and cheaper by creating SQL dump. A dump file is created daily and stored in a Dropbox folder which is free to use and keeps a history of the file stored in it for 30 days. The presentation layer (UI and API implementation) and the monitoring module are omitted to keep the diagram simple.

management layer, the application logic layer and the User interface (widget) layer, as shown in Figure 6.5.

Part (b) of Figure 6.5 shows the deployment structure of the CloudRecommender system. For persistence, we have chosen MySQL for its agility and popularity, but any other relational database can be plugged in. Furthermore, many APIs provided by cloud providers (such as Amazon) and open source cloud management frameworks (e.g. jclouds) are written in Java. Thus, Java is chosen as the preferred language to implement the application logic layer to ease the integration with external libraries.

The configuration layer maintains the basic cloud domain model related to compute, storage, and network services. Initially, we stored recommender system data into a relational database, and its implementation is detailed in Section 6.3.1.1. However, as we go more in-depth with the understanding of the Cloud domain, we find

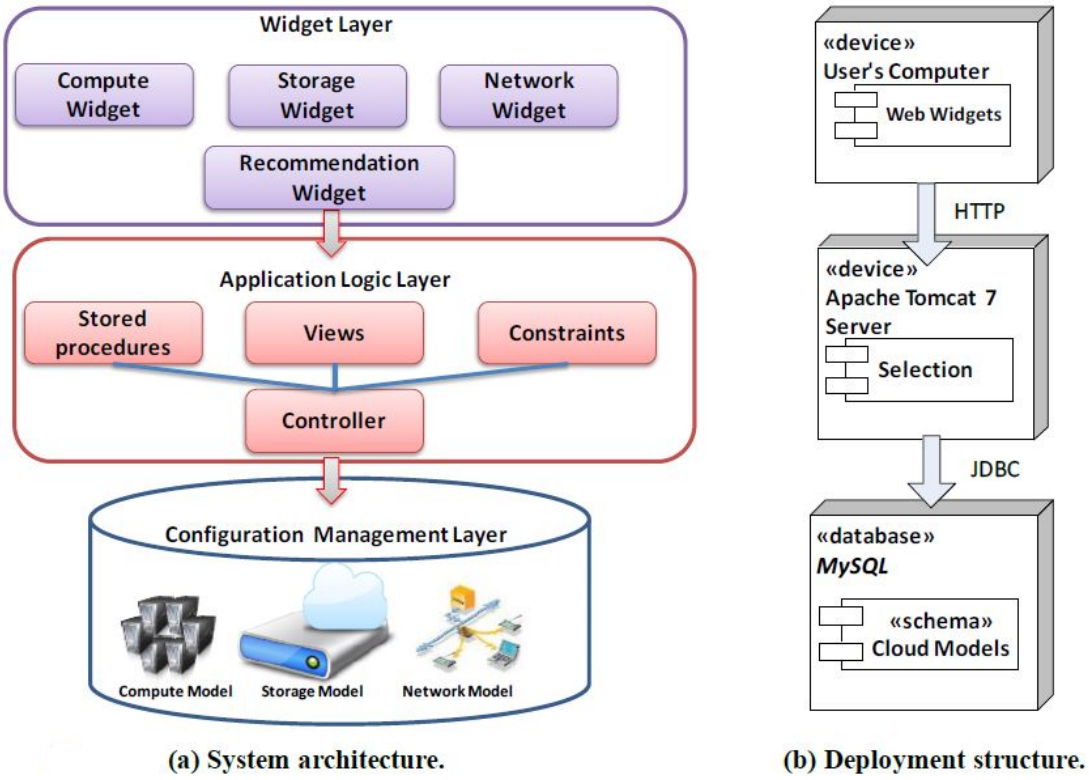


Figure 6.5: System architecture and deployment structure

this database model hard to update, thus hard to keep up to date with rapidly changing Cloud services. Furthermore, this database model is also not very readable to users who do not know our recommender system, as the database schema is designed to be optimal for recommendation services with normalized tables. So later, we developed the Cloud Computing Ontology (CoCoOn) model with OWL, which is discussed in Chapter 3. CoCoOn is flexible and extensible enough to accommodate new services with minimal changes.

6.3.1.1 Configuration Management Layer

Infrastructure services from different providers have different configurations and pricing models, which we described in Section 4.1. We collected service configuration information from many public cloud providers (e.g., Windows Azure, Amazon, GoGrid, RackSpace, Nirvanix, Ninefold, SoftLayer, AT and T Synaptic, Cloud Central, etc.) to demonstrate the generic nature of the domain model for capturing heterogeneous configuration information of infrastructure services.

We formally capture the domain knowledge (e.g., IaaS configurations) using a declarative logic-based language. Based on the domain knowledge, we have drawn the relationships in the conceptual IaaS configuration model and represented in Figure 6.6. Relationships are carefully considered and normalized to avoid update anomalies. Services from various providers often have different configurations and

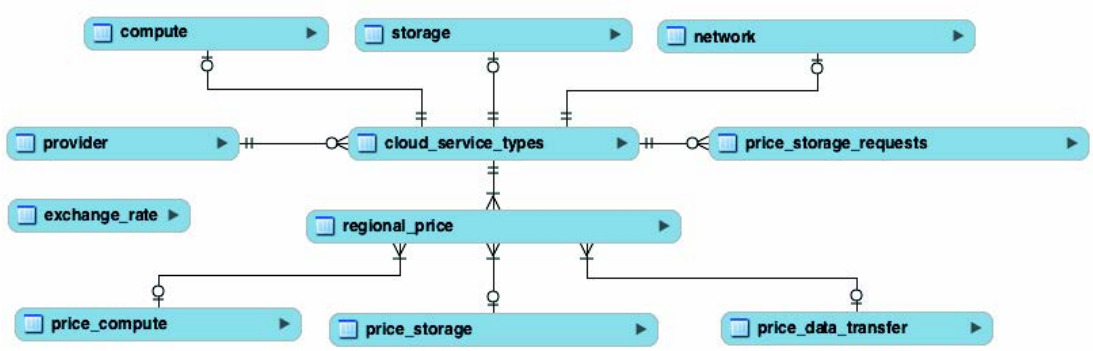


Figure 6.6: UML data model representing infrastructure service entities and their relationships

pricing models. Distinct and ambiguous terminologies are often used to describe similar configurations. Regardless of how providers name their services, we categorize infrastructure services based on their basic functionality. Unit conversions were performed during instantiation of concepts.

The choice of a relational model and SQL as a query language was made because of the convenience SQL procedures offer us, in regards to defining templates for a given widget type. We use stored procedures to create temporary tables and to concatenate parameters to generate queries based on a user's input dynamically.

For providers that offer different regional prices, we store the location information in the price table. If multiple regions have the same price, we choose to combine them. In the database version implementation, any changes to existing configurations (such as updating memory size, storage provision, etc.) of services can be done by executing customized SQL queries.

We have applied declarative service selection techniques by utilizing SQL and regular expressions to minimize side effects and reinforce constraints. This leads to an improved Cloud service representation and selection.

The service selection logic developed by our research is transactional and applies well-defined SQL semantics for querying, inserting, and updating IaaS configurations. Also, the proposed declarative approach allows us to take advantage of optimized query operations (e.g. select and join).

The problem we aim to solve involves computing the Cartesian product O(N * M) of multiple sets of options. A widely used solution of such operation is the JOIN operation in the database. Note that much work in database-systems has aimed at efficient implementation of joins. Modern databases often use HASH JOIN O(N + M) and MERGE JOIN O(N*Log(N) + M*Log(M)). They are much faster than O(N * M).

6.3.1.2 Application Logic Layer

The request for service selection in CloudRecommender is expressed as SQL queries. The selection process supports an application logic that builds upon the following

```
SELECT *
FROM `compute_service_price`
left join compute_choice_criterias
on `Memory(GB)` >= compute_choice_criterias.ram_low
and `Memory(GB)` <= compute_choice_criterias.ram_high
and `Local Storage(GB)` >= compute_choice_criterias.local_storage_low
and `Local Storage(GB)` <= compute_choice_criterias.local_storage_high
where if(all_provider_considered, 1, find_in_set(`Provider Name`,provider_LIST))
```

Figure 6.7: Example query in procedure.

declarative constructs: criterion, views and stored procedures. The CloudRecommender builds upon SQL queries which are executed on top of the relational data model.

Criterion: Criterion is a quantitative or qualitative bound (minimum, maximum, equal) on the configuration parameters provided by a service. Cloud services' configuration parameters and their range/values listed in Table 6.2 form the basis for expressing selection goal and criteria (e.g., select a cheapest (goal) compute service where (criterion) 0<Ram<=20, 0<=local storage<=2040, number of hours to be used per month = 244). An example query is shown below in Figure 6.7:

Procedures: We have implemented many customized procedures that automate the service selection process. Many routines are prepared to process a user service selection request. A list of inputs is stored in a temporary table to be passed into the procedures. As such, the size of the input list can be very long. Final results are also stored in temporary tables, which are automatically cleared after the expiration of the user session.

Controller: The controller supports enforcement of criteria and dynamically generates SQL queries which fulfil a given selection preference stated by the user. Due to space considerations, we are not able to depict the complete algorithm, but Figure 6.8 shows the selection logic in a simplified diagram. Next, we explain the basic steps which are executed for resolving a service selection request:

1. Basic validation is performed on user inputs at the controller, and appropriate errors are returned accordingly.

2. Depending on a user's requirements, the steps 3.2 and 3.3 may not happen. This is why they are shown in dotted lines, i.e. a user can query storage or compute only IaaS services. However, data transfer parameters have to be set. A user will transfer data in and out of the compute or storage services.

3. Multiple temporary tables are created during the process, so intermediate results (i.e. selection details of the final recommendation) can be fetched later as needed.

4. It is possible for a user to choose multiple compute services, each with different criteria. For example, they may have ten sets of requirements and choose five instances for each. So at step 5, queries with different numbers of join operations are dynamically constructed.

Service	Configurations Parameters	Range/possible values
Compute	Core	>=1
	CPUClockSpeed	>0
	hasMemory	>0
	hasCapacity	>=0
	Location	North America, South America, Africa, Europe, Asia, Australia
	CostPerPeriod	>= 0
	PeriodLength	>0
	CostOverLimit	>= 0
	PlanType	Pay-as-you-go, Prepaid
Storage	StorageSizeMin	>= 0
	StorageSizeMax	> 0
	CostPerPeriod (e.g. Period = Month) (e.g. UnitOfMeasurement = GB)	>= 0
	Location	North America, South America, Africa, Europe, Asia, Australia
	RequestType	put, copy, post, list, get, delete, search
	CostPerRequest	>= 0
	PlanType	Pay-as-you-go, Prepaid, Reduced Redundancy
Network	CostDataTransferIn	>=0
	CostDataTransferOut	>=0

Table 6.2: Infrastructure service types and their configurations

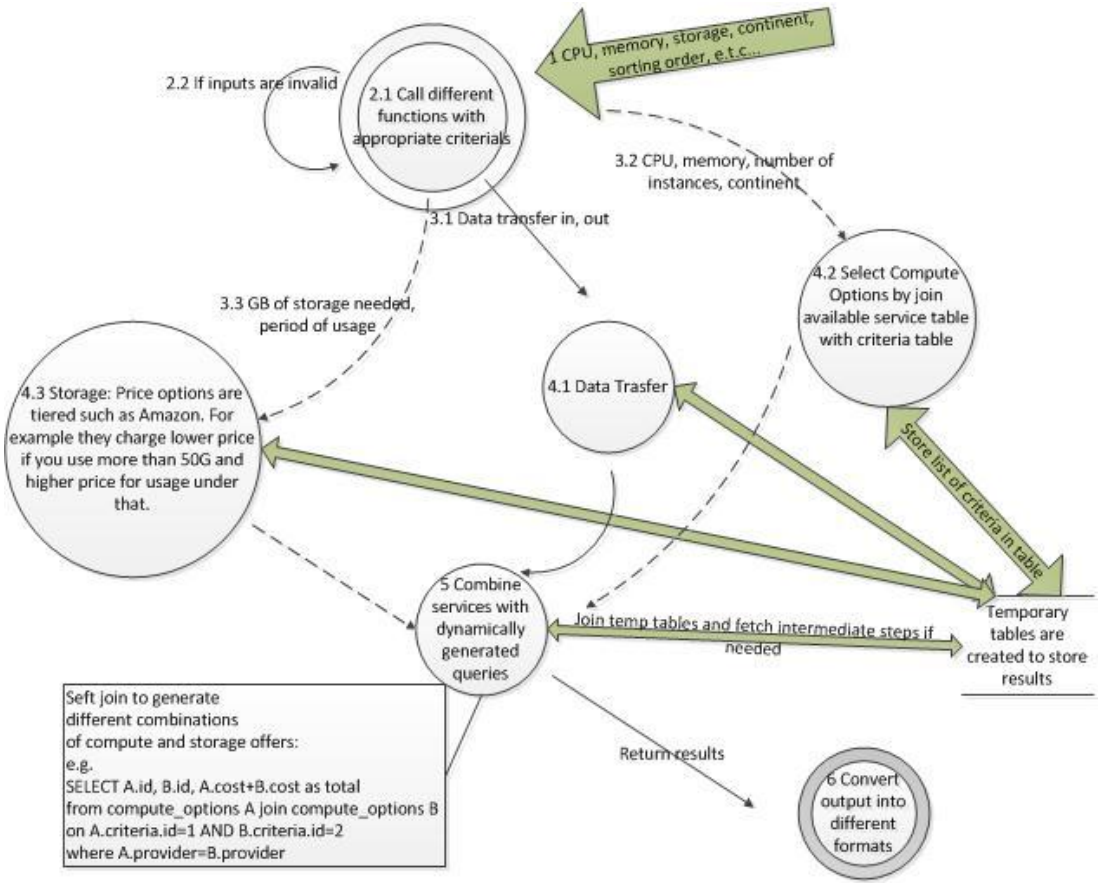


Figure 6.8: Service Selection logic.

5. Currency conversions are performed before costs are compared.

Computational Complexity of Service Selection Logic: For a provider p, suppose it has r regions, v different kinds of VMs, s storage options, and n network services. There will be $r \times v$ different prices for VM, similarly for prices of storage ($r \times s$) and network ($r \times n$). Theoretically, a total number of $r^3 \times v \times s \times n$ possible combinations. We can generally reduce the number of options significantly in the early stage if a user has strict requirements. In the worst case scenario, the logic needs to compute a full cross join (Cartesian product).

Another factor affecting the price calculation is the different price tiers for some services. For example, AWS S3 charges 0.125 USD per GB for the first 1 TB / the month of usage, 0.093 for the next 49 TB, etc. Depending on the estimated usage, the more significant the usage, the more price tiers will be involved.

Let us assume that each provider offers approximately the same service in each region to simplify the derivation of the computational complexity.

Let cs_p be the number of compute services of provider p. All regions are included, i.e. $cs_p = r \times v$. Let ss_p be the number of storage services, and st_p be the number of storage service tiers, nt_p be the number of network service tiers, ns_p be the number of

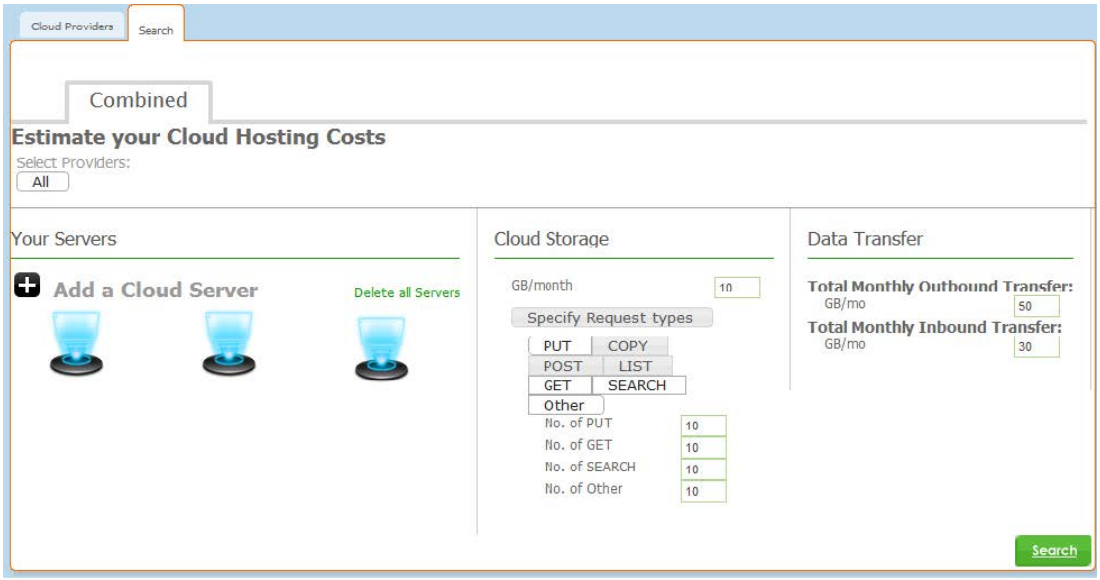


Figure 6.9: Screenshot of the widget combined tab.

network services. As such, the total number of offers can be represented as follows:

$$\sum_{p=1}^{P} cs_p \times (ss_p \times st_p) \times (ns_p \times nt_p)$$

where P is the number of cloud providers.

The queries of the selection logic work as follows. After filtering out criteria-violating services, resulting services are combined via JOIN operation(s) with final costs calculated. In the worst case scenario where a few or no criteria are defined, the combination of the services is a full CROSS JOIN over all existing services. Therefore, the selection queries, to our best knowledge, have the upper bound computational complexity of

$$O_{query}(\sum_{p=1}^{P} |cs_p c_v (ss_p st_p c_s)(ns_p nt_p c_n)|)$$

where c means criteria, and c_v, c_s, c_n are criteria for compute, storage and network respectively. $ss_p st_p$ and $ns_p nt_p$ can be pre-computed, and stored as views.

6.3.2 Interface: GUI and API

The widget layer is implemented using many JavaScript frameworks, including jQuery, ExtJS and YUI. CloudRecommender also exposes RESTful (REpresentational State Transfer) APIs (application programming interface) that help external applications to programmatically compose infrastructure cloud services based on the CloudRecommender selection process.

This layer features a rich set of user-interfaces, as shown in Figures 6.9 and 6.10.

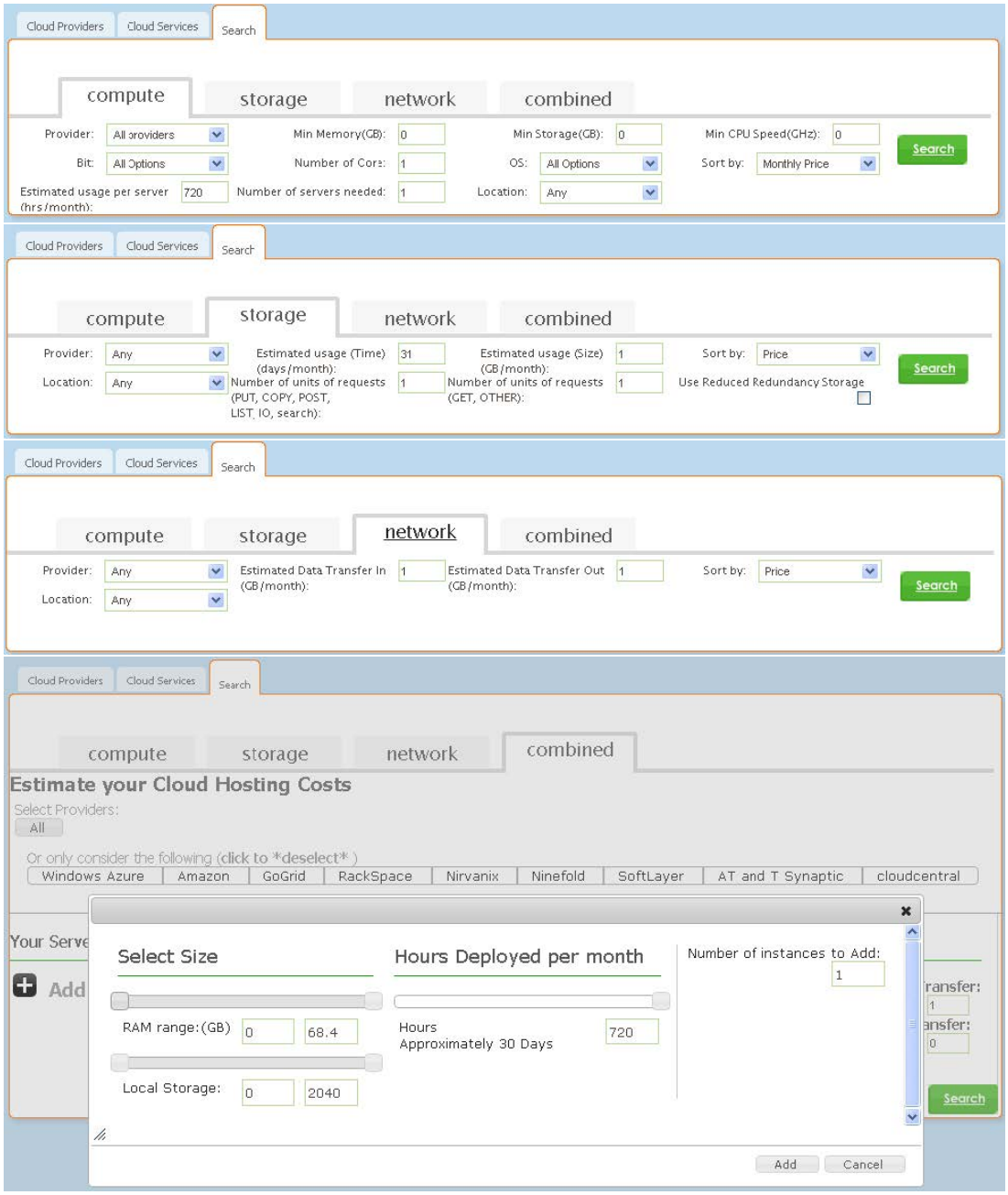


Figure 6.10: Compute, Storage, Network and the combined service selection widgets screenshots.

Those interfaces further simplify the selection of configuration parameters related to cloud services. This layer encapsulates the user interface components in the form of four principle widgets, including Compute, Storage, Network, and Recommendation. The selection of basic configuration parameters related to compute services, including their RAM capacity, cores, and location can be facilitated through the Compute widget. It also allows users to sort compute services by a specific column, and search by using regular expressions. Using the Compute widget, users can choose which columns to display and rearrange their order as well. The Storage widget allows users to define configuration parameters such as storage size and request types (e.g., get, put, post, copy, etc.). Service configuration parameters, such as the size of incoming data transfer and outgoing data transfer, can be issued via the Network widget. Users have the option to select single service types as well as bundled (combined search) services driven by use cases. The selection results are displayed and can be browsed via the recommendation widget, which is illustrated in Section 6.3.4.

CloudRecommender also exposes REpresentational State Transfer (RESTful) APIs that help external applications to programmatically obtain results, i.e. recommended infrastructure Cloud services configurations, as shown in Figure 6.11.

Figure 6.11: REST call via HTTP GET request

6.3.3 Experiment

In this section, we present the experiments and evaluation that we undertook.

Experiment Setup: We hosted our CloudRecommender system instance on Amazon EC2 using a standard small instance in the US-west location. By default, the small instance has the following hardware configuration: 1.7 GB of main memory, 1 EC2 Compute Unit, 160 GB of local instance storage, and a 32-bit platform with an Ubuntu 10.04 operating system. We populated CloudRecommender with infrastructure service configuration information related to Amazon, Azure, GoGrid, RackSpace, Nirvanix, Ninefold, SoftLayer, AT and T Synaptic, and Cloud Central (an Australian provider).

Service Selection Test: Suppose an enterprise wants to migrate its data to the cloud with the aim of storing and sharing it with other branches through public cloud

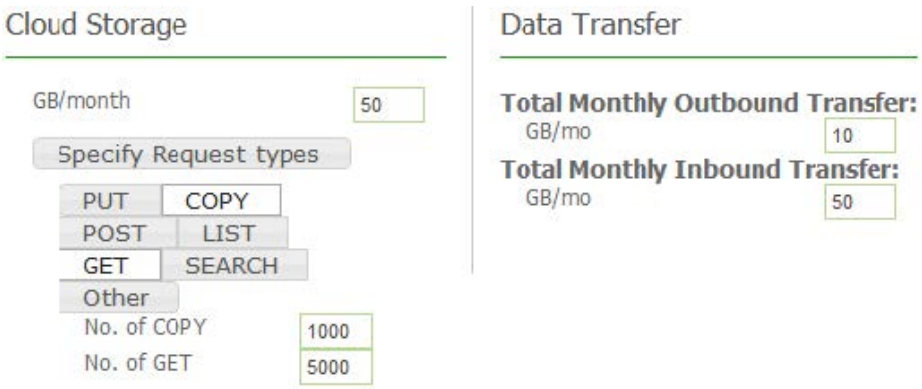


Figure 6.12: Service selection criteria set by business analyst.

storage (note that security issues are dealt within the enterprise applications). At this stage, we assume the business analyst of the enterprise has a reasonable estimation of the data storage and transfer (network in/network out) requirements. By using CloudRecommender, the analyst would like to find out which of the public cloud providers would be the most cost-effective in regards to data storage and transfer costs. For this selection scenario, suppose the analyst has the following anticipated usage for the storage and network services: (i) Data size of 50 GB, 1000 copy requests and 5000 get requests and (ii) data transfer in size of 10 GB and data transfer out the size of 50 GB. As shown in Figure 6.12, the analyst specifies service selection criteria via the storage and network widgets. The above request can also be submitted via the RESTful service interface of the CloudRecommender, as shown below in Figure 6.13.

Call

```
localhost/cloud_demo_1_1/api/cost/storage?
media_type=xml&currency=AUD&storage=50&duration=31&data_upload_size=50&data_download_size=10&copy=1000&get=5000
```

Figure 6.13: An Example REST call.

Once this selection request is submitted, the controller validates and parameterizes the criteria (user inputs). Though not shown in the above figures, the business analyst also has the option to express whether the selection criteria should be evaluated against all the available cloud providers or only the selected ones (e.g., Amazon, Azure, and GoGrid only). As mentioned earlier, the application logic layer implements specialized views and procedures for evaluating different service selection scenarios. CloudRecommender captures and inserts multiple storage and network service selection criteria into specialized views called "`storage_selection_criteria`" and "`network_selection_criteria`". The aforementioned views are then joined against the "`storage_service_price`" and "`network_service_price`" views for estimating the cost of using the combined cloud services.

Figure 6.14 shows the result of the selection scenario depicted in Figure 6.12. Results are sorted into increasing total cost order (i.e. "`storage_dataTransfer_cost`"

Provider Name	region_name	storage_cost	cost_data_in	cost_data_out	storage_dataTransfer_cost
SoftLayer	Any	6.00000000	0.000	1.000	7
Windows Azure	North America and Europe	7.00000000	0.000	1.200	8.205
Amazon	Asia Pacific(Tokyo)	6.76500000	0.000	0.000	8.589
Windows Azure	Asia Pacific Region	7.00000000	0.000	1.900	8.905
Ninefold	Any	4.60000000	0.000	9.000	13.606
AT and T Synaptic	Any	10.00000000	5.000	1.000	16
AT and T Synaptic	Any	10.00000000	5.000	1.000	16
AT and T Synaptic	Any	10.00000000	5.000	1.000	16
AT and T Synaptic	Any	12.50000000	5.000	1.000	18.5
Nirvanix	Any	12.50000000	5.000	1.500	19
Nirvanix	Any	12.50000000	10.000	1.500	24
Nirvanix	Any	12.50000000	15.000	1.500	29

Figure 6.14: Storage and network services recommendations for the business analyst selection use case.

column). "Any" means the provider offers the same price for all of its locations. In the case of SoftLayer, it charges the same price in all locations. In the case of AWS, since it offers different prices for different locations, the enterprise may be able to consume the same service with a lower price in a different location. The base currency is USD, but since in the call the analyst has specified "currency=AUD", the result shown is in AUD accurate to one decimal place.

Figure 6.15 shows another example which selects compute, storage, and network using the RESTful API. The selection criteria include 1 compute service instance (shown as "n=1"): 0<Ram<=69, 0<=local storage<=2040, number of hours to be used per month 744. The selection results are displayed at the end of the figure.

Due to high inter-cloud data transfer cost overhead and communication delay, our recommender logic does not consider the combination of services from multiple providers. For example, the CloudRecommender will not select and combine compute services from Amazon with storage services from Azure. Similarly, some provider charge for data transfer across their services that are hosted at different locations. For example, data transferred between Amazon services in different locations are charged as “Internet Data Transfer” on both sides of the transfer. We currently choose to put all services at the same location. In future (if necessary) we may extend our recommendation logic to allow users to choose between different locations for each service type. Additionally, providers often offer a discounted price for higher usage, and keeping all data together means higher usage, which can consume a cheaper price tier. Network services are always bundled with either compute or storage services as it is impossible to consume other services without incurring network costs.

6.3.4 Case Studies

Gaia is a global space astrometry mission to make the largest, most precise three-dimensional map of our Galaxy by surveying more than one billion stars. For the number of images produced by the satellite (1 billion stars $\times$ 80 observations $\times$ 10

Call

localhost/cloud_demo_1_1/api/cost/combined?
media_type=xml&currency=AUD&storage=10&duration=30&data_upload_size=1&data_download_size=1&ram_range=0%2C69&storage_range=0%2C204
0&hour=720&n=1

Parameter	Value	Type	Description
media_type	xml	string	The format of response. Allowed values are json and xml.
currency	AUD	string	The currency of response. Available currencies: AED, AFN, ALL, AMD, ANG, AOA, ARS, AUD, AWG, AZN, BAM, BBD, BDT, BGN, BHD, BIF, BMD, BND, BOB, BRL, BSD, BTN, BWP, BYR, BZD, CAD, CDF, CHF, CLF, CLP, CNH, CNY, COP, CRC, CUP, CVE, CZK, DJF, DKK, DOP, DZD, EGP, ETB, EUR, FJD, FKP, GBP, GEL, GHS, GIP, GMD, GNF, GTQ, GYD, HKD, HNL, HRK, HTG, HUF, IDR, IEP, ILS, INR, IQD, IRR, ISK, JMD, JOD, JPY, KES, KGS, KHR, KMF, KPW, KRW, KWD, KZT, LAK, LBP, LKR, LRD, LSL, LTL, LVL, LYD, MAD, MDL, MGA, MKD, MMK, MNT, MOP, MRO, MUR, MVR, MWK, MXN, MYR, MZN, NAD, NGN, NIO, NOK, NPR, NZD, OMR, PAB, PEN, PGK, PHP, PKR, PLN, PYG, QAR, RON, RSD, RUB, RWF, SAR, SBD, SCR, SDG, SEK, SGD, SHP, SLL, SOS, SRD, STD, SVC, SYP, SZL, THB, TJS, TMT, TND, TOP, TRY, TTD, TWD, TZS, UAH, UGX, USD, UYU, UZS, VEF, VND, VUV, WST, XAF, XCD, XDR, XOF, XPF, YER, ZAR, ZMK, ZWL.
storage	10	int	Specify the estimated usage (in GB). Bad request exception will be thrown if this parameter is missing or invalid.
duration	30	int	Specify the service usage duration (in calendar days).
data_upload_size	2	int	Specify the estimated usage (in GB).
data_download_size	3	int	Specify the estimated usage (in GB).
ram_range	0,69	string	Similar to the provider_id parameter. Format should be: range_low,rang_high;range_low,rang_high Multiple values should be separated by semi colon, value pair should be separated by comma.
storage_range	0,2040	string	Similar to the ram_range parameter. Specifies the local storage requirements. Format should be: range_low,rang_high;range_low,rang_high Multiple values should be separated by semi colon, value pair should be separated by comma.
hour	744	string	Similar to the provider parameter. Specifies the number of hours expected to use. Format should be: hour,hour Multiple values should be separated by comma.
month		string	Similar to the hour parameter. Specifies the number of months expected to use. Format should be: month,month Multiple values should be separated by comma.
n	1	string	Similar to the hour parameter. Specifies the number of instances expected to use. Format should be: n,n Multiple values should be separated by comma. If this and the above 4 parameter have multiple values, they should match with each other.

Filter: Fetched 28 records. Duration: 0.001 sec, fetched in: 0.000 sec

Provider Name	region_name	storage_cost	requests_cost	data_transfer_cost	choices_compute_total_cost	compute_storage_dataTransfer_cost
Amazon	Asia Pacific(Tokyo)	1.36500000	0.0000	0.402	20.0880000000	21.855
Amazon	Asia Pacific(Tokyo)	1.36500000	0.0000	0.402	26.0400000000	27.807
Windows Azure	North America and Europe	1.40000000	0.0000	0.36	37.2000000000	38.96
Windows Azure	Asia Pacific Region	1.40000000	0.0000	0.57	37.2000000000	39.17

Figure 6.15: Result of selection process for Compute, Storage and network services

readouts), if it took one millisecond to process one image, it would take 30 years of data processing time on a single processor. Luckily the data does not need to be processed continuously. Every six months, they need to process all the observations within two weeks of [197]. Hypothetically speaking, if they choose to use 120 high CPU and memory VMs. The example search via CloudRecommender is shown in Figure 6.16. With each VM running 12 threads, 1440 processes were working in parallel. This will reduce the processing time to less than 200 hours (about a week).

In this case, since data can be moved into/out of the cloud in bulk periodically, FedEx hard drive may be preferred over transferring data over the internet. Promotional offers may not matter much in this case compared to the considerable time and capital investment savings, but it makes a big difference for small businesses or startups.

Another example usage is sites with sizeable continuous data input and processing need like Yelp. Everyday Yelp generates and stores around 100GB of logs and photos, and runs approximately 200 MapReduce jobs and processing 3TB of data [228]. Yelp.com has more than 71 million monthly unique visitors [61]. The average page size of a typical website is about 784 kB [211]. So the estimated data download

Figure 6.16: Example input parameter values.

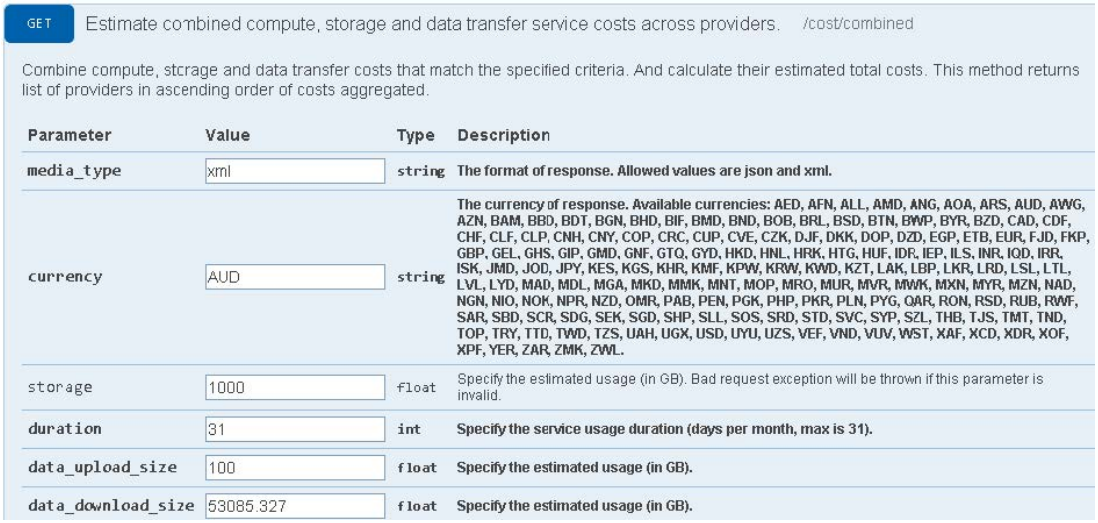


Figure 6.17: Example parameters for REST API.

traffic is about 51TB per month if every unique user only views one page once a month. Figure 6.17 shows a sample search for the scenario, as mentioned above.

6.4 Summary

This chapter discussed the implementation details and evaluation of the techniques we developed in Chapters 3 and 4. Specifically, for CoCoOn proposed in Chapter 3, we illustrated how to use CoCoOn to represent the semantic information. We collected data from the websites and APIs of various Cloud service providers, including Google Cloud and Microsoft Azure. We mapped the data to CoCoOn, to create an RDF dataset which consists of 132,282 triples. We have made the dataset available on Github [7]. To evaluate the AHP based QoS-aware technique proposed in Chapter 4, we developed two tools in this chapter: one is the QoS profiler which collects the end to end QoS statistics of Cloud services and the other is CloudRecommender which ranks Cloud service according to user preferences. We discussed the system architecture, user interface and API of these tools - QoS profiler and CloudRecommender, and illustrated how the AHP based QoS-aware technique can be applied to select

[7] https://github.com/miranda-zhang/cloud-computing-schema/blob/ldf-server/v1.0.1/v1_0_1.ttl

Cloud services in these tools.

Chapter 7

Conclusions and Future Directions

In this chapter, we conclude the contributions of this thesis and discuss future directions in this area. This thesis has mainly addressed the problem of automatic Cloud IaaS service discovery and comparison. It has also explored ways to forecast cloud usage in order to estimate cost, thus providing optimal Cloud deployment options, and answer questions like which provider to choose? What is the total cost? What kind of servers to use? How many servers are needed? What expected average network performance would there be?

Although elasticity, cost benefits and abundance of resources motivate many organizations to migrate their enterprise applications to the Cloud, end-users (e.g., CIOs, scientists, developers, engineers, etc.) are faced with the complexity of choosing the right set of Cloud services for deploying their applications. Manually reading Cloud providers' documentation to find out which services are suitable for building their Cloud-based applications is a cumbersome task for decision makers. The multi-layered organizations (e.g., SaaS, PaaS, and IaaS) of Cloud services, along with their various types and features, make the task of service identification a hard problem. For example, "EC2 instances", "virtual cloud servers", "compute cloud servers" and "DigitalOcean droplets" refer to the same thing by different providers. Similarly, "S3", "Cloud files" and "object store" all refer to storage. In addition to the actual price differences for similar services among various providers, there is a range of pricing models for how services are charged with each provider, for example, pay-as-you-go, spot instance or bidding, two-part tariff, block-declining, free for a period and discount with bulk buy. So, to deal with the complexity of choosing from a large number of different cloud services from diverse providers, end-users need access to a specialized intermediary which can act as a "one-stop-shop" for procuring and comparing Cloud services.

We have addressed the problem as mentioned above by introducing many techniques with their implementation to form a Cloud recommendation tool-set, which simplifies the Cloud service selection. It allows multi-criteria searches on infrastructure cloud offers across different cloud providers. It is a semi-automated approach which supports the network-QoS-aware selection of Cloud services, along with a unified domain model that is capable of fully describing infrastructure services in Cloud computing. This approach takes account of real-time and variable network QoS constraints, and applies a utility function that combines multiple selection crite-

ria (e.g., total cost, the maximum size limit for storage, and memory size for servers) about storage, compute, and network services.

The Cloud recommendation tool-set is expected to be beneficial to end-users, who are:

1. considering migrating their applications to Cloud services for cost savings.

2. hard-pressed to understand the cost and benefits of moving to the Cloud, especially when the market evolves so rapidly.

7.1 Contributions

The Cloud has great potential for a large variety of users with diverse needs, but the selection of a right provider is crucial to this end. Aiming to eliminate potential bottlenecks that limit the ability of general users to take advantage of Cloud computing, we present a comprehensive solution set, which allows users to make multi-criteria selection and comparison on IaaS offer considering QoS requirements.

We have proposed an ontology, named CoCoOn, for classifying and representing the configuration information related to Cloud-based IaaS services including compute, storage, and network. The proposed ontology is comprehensive as it captures both static configurations and dynamic QoS configurations at the IaaS layer. We also implemented relevant tools for CoCoOn, which enables automatic linking of data from different providers as well as external domains, e.g. Geo Locations, Units, Business Service (i.e. GoodRelations). Thus, unify the process of selection and comparison Cloud IaaS Services. This work will also help readers in understanding the core concepts and inter-relationship between different IaaS-level Cloud computing service types. This, in turn, may lead to a harmonization of research efforts and more inter-operable Cloud technologies and services at the IaaS layer.

Additionally, we have developed an AHP-based decision making and service selection technique that handles multiple quantitative (i.e., numeric) and qualitative criteria. Qualitative criteria are descriptive and non-numeric criteria, such as location, CPU architecture (i.e. a 32-bit or 64-bit operating system). The AHP method determines the relative importance of criteria to each user by conducting pairwise comparisons.

We have also developed a model for Cloud resource usage estimation, based on benchmarking and performance modelling. We have applied the queueing theory to model the performance of a system. We have formulated the questions of minimizing cost and maximizing performance as constraint satisfaction problems, which can be solved by MIP methods.

Furthermore, we have proposed a system, named CloudRecommender, that transforms the cloud service configuration selection from an ad-hoc process that involves manually reading the provider documentation to a process that is structured, and to a large extent, automated. This implementation also includes a generic service that helps in collecting network QoS values from different points on the Internet

(modelling a big data source location) to the Cloud data centres. Although we believe that CloudRecommender leaves room for further enhancements, yet provides a practical approach. We have implemented a prototype of CloudRecommender and evaluated it with some example selection scenarios. The prototype demonstrates the feasibility of the CloudRecommender design and its practical aspects.

7.2 Reflection and Future Work

Many possible extensions can be made to CoCoOn. More providers should be included to verify the completeness of our model further; Units can be improved with Custom Datatypes (`cdt:ucum` [116]), so composite units do not need to be defined specifically, i.e. instead of cocoon:MegabitsPerSecondPerHour, something like "MB/s/h" could be used. Various discount offers can also be better supported, such as reserved instance, unlimited mode [57] and on-demand capacity reservation [58] pricing models. To clarify, spot instances can be modelled in the same way as on-demand price instances. The free tier can be modelled with gr:hasEligibleQuantity to set the limit one can use for free. Reserved instance offers can make use of gr:eligibleDuration to model reservation period, e.g. 1-year terms, but additional properties and classes need to be defined to model upfront costs, CPU credits, etc. Moreover, we have mainly focused on the IaaS layer, which is fundamental to the other higher-level layers. A possible future work would be to cover the PaaS and the SaaS layers. In terms of system implementation, we could migrate the infrastructure services definitions to an RDF database and use SPIN templates [190] to encode our procedures in SPARQL [188].

It would also be helpful to capture the dependencies of services across different layers. For example, before mapping a MySQL database appliance to an Amazon EC2 compute service at the IaaS layer, one needs to consider whether they are compatible. Furthermore, SLA and legal compliance [137] information can also be investigated for inclusion into the decision making the process.

Another avenue that would be good to explore is the spot instance bidding/auction market. How to take advantage of the service brokerage [162, 161] market can be investigated further.

At a higher level, there are two parts to the Cloud service selection problem: knowledge extraction [63] and decision making. Under knowledge extraction, the subfield of ontology learning [106, 222] covers research on the automatic or semi-automatic creation of ontologies. Advances in machine learning techniques may promote better ways to solve this problem.

Other decision making techniques has been explored in recent works include collaborative filtering recommendation techniques [127], TOPSIS [230, 199, 127], fuzzy methods [93, 96, 226], neural network [70], ELECTRE [123] and best-worst method (BWM) [171, 113, 230]. This work did not consider collaborative filtering techniques because of the impracticality of acquiring user input for experimentation. This work did not consider TOPSIS, becasue it can suffer from ranking abnormality, see section 2.3.1.4. Future research should compare AHP methods with fuzzy, neural net-

work, ELECTRE and BWM based methods.

Bibliography

[1] *13,000 Projects Ditched GitHub for GitLab Monday Morning - VICE*. URL: https://www.vice.com/en%7B%5C_%7Dus/article/ywen8x/13000-projects-ditched-github-for-gitlab-monday-morning?utm%7B%5C_%7Dmedium=referral%7B%5C&%7Dutm%7B%5C_%7Dsource=quora (visited on 06/05/2019) (cit. on p. 6).

[2] *175 Zettabytes By 2025*. URL: https://www.forbes.com/sites/tomcoughlin/2018/11/27/175-zettabytes-by-2025/%7B%5C#%7D24af56c25459 (visited on 06/17/2019) (cit. on p. 76).

[3] *ab - Apache HTTP server benchmarking tool - Apache HTTP Server Version 2.4*. URL: https://httpd.apache.org/docs/2.4/programs/ab.html (visited on 06/23/2019) (cit. on p. 110).

[4] Y. M. Afify et al. "A semantic-based Software-as-a-Service (SaaS) discovery and selection system". In: *2013 8th International Conference on Computer Engineering & Systems (ICCES)*. IEEE, Nov. 2013, pp. 57–63. ISBN: 978-1-4799-0080-0. DOI: 10.1109/ICCES.2013.6707171. URL: http://ieeexplore.ieee.org/document/6707171/ (cit. on pp. 23, 24).

[5] Ali Afshari, M Mojahed, and Rosnah Yusuff. "Simple Additive Weighting Approach to Personnel Selection Problem." In: *International Journal of Innovation, Management and Technology* 1 (Dec. 2010), pp. 511–515. DOI: 10.7763/IJIMT.2010.V1.89. URL: https://www.researchgate.net/publication/285828294_Simple_Additive_Weighting_Approach_to_Personnel_Selection_Problem (cit. on p. 34).

[6] *Agner Krarup Erlang Wikipedia*. URL: https://en.wikipedia.org/wiki/Agner_Krarup_Erlang (visited on 05/10/2019) (cit. on pp. 44, 104).

[7] *Alibaba Cloud*. URL: https://www.alibabacloud.com/ (visited on 09/13/2017) (cit. on p. 2).

[8] Asma Alkalbani et al. "Design and Implementation of the Hadoop-Based Crawler for SaaS Service Discovery". In: *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*. IEEE, Mar. 2015, pp. 785–790. ISBN: 978-1-4799-7905-9. DOI: 10.1109/AINA.2015.268. URL: http://ieeexplore.ieee.org/document/7098053/ (cit. on p. 31).

[9] *Amazon EC2*. URL: http://aws.amazon.com/ec2/instance-types/ (visited on 10/18/2018) (cit. on p. 4).

[10] *Amazon Web Services (AWS) - Cloud Computing Services*. URL: https://aws.amazon.com/ (visited on 09/15/2017) (cit. on p. 2).

[11] *Analytic Hierarchy Process AHP - Business Performance Management - YouTube*. URL: https://www.youtube.com/watch?v=18GWVtVAAzs (visited on 06/11/2019) (cit. on p. 37).

[12] *Apache Hadoop*. URL: https://en.wikipedia.org/wiki/Apache_Hadoop (cit. on p. 31).

[13] *Apache Jena - Reasoners and rule engines: Jena inference support*. URL: https://jena.apache.org/documentation/inference/ (visited on 03/12/2018) (cit. on pp. 25, 26).

[14] *Apache Solr*. URL: https://en.wikipedia.org/wiki/Apache_Solr (cit. on p. 31).

[15] Michael Armbrust et al. "A View of Cloud Computing". In: *Commun. ACM* 53.4 (Apr. 2010), pp. 50–58. ISSN: 0001-0782. DOI: 10.1145/1721654.1721672. URL: http://doi.acm.org/10.1145/1721654.1721672 (cit. on p. 2).

[16] *AWS | Amazon Simple Storage Service (S3) - Online Cloud Storage for Data & Files*. http://aws.amazon.com/s3/. Accessed: 2015-Feb-05 (cit. on p. 84).

[17] *AWS Simple Monthly Calculator*. URL: https://calculator.s3.amazonaws.com/index.html (visited on 09/18/2017) (cit. on p. 32).

[18] Ardalan Bafahm and Minghe Sun. "Some Conflicting Results in the Analytic Hierarchy Process". In: *International Journal of Information Technology & Decision Making* 18.02 (Mar. 2019), pp. 465–486. ISSN: 0219-6220. DOI: 10.1142/S0219622018500517. URL: https://www.worldscientific.com/doi/abs/10.1142/S0219622018500517 (cit. on pp. 39, 87).

[19] Vasiliki Balioti, Christos Tzimopoulos, and Chris Evangelides. "Multi-Criteria Decision Making Using TOPSIS Method Under Fuzzy Environment. Application in Spillway Selection". In: *Proceedings* 2 (July 2018), p. 637. DOI: 10.3390/proceedings2110637. URL: https://www.researchgate.net/publication/326738995_Multi-Criteria_Decision_Making_Using_TOPSIS_Method_Under_Fuzzy_Environment_Application_in_Spillway_Selection (cit. on p. 39).

[20] Paul Barham et al. *Xen and the Art of Virtualization*. Tech. rep. 2003. URL: http://www.cs.yale.edu/homes/yu-minlan/teach/csci599-fall12/papers/xen.pdf (cit. on p. 2).

[21] Michael Bedford Taylor. "Bitcoin and the age of bespoke silicon". In: *Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013 International Conference on*. IEEE. 2013, pp. 1–10 (cit. on p. 77).

[22] Tom Beigbeder et al. "The effects of loss and latency on user performance in unreal tournament 2003®". In: *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. ACM. 2004, pp. 144–151 (cit. on p. 75).

[23] Valerie Belton and Tony Gear. "On a short-coming of Saaty's method of analytic hierarchies". In: *Omega* 11.3 (Jan. 1983), pp. 228–230. ISSN: 0305-0483. DOI: 10.1016/0305-0483(83)90047-6. URL: https://www.sciencedirect.com/science/article/pii/0305048383900476 (cit. on p. 37).

[24] B. Benatallah et al. "QoS-aware middleware for Web services composition". In: *IEEE Transactions on Software Engineering* 30.5 (May 2004), pp. 311–327. ISSN: 0098-5589. DOI: 10.1109/TSE.2004.11. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1291834 (cit. on p. 73).

[25] *Big data*. URL: https://en.wikipedia.org/wiki/Big_data (visited on 06/17/2019) (cit. on p. 77).

[26] *Big Data: Where we at?* https://www.centrodeinnovacionbbva.com/en/magazines/innovation-edge/publications/20-big-data/posts/147-big-data-where-we-at (cit. on p. 77).

[27] Khouloud Boukadi et al. "Cloud Service Description Ontology : Construction , Evaluation and Interrogation". In: *Cloud Service Description Ontology : Construction , Evaluation and Interrogation*. 2016. URL: https://www.semanticscholar.org/paper/Cloud-Service-Description-Ontology-%3A-Construction-%2C-Boukadi-Rekik/5a19b9a60fc91118059784e60db43d5f0a246204?tab=abstract (cit. on pp. 17–19).

[28] Alexander Brodsky et al. "A decisions query language (DQL)". In: *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09*. New York, New York, USA: ACM Press, June 2009, p. 1059. ISBN: 9781605585512. DOI: 10.1145/1559845.1559981. URL: http://dl.acm.org/citation.cfm?id=1559845.1559981 (cit. on p. 120).

[29] *Burstorm Platform*. URL: https://www.burstorm.com/platform/ (visited on 09/15/2017) (cit. on pp. 6, 73).

[30] Don Caldwell et al. "The cutting EDGE of IP router configuration". In: *ACM SIGCOMM Computer Communication Review* 34.1 (Jan. 2004), p. 21. ISSN: 01464833. DOI: 10.1145/972374.972379. URL: http://dl.acm.org/citation.cfm?id=972374.972379 (cit. on p. 120).

[31] R.N. Calheiros, R. Ranjan, and R. Buyya. "Virtual Machine Provisioning Based on Analytical Performance and QoS in Cloud Computing Environments". In: *Parallel Processing (ICPP), 2011 International Conference on*. Sept. 2011, pp. 295–304. DOI: 10.1109/ICPP.2011.17 (cit. on pp. 9, 44).

[32] C. Chang, P. Liu, and J. Wu. "Probability-Based Cloud Storage Providers Selection Algorithms with Maximum Availability". In: *2012 41st International Conference on Parallel Processing*. Sept. 2012, pp. 199–208. DOI: 10.1109/ICPP.2012.51 (cit. on p. 41).

[33] Yue-Shan Chang et al. "Integrating intelligent agent and ontology for services discovery on cloud environment". In: *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, Oct. 2012, pp. 3215–3220. ISBN: 978-1-4673-1714-6. DOI: 10.1109/ICSMC.2012.6378286. URL: http://ieeexplore.ieee.org/document/6378286/ (cit. on p. 25).

[34] Fei Chen, Xiaoli Bai, and Bingbing Liu. "Efficient Service Discovery for Cloud Computing Environments". In: Springer, Berlin, Heidelberg, 2011, pp. 443–448. DOI: 10.1007/978-3-642-21411-0_72. URL: http://link.springer.com/10.1007/978-3-642-21411-0_72 (cit. on p. 32).

[35] Xu Chen et al. "Declarative configuration management for complex and dynamic networks". In: *Proceedings of the 6th International COnference on - Co-NEXT '10*. New York, New York, USA: ACM Press, Nov. 2010, p. 1. ISBN: 9781450304481. DOI: 10.1145/1921168.1921176. URL: http://dl.acm.org/citation.cfm?id=1921168.1921176 (cit. on p. 120).

[36] Sergei Chichin. "An Open Market for Trading Cloud Services". PhD thesis. 2016. URL: https://researchbank.swinburne.edu.au/file/8cfa0100-3af0-442a-9c17-c077a828ec90/1/Sergi%20Chichin%20Thesis.pdf (cit. on p. 78).

[37] *Chinese Idioms / Chengyu*. URL: http://www.standardmandarin.com/idiom/almost-everything-has-a-start-but-not-many-things-have-an-end-idiom-dont-start-sth-you-cant-handle (visited on 03/18/2018) (cit. on p. vi).

[38] *ClarkNet-HTTP - Two Weeks of HTTP Logs from the ClarkNet WWW Server*. URL: https://github.com/aespinosa/webserver-invariants (visited on 06/21/2019) (cit. on p. 108).

[39] *Cloud Calculator Rackspace Australia*. URL: https://www.rackspace.com/en-au/calculator (visited on 09/18/2017) (cit. on p. 32).

[40] *Cloud Network Test | CloudHarmony*. URL: https://cloudharmony.com/speedtest (visited on 09/16/2017) (cit. on pp. 9, 77).

[41] *Cloudorado*. URL: https://www.cloudorado.com/ (visited on 09/18/2017) (cit. on pp. 33, 77).

[42] *Home - Cmap*. URL: https://cmap.ihmc.us/ (visited on 08/13/2018) (cit. on p. 51).

[43] *CmapTools Ontology Editor*. URL: http://cmap.ihmc.us/coe/test/ (visited on 03/14/2018) (cit. on p. 51).

[44] *Community:ListPatterns - Odp*. URL: http://ontologydesignpatterns.org/wiki/Community:ListPatterns (visited on 06/27/2018) (cit. on p. 12).

[45] Michael Compton et al. "The SSN ontology of the W3C semantic sensor network incubator group". In: *J. Web Semant.* 17 (2012), pp. 25–32. URL: https://www.sciencedirect.com/science/article/pii/S1570826812000571 (cit. on p. 8).

[46] *Cryptocurrency*. http://en.wikipedia.org/wiki/Cryptocurrency. Accessed: 25-Mar-2014 (cit. on p. 77).

[47] *Custom Datatypes*. URL: https://ci.mines-stetienne.fr/lindt/v2/custom_datatypes.html (visited on 01/24/2019) (cit. on p. 57).

[48] A. V. Dastjerdi, S. G. H. Tabatabaei, and R. Buyya. "An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery". In: *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. May 2010, pp. 104–112. DOI: 10.1109/CCGRID.2010.87 (cit. on p. 50).

[49] *Data access fees on local disk for Azure VM*. URL: https://www.rhipe.com/azure-storage-transactions/ (visited on 10/10/2018) (cit. on pp. 59, 65).

[50] *DataCentred Public Cloud Pricing*. URL: https://www.datacentred.co.uk/pricing/ (visited on 09/05/2017) (cit. on p. 2).

[51] Manuel Parra-Royon and J.M. Benítez. *DataMining-LD*. URL: http://cookingbigdata.com/linkeddata/dmservices/ (visited on 08/10/2018) (cit. on pp. 17, 18).

[52] *DCMI: DCMI Metadata Terms*. URL: http://dublincore.org/documents/dcmi-terms/ (visited on 08/13/2018) (cit. on p. 50).

[53] *difference between AHP and TOPSIS*. URL: https://www.researchgate.net/post/What_are_the_basic_and_technical_difference_between_AHP_and_TOPSIS (cit. on p. 79).

[54] *Dion Hinchcliffe's Next-Generation Enterprises*. URL: http://www.ebizq.net/blogs/enterprise/ (visited on 05/10/2019) (cit. on p. 5).

[55] G. Dobson, R. Lock, and I. Sommerville. "QoSOnt: a QoS Ontology for Service-Centric Systems". English. In: *31st EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2005, pp. 80–87. ISBN: 0-7695-2431-1. DOI: 10.1109/EUROMICRO.2005.49. URL: https://ieeexplore.ieee.org/document/1517730 (cit. on pp. 28, 30, 53, 66).

[56] Yucheng Dong et al. "Consensus models for AHP group decision making under row geometric mean prioritization method". In: *Decision Support Systems* 49.3 (June 2010), pp. 281–289. ISSN: 0167-9236. DOI: 10.1016/J.DSS.2010.03.003. URL: https://www.sciencedirect.com/science/article/pii/S0167923610000643 (cit. on p. 87).

[57] *Unlimited Mode for Burstable Performance Instances*. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances-unlimited-mode.html (visited on 06/25/2019) (cit. on p. 137).

[58] *On-Demand Capacity Reservation*. URL: https://aws.amazon.com/ec2/faqs/#On-Demand_Capacity_Reservation (visited on 06/25/2019) (cit. on p. 137).

[59] *Erlang C's formula*. URL: https://en.wikipedia.org/wiki/Erlang_(unit) (visited on 06/25/2019) (cit. on p. 104).

[60] *ESA Future Missions - Earthnet Online*. URL: https://earth.esa.int/web/guest/missions/esa-future-missions#_56_INSTANCE_hH2r_matmp (cit. on p. 77).

[61] *Factsheet | Yelp*. URL: https://www.yelp.com.au/factsheet (visited on 10/19/2018) (cit. on p. 132).

[62] *FOAF*. URL: https://en.wikipedia.org/wiki/FOAF_(ontology) (cit. on p. 50).

[63] Aldo Gangemi. "A Comparison of Knowledge Extraction Tools for the Semantic Web". In: *The Semantic Web: Semantics and Big Data*. Ed. by Philipp Cimiano et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 351–366. ISBN: 978-3-642-38288-8 (cit. on p. 137).

[64] *Ganglia Monitoring System*. URL: http://ganglia.info/ (visited on 06/21/2019) (cit. on p. 108).

[65] *Ganglia:: Wikimedia Grid Report*. URL: http://ganglia.wikimedia.org/latest/ (visited on 03/12/2015) (cit. on p. 108).

[66] Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. "SMICloud: A Framework for Comparing and Ranking Cloud Services". In: (2011). DOI: 10.1109/UCC.2011.36. URL: http://www.cloudbus.org/papers/SMICloud2011.pdf (cit. on pp. 11, 36).

[67] Vyas S Gayatri and Misal S Chetan. "Comparative Study of Different Multicriteria Decision-making Methods". In: *ISSN (Print* 24 (2013), pp. 2319–2526. URL: http://www.irdindia.in/journal_ijacte/pdf/vol2_iss4/2.pdf (cit. on pp. 33, 36, 39).

[68] *GeoNames*. URL: https://www.geonames.org/ (visited on 02/14/2019) (cit. on p. 68).

[69] Seyed Hassan Ghodsypour and C O'brien. "A decision support system for supplier selection using an integrated analytic hierarchy process and linear programming". In: *International journal of production economics* 56 (1998), pp. 199–212 (cit. on p. 84).

[70] O. Gireesha et al. "WNN-EDAS: A Wavelet Neural Network Based Multicriteria Decision-Making Approach for Cloud Service Selection". In: 2020 (cit. on p. 137).

[71] Alfio Gliozzo et al. *Semantic Technologies in IBM Watson TM*. Tech. rep. IBM, 2013. URL: http://www.patwardhans.net/papers/GliozzoBPM13.pdf (cit. on p. 12).

[72] Manish Godse and Shrikant Mulik. "An Approach for Selecting Software-as-a-Service (SaaS) Product". In: (2009). DOI: 10.1109/CLOUD.2009.74. URL: http://barbie.uta.edu/%7B~%7Dhdfeng/CloudComputing/cloud/cloud29.pdf (cit. on p. 36).

[73] *Google Cloud Computing, Hosting Services & APIs | Google Cloud Platform*. URL: https://cloud.google.com/ (visited on 09/15/2017) (cit. on p. 2).

[74] *Google Cloud Platform Pricing Calculator*. URL: https://cloud.google.com/products/calculator/ (visited on 09/18/2017) (cit. on p. 32).

[75] *Gcloud price*. URL: https://cloud.google.com/compute/pricing (visited on 10/10/2018) (cit. on p. 64).

[76] *Google stabs Wikipedia in the front • The Register*. URL: https://en.wikipedia.org/wiki/The_Register (visited on 06/09/2019) (cit. on p. 13).

[77] Andrzej Goscinski and Michael Brock. "Toward dynamic and attribute based publication, discovery and selection for cloud computing". In: *Future Generation Computer Systems* 26.7 (July 2010), pp. 947–970. ISSN: 0167739X. DOI: 10.1016/j.future.2010.03.009. URL: http://linkinghub.elsevier.com/retrieve/pii/S0167739X10000543 (cit. on p. 32).

[78] D Gross and C Harris. *Fundamentals of queueing theory*. Wiley, 1998, pp. 244–247. ISBN: 0-471-17083-6 (cit. on pp. 44, 101).

[79] Thomas R. Gruber. "Toward principles for the design of ontologies used for knowledge sharing?" In: *International Journal of Human-Computer Studies* 43.5-6 (Nov. 1995), pp. 907–928. ISSN: 1071-5819. DOI: 10.1006/IJHC.1995.1081. URL: https://www.sciencedirect.com/science/article/pii/S1071581985710816 (cit. on p. 57).

[80] R. V. Guha, Dan Brickley, and Steve MacBeth. "Schema.Org: Evolution of Structured Data on the Web". In: *Queue* 13.9 (Nov. 2015), 10:10–10:37. ISSN: 1542-7730 (cit. on p. 58).

[81] Ranier Haas and Oliver Meixner. "An illustrated guide to the analytic hierarchy process". In: *Institute of Marketing & Innovation, University of Natural Resources and Applied Life Sciences, Vienna* (2005), pp. 10–13 (cit. on p. 84).

[82] Armin Haller et al. "The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation". In: *Semantic Web* 10.1 (2019), pp. 9–32. URL: http://www.semantic-web-journal.net/content/sosassn-ontology-joint-w3c-and-ogc-standard-specifying-semantics-sensors-observations (cit. on pp. 8, 53, 66).

[83] Armin Haller et al. "WSMX - A Semantic Service-Oriented Architecture". In: *ICWS*. 2005, pp. 321–328 (cit. on pp. 18, 50).

[84] Saouli Hamza et al. "A Cloud computing approach based on mobile agents for Web services discovery". In: *Second International Conference on the Innovative Computing Technology (INTECH 2012)*. IEEE, Sept. 2012, pp. 297–304. ISBN: 978-1-4673-2679-7. DOI: 10.1109/INTECH.2012.6457745. URL: http://ieeexplore.ieee.org/document/6457745/ (cit. on p. 31).

[85] *Heavy traffic approximation*. URL: https://en.wikipedia.org/wiki/Heavy_traffic_approximation (visited on 06/09/2019) (cit. on pp. 48, 105).

[86] *Heavy traffic approximation*. URL: https://en.wikipedia.org/wiki/Heavy_traffic_approximation (visited on 06/23/2019) (cit. on p. 105).

[87] Martin Hepp. "GoodRelations: An Ontology for Describing Products and Services Offers on the Web". In: *EKAW*. 2008, pp. 329–346 (cit. on p. 63).

[88] Anthony JG Hey and Anne E Trefethen. "The data deluge: An e-science perspective". In: (2003) (cit. on p. 77).

[89] *Hitachi Consulting*. URL: https://www.hitachiconsulting.com/ (visited on 06/04/2019) (cit. on p. 1).

[90] *home | NeCTAR*. https://nectar.org.au/. Accessed: 05-May-2014 (cit. on p. 93).

[91] Ian Horrocks. "Description Logic: A Formal Foundation for Ontology Languages and Tools Part 2". In: (). URL: http://www.cs.ox.ac.uk/ian.horrocks/Seminars/download/oracle-seminar-20101210.pdf (cit. on p. 12).

[92] *HtmlUnit:GUI-Less browser for Java programs*. http://htmlunit.sourceforge.net/. Accessed: 20-Nov-2014 (cit. on p. 118).

[93] A. Hussain, Jin Chun, and Maria Khan. "A novel framework towards viable Cloud Service Selection as a Service (CSSaaS) under a fuzzy environment". In: *Future Gener. Comput. Syst.* 104 (2020), pp. 74–91 (cit. on p. 137).

[94] Iaeng. "An Ontology-enhanced Cloud Service Discovery System". In: *Proceedings of the international multi conference of engineers and computer scientists (IMECS), HongKong*. 2010. ISBN: 9789881701282. URL: https://pdfs.semanticscholar.org/c402/c041057a04f57bef423435026cb920b2998a.pdf (cit. on p. 34).

[95] *IBM Cloud*. URL: http://www.softlayer.com/cloud-servers (visited on 09/18/2017) (cit. on p. 2).

[96] Galina Ilieva et al. "Cloud Service Selection as a Fuzzy Multi-criteria Problem". In: *TEM Journal* (2020), pp. 484–495 (cit. on p. 137).

[97] A Ishizaka, D Balkenborg, and T Kaplan. "Influence of aggregation and measurement scale on ranking a compromise alternative in AHP". In: *Journal of the Operational Research Society* 62.4 (Apr. 2011), pp. 700–710. ISSN: 0160-5682. DOI: 10.1057/jors.2010.23. URL: https://www.tandfonline.com/doi/full/10.1057/jors.2010.23 (cit. on p. 87).

[98] M.C. Jaeger, G. Muhl, and S. Golze. "QoS-aware composition of Web services: a look at selection algorithms". English. In: *IEEE International Conference on Web Services (ICWS'05)*. IEEE, 2005, p. 808. ISBN: 0-7695-2409-5. DOI: 10.1109/ICWS.2005.95. URL: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1530884%20http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1530884 (cit. on p. 75).

[99] Krzysztof Janowicz et al. "SOSA: A Lightweight Ontology for Sensors, Observations, Samples, and Actuators". In: *CoRR* abs/1805.09979 (2018) (cit. on pp. 58, 66).

[100] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated gain-based evaluation of IR techniques". In: *ACM Transactions on Information Systems (TOIS)* 20.4 (2002), pp. 422–446. URL: https://dl.acm.org/citation.cfm?id=582418 (visited on 05/10/2019) (cit. on p. 43).

[101] *JMeter and Performance Testing for DevOps | BlazeMeter*. URL: https://blazemeter.com/ (visited on 06/23/2019) (cit. on p. 111).

[102] Karuna Pande Joshi, Yelena Yesha, and Timothy W. Finin. "Automating Cloud Services Life Cycle through Semantic Technologies". In: *IEEE Transactions on Services Computing* 7 (2014), pp. 109–122. URL: https://www.semanticscholar.org/paper/Automating-Cloud-Services-Life-Cycle-through-Joshi-Yesha/1dea78031a3f6c3a8b4d9bb98940e69b615486bb (cit. on pp. 17, 20, 23).

[103] *JSON-LD*. URL: https://en.wikipedia.org/wiki/JSON-LD (cit. on p. 13).

[104] Jaeyong Kang and Kwang Mong Sim. "Cloudle: A Multi-criteria Cloud Service Search Engine". In: *2010 IEEE Asia-Pacific Services Computing Conference*. IEEE, Dec. 2010, pp. 339–346. ISBN: 978-1-4244-9396-8. DOI: 10.1109/APSCC.2010.44. URL: http://ieeexplore.ieee.org/document/5708589/ (cit. on pp. 22, 24).

[105] Raed Karim, Chen Ding, and Ali Miri. "An End-to-End QoS Mapping Approach for Cloud Service Selection". In: *Proceedings of the 2013 IEEE Ninth World Congress on Services*. SERVICES '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 341–348. ISBN: 978-0-7695-5024-4. DOI: 10.1109/SERVICES.2013.71. URL: http://dx.doi.org/10.1109/SERVICES.2013.71 (cit. on p. 36).

[106] Ahlem Chérifa Khadir, Hassina Aliane, and Ahmed Guessoum. "Ontology learning: Grand tour and challenges". In: *Computer Science Review* 39 (2021), p. 100339. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2020.100339. URL: https://www.sciencedirect.com/science/article/pii/S1574013720304391 (cit. on p. 137).

[107] E. Khanfir et al. "A Web Service Selection Framework Based on User's Context and QoS". In: *2014 IEEE International Conference on Web Services*. June 2014, pp. 708–711. DOI: 10.1109/ICWS.2014.119 (cit. on pp. 28, 31).

[108] E Kharlamov et al. *Capturing Industrial Information Models with Ontologies and Constraints*. Tech. rep. University of Oxford, Siemens AG, 2016. URL: https://www.cs.ox.ac.uk/files/8278/ISWC16_ModelManager.pdf (cit. on p. 12).

[109] *Kjoshi storage ontology*. URL: https://www.csee.umbc.edu/~kjoshi1/storage_ontology.owl (visited on 10/10/2018) (cit. on p. 23).

[110] Adrian Klein, Fuyuki Ishikawa, and Shinichi Honiden. "Towards network-aware service composition in the cloud". In: *Proceedings of the 21st international conference on World Wide Web*. ACM. 2012, pp. 959–968 (cit. on p. 91).

[111] *Knowledge_Graph*. URL: https://en.wikipedia.org/wiki/Knowledge_Graph (visited on 04/21/2019) (cit. on p. 13).

[112] Jana Krejčí and Jan Stoklasa. "Aggregation in the analytic hierarchy process: Why weighted geometric mean should be used instead of weighted arithmetic mean". In: *Expert Systems with Applications* 114 (Dec. 2018), pp. 97–106. ISSN: 0957-4174. DOI: 10.1016/J.ESWA.2018.06.060. URL: https://www.sciencedirect.com/science/article/pii/S0957417418303981?via%7B%5C%%7D3Dihub (cit. on pp. 37, 87).

[113] Rakesh Ranjan Kumar, Binita Kumari, and Chiranjeev Kumar. "CCS-OSSR: A framework based on Hybrid MCDM for Optimal Service Selection and Ranking of Cloud Computing Services". In: *Clust. Comput.* 24.2 (2021), pp. 867–883. DOI: 10.1007/s10586-020-03166-3. URL: https://doi.org/10.1007/s10586-020-03166-3 (cit. on p. 137).

[114] Kwang Mong Sim. "Agent-Based Cloud Computing". In: *IEEE Transactions on Services Computing* 5.4 (2012), pp. 564–577. ISSN: 1939-1374. DOI: 10.1109/TSC.2011.52. URL: http://ieeexplore.ieee.org/document/6042853/ (cit. on p. 24).

[115] Young Choon Lee et al. "Profit-Driven Service Request Scheduling in Clouds". In: *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid'10)*. 2010. URL: https://ieeexplore.ieee.org/document/5493498 (cit. on pp. 9, 44).

[116] Maxime Lefrançois and Antoine Zimmermann. "The Unified Code for Units of Measure in RDF: cdt:ucum and other UCUM Datatypes". In: *Proc. Extended Semantic Web Conference (ESWC'18)*. Heraklion, Crete, Greece, June 2018. URL: http://www.maxime-lefrancois.info/docs/LefrancoisZimmermann-ESWC2018-UCUM.pdf (cit. on p. 137).

[117] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. "A SPARQL extension for generating RDF from heterogeneous formats". In: *Proc. Extended Semantic Web Conference (ESWC'17)*. Portoroz, Slovenia, May 2017. URL: http://www.maxime-lefrancois.info/docs/LefrancoisZimmermannBakerally-ESWC2017-Generate.pdf (cit. on pp. 113, 115).

[118] Ang Li et al. "CloudCmp". In: *Proceedings of the 10th annual conference on Internet measurement - IMC '10*. New York, New York, USA: ACM Press, Nov. 2010, p. 1. ISBN: 9781450304832. DOI: 10.1145/1879141.1879143. URL: http://dl.acm.org/citation.cfm?id=1879141.1879143 (cit. on p. 77).

[119] *Linked data*. URL: https://en.wikipedia.org/wiki/Linked_data (cit. on p. 13).

[120] *Linked Open Vocabularies*. URL: https://lov.linkeddata.es/dataset/lov/ (visited on 06/11/2019) (cit. on p. 16).

[121] *Little's law*. URL: https://en.wikipedia.org/wiki/Little's_law (visited on 06/09/2019) (cit. on p. 47).

[122] Changbin Liu, Boon Thau Loo, and Yun Mao. "Declarative automated cloud resource orchestration". In: *Proceedings of the 2nd ACM Symposium on Cloud Computing - SOCC '11*. New York, New York, USA: ACM Press, Oct. 2011, pp. 1–8. ISBN: 9781450309769. DOI: 10.1145/2038916.2038942. URL: http://dl.acm.org/citation.cfm?id=2038916.2038942 (cit. on p. 120).

[123] Mingming Liu et al. "A Heterogeneous QoS-Based Cloud Service Selection Approach Using Entropy Weight and GRA-ELECTRE III". In: *Mathematical Problems in Engineering* 2020 (2020), pp. 1–17 (cit. on p. 137).

[124] *M/G/1 queue*. URL: https://en.wikipedia.org/wiki/M/G/1_queue (visited on 06/23/2019) (cit. on p. 104).

[125] *M/G/k Queue*. URL: http://en.wikipedia.org/wiki/M/G/k%5C_queue (visited on 03/08/2015) (cit. on p. 105).

[126] *M/M/c Queue*. URL: http://en.wikipedia.org/wiki/M/M/c%5C_queue (visited on 03/06/2015) (cit. on p. 103).

[127] Hua Ma et al. "Variation-aware Cloud Service Selection via Collaborative QoS Prediction". In: *IEEE Transactions on Services Computing* (2019), pp. 1–1. ISSN: 1939-1374. DOI: 10.1109/TSC.2019.2895784 (cit. on p. 137).

[128] *OWL 2 Web Ontology Language Manchester Syntax (Second Edition)*. URL: https://www.w3.org/TR/owl2-manchester-syntax/ (visited on 03/06/2019) (cit. on p. 57).

[129] Yun Mao et al. "Cloud Resource Orchestration: A Data-Centric Approach". In: *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*. 2011, pp. 241–248. URL: http://cidrdb.org/cidr2011/Papers/CIDR11_Paper34.pdf (cit. on p. 120).

[130] *Markovs Inequality*. URL: http://en.wikipedia.org/wiki/Markov%5C%27s%5C_inequality (visited on 03/10/2015) (cit. on p. 105).

[131] David Martin et al. "Bringing Semantics to Web Services: The OWL-S Approach". In: Springer, Berlin, Heidelberg, 2005, pp. 26–42. DOI: 10.1007/978-3-540-30581-1_4. URL: http://link.springer.com/10.1007/978-3-540-30581-1_4 (cit. on p. 50).

[132] David L. Martin et al. "Bringing Semantics to Web Services with OWL-S". In: *World Wide Web* 10.3 (2007), pp. 243–277 (cit. on pp. 17, 50).

[133] P M Mell and T Grance. *The NIST definition of cloud computing*. Tech. rep. Gaithersburg, MD: National Institute of Standards and Technology, 2011. DOI: 10.6028/NIST.SP.800-145. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf (cit. on p. 1).

[134] Michael Menzel et al. "CloudGenius: A Hybrid Decision Support Method for Automating the Migration of Web Application Clusters to Public Clouds". In: *IEEE Transactions on Computers* 64.5 (May 2015), pp. 1336–1348. ISSN: 0018-9340. DOI: 10.1109/TC.2014.2317188. URL: http://ieeexplore.ieee.org/document/6811183/ (cit. on p. 36).

[135] Sudip Mittal et al. "Automatic Extraction of Metrics from SLAs for Cloud Service Management". In: *2016 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, Apr. 2016, pp. 139–142. ISBN: 978-1-5090-1961-8. DOI: 10.1109/IC2E.2016.14. URL: http://ieeexplore.ieee.org/document/7484176/ (cit. on pp. 27, 28).

[136] Francesco Moscato et al. "An analysis of mOSAIC ontology for Cloud resources annotation". In: *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)* (2011), pp. 973–980. URL: https://www.semanticscholar.org/paper/An-analysis-of-mOSAIC-ontology-for-Cloud-resources-Moscato-Aversa/86c8a95d74aa5387e3a0dbcc2da16fb769ee7697 (cit. on pp. 17, 19, 22, 50).

[137] Haralambos Mouratidis et al. "A framework to support selection of cloud providers based on security and privacy requirements". In: *Journal of Systems and Software* 86.9 (2013), pp. 2276–2293 (cit. on p. 137).

[138] *Multiple-criteria decision analysis*. URL: https://en.wikipedia.org/wiki/Multiple-criteria_decision_analysis (visited on 02/03/2019) (cit. on p. 33).

[139] *MySQL benchmark suite git repo*. URL: https://github.com/percona/ab/tree/master/tests/mysql-bench/mysql-bench (visited on 06/21/2019) (cit. on p. 110).

[140] *N-Triples*. URL: https://en.wikipedia.org/wiki/N-Triples (visited on 06/09/2019) (cit. on p. 13).

[141] Nada A Nabeeh, Haitham A El-Ghareeb, and A M Riad. "Integrating Software Agents and Web Services in Service Oriented Architecture Based Cloud Services Discovery Framework". In: (). URL: http://www.globalcis.org/jcit/ppl/JCIT4280PPL.pdf (cit. on p. 31).

[142] Vlad Nae, Alexandru Iosup, and Radu Prodan. "Dynamic resource provisioning in massively multiplayer online games". In: *Parallel and Distributed Systems, IEEE Transactions on* 22.3 (2011), pp. 380–395 (cit. on p. 75).

[143] *Network selection techniques:SAW and TOPSIS*. URL: https://www.slideshare.net/ysdagar/network-selection-techniquessaw-and-topsis (visited on 02/03/2019) (cit. on p. 40).

[144] Le Duy Ngan and Rajaraman Kanagasabai. "OWL-S Based Semantic Cloud Service Broker". In: *2012 IEEE 19th International Conference on Web Services*. IEEE, June 2012, pp. 560–567. ISBN: 978-1-4673-2131-0. DOI: 10.1109/ICWS.2012.103. URL: http://ieeexplore.ieee.org/document/6257853/ (cit. on p. 26).

[145] Guihua Nie, Qiping She, and Donglin Chen. "Evaluation Index System of Cloud Service and the Purchase Decision- Making Process Based on AHP". In: *Proceedings of the 2011 International Conference on Informatics, Cybernetics, and Computer Engineering (ICCE2011) November 19–20, 2011, Melbourne, Australia*. Ed. by Liangzhong Jiang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 345–352. ISBN: 978-3-642-25194-8 (cit. on p. 36).

[146] Shahzad Ahmed Nizamani. *A quality-aware cloud selection service for computational modellers*. University of Leeds, 2012. URL: http://etheses.whiterose.ac.uk/id/eprint/3131 (cit. on p. 36).

[147] Talal H. Noor et al. "CSCE: A Crawler Engine for Cloud Services Discovery on the World Wide Web". In: *2013 IEEE 20th International Conference on Web Services*. IEEE, June 2013, pp. 443–450. ISBN: 978-0-7695-5025-1. DOI: 10.1109/ICWS.2013.66. URL: http://ieeexplore.ieee.org/document/6649610/ (cit. on pp. 21, 23).

[148] *Notation3 Wiki*. URL: https://en.wikipedia.org/wiki/Notation3 (visited on 06/11/2019) (cit. on p. 14).

[149] *Optimization Problem Types - Mixed-Integer and Constraint Programming | solver*. URL: https://www.solver.com/integer-constraint-programming (visited on 06/25/2019) (cit. on p. 107).

[150] *OWL - Semantic Web Standards*. URL: https://www.w3.org/OWL/ (visited on 03/14/2018) (cit. on p. 50).

[151] *OWL-S: Semantic Markup for Web Services*. URL: https://www.w3.org/Submission/OWL-S/ (visited on 09/19/2018) (cit. on p. 57).

[152] Gultekin Ozsoyoglu and Abdullah Al-Hamdani. "Web Information Resource Discovery: Past, Present, and Future". In: Springer, Berlin, Heidelberg, 2003, pp. 9–18. DOI: 10.1007/978-3-540-39737-3_2. URL: http://link.springer.com/10.1007/978-3-540-39737-3%7B%5C_%7D2 (cit. on p. 50).

[153] *Pairwise Comparison*. URL: http://deseng.ryerson.ca/~fil/t/pwisecomp.html (visited on 09/17/2017) (cit. on p. 8).

[154] Manoranjan Parhi, Binod Kumar Pattanayak, and Manas Ranjan Patra. "A Multi-agent-Based Framework for Cloud Service Description and Discovery Using Ontology". In: *Intelligent Computing, Communication and Devices*. Springer, New Delhi, 2015, pp. 337–348. DOI: 10.1007/978-81-322-2012-1_35. URL: https://www.researchgate.net/publication/278647856_A_Multi-agent-Based_Framework_for_Cloud_Service_Description_and_Discovery_Using_Ontology (cit. on p. 32).

[155] *pgbench*. URL: https://www.postgresql.org/docs/10/pgbench.html (visited on 06/25/2019) (cit. on p. 111).

[156] *Preference ranking organization method for enrichment evaluation*. URL: https://en.wikipedia.org/wiki/Preference_ranking_organization_method_for_enrichment_evaluation (cit. on p. 40).

[157] *Pricing - FlexiScale Technologies Limited*. URL: http://www.flexiscale.com/pricing/ (visited on 09/05/2017) (cit. on p. 2).

[158] *Pricing calculator | Microsoft Azure*. URL: https://azure.microsoft.com/en-au/pricing/calculator/ (visited on 09/18/2017) (cit. on p. 32).

[159] *PROMETHEE - ML Wiki*. URL: http://mlwiki.org/index.php/PROMETHEE (visited on 02/04/2019) (cit. on p. 40).

[160] *PROV-O: The PROV Ontology*. URL: https://www.w3.org/TR/prov-o/ (visited on 06/25/2019) (cit. on pp. 8, 53).

[161] Chenxi Qiu and Haiying Shen. "Dynamic Demand Prediction and Allocation in Cloud Service Brokerage". In: *IEEE Transactions on Cloud Computing* (2019), pp. 1–1 (cit. on p. 137).

[162] Chenxi Qiu, Haiying Shen, and Liuhua Chen. "Towards Green Cloud Computing: Demand Allocation and Pricing Policies for Cloud Service Brokerage". In: *IEEE Transactions on Big Data* 5 (2019), pp. 238–251 (cit. on p. 137).

[163] *QUDT*. URL: http://www.qudt.org/ (visited on 02/07/2019) (cit. on pp. 8, 18, 53).

[164] *QUT MEN170: SYSTEMS MODELLING AND SIMULATION*. URL: https://studylib.net/doc/10284701/qut-men170--systems-modelling-and-simulation-school-of-me (visited on 05/30/2019) (cit. on p. 44).

[165] *Rackspace Australia*. URL: https://www.rackspace.com/en-au/cloud/servers/pricing (visited on 09/13/2017) (cit. on p. 2).

[166] *RDF - Semantic Web Standards*. URL: https://www.w3.org/RDF/ (visited on 08/28/2018) (cit. on pp. 12, 50).

[167] *RDF 1.1 Turtle - Terse RDF Triple Language Turtle*. Feb. 2014. URL: http://www.w3.org/TR/turtle/ (visited on 10/10/2018) (cit. on p. 57).

[168] *RDF and OWL*. URL: https://www.slideshare.net/rlovinger/rdf-and-owl (visited on 03/05/2019) (cit. on pp. 13, 16).

[169] *RDF Schema Wiki*. URL: https://en.wikipedia.org/wiki/RDF_Schema (visited on 06/11/2019) (cit. on p. 15).

[170] *RDF Triples*. URL: http://www.lesliesikos.com/rdf-triples/ (visited on 04/29/2018) (cit. on p. 13).

[171] Jafar Rezaei. "Best-worst multi-criteria decision-making method". In: *Omega* 53 (2015), pp. 49–57. ISSN: 0305-0483. DOI: https://doi.org/10.1016/j.omega.2014.11.009. URL: https://www.sciencedirect.com/science/article/pii/S0305048314001480 (cit. on p. 137).

[172] *RIPE Atlas - RIPE Network Coordination Centre*. URL: https://atlas.ripe.net/ (visited on 06/23/2019) (cit. on p. 110).

[173] Miguel Ángel Rodríguez-García et al. "Ontology-based annotation and retrieval of services in the cloud". In: *Knowledge-Based Systems* 56 (Jan. 2014), pp. 15–25. ISSN: 0950-7051. DOI: 10.1016/J.KNOSYS.2013.10.006. URL: https://www.sciencedirect.com/science/article/pii/S0950705113003171 (cit. on pp. 26, 27).

[174] Dumitru Roman et al. "Web Service Modeling Ontology". In: *Applied Ontology* 1.1 (2005), pp. 77–106 (cit. on pp. 17, 50).

[175] *Difference Between Amazon S3 and Amazon EBS*. URL: http://www.differencebetween.net/technology/internet/difference-between-amazon-s3-and-amazon-ebs (visited on 06/25/2019) (cit. on p. 51).

[176] R.W. Saaty. "The analytic hierarchy process—what it is and how it is used". In: *Mathematical Modelling* 9.3-5 (Jan. 1987), pp. 161–176. ISSN: 0270-0255. DOI: 10.1016/0270-0255(87)90473-8. URL: https://www.sciencedirect.com/science/article/pii/0270025587904738 (cit. on p. 36).

[177] T. Saaty and K. Kearns. "The Analytic Hierarchy Process". In: 1985 (cit. on p. 32).

[178] Thomas L. Saaty. "How to make a decision: The analytic hierarchy process". In: *European Journal of Operational Research* 48.1 (1990). Desicion making by the analytic hierarchy process: Theory and applications, pp. 9–26. ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(90)90057-I. URL: http://www.sciencedirect.com/science/article/pii/037722179090057I (cit. on p. 36).

[179] Prasad Saripalli and Gopal Pingali. "MADMAC: Multiple Attribute Decision Methodology for Adoption of Clouds". In: (2011). DOI: 10.1109/CLOUD.2011.61. URL: http://barbie.uta.edu/%7B~%7Dhdfeng/CloudComputing/cloud/cloud46.pdf (cit. on p. 34).

[180] *Schema.org*. URL: https://en.wikipedia.org/wiki/Schema.org (cit. on p. 13).

[181] *Selection (relational algebra) - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Selection_(relational_algebra). Accessed: 20-Feb-2014 (cit. on p. 88).

[182] *Semantic Web*. URL: https://en.wikipedia.org/wiki/Semantic_Web (cit. on p. 11).

[183] *semantic web - What is the difference between RDF and OWL? - Stack Overflow*. URL: https://stackoverflow.com/questions/1740341/what-is-the-difference-between-rdf-and-owl (visited on 03/05/2019) (cit. on p. 16).

[184] *Service Oriented Architecture*. URL: https://en.wikipedia.org/wiki/Service-oriented_architecture (cit. on p. 31).

[185] Anees Shaikh et al. "On demand platform for online games". In: *IBM Systems Journal* 45.1 (2006), pp. 7–19 (cit. on p. 75).

[186] *Signal transport and networks - SKA Telescope*. http://www.skatelescope.org/the-technology/signal-processing/. Accessed: 18-Sep-2013 (cit. on p. 77).

[187] *SoftLayer TCO Calculator*. URL: http://www.softlayer.com/tco/ (visited on 09/18/2017) (cit. on p. 32).

[188] *SPARQL*. URL: https://en.wikipedia.org/wiki/SPARQL (visited on 10/18/2018) (cit. on p. 137).

[189] *SPARQL-Generate*. URL: http://w3id.org/sparql-generate/ (visited on 08/13/2018) (cit. on pp. 113, 115).

[190] *SPIN - Overview and Motivation*. URL: https://www.w3.org/Submission/spin-overview/ (visited on 10/18/2018) (cit. on p. 137).

[191] Le Sun et al. "Cloud service selection: State-of-the-art and future research directions". In: *Journal of Network and Computer Applications* 45 (2014), pp. 134–150. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2014.07.019. URL: http://www.sciencedirect.com/science/article/pii/S108480451400160X (cit. on pp. 32, 36).

[192] S. Sundareswaran, A. Squicciarini, and D. Lin. "A Brokerage-Based Approach for Cloud Service Selection". In: *2012 IEEE Fifth International Conference on Cloud Computing*. June 2012, pp. 558–565. DOI: 10.1109/CLOUD.2012.119 (cit. on p. 41).

[193] *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. URL: https://www.w3.org/Submission/SWRL/ (visited on 04/26/2019) (cit. on p. 26).

[194] János Sztrik. *Basic Queueing Theory Foundations of System Performance Modeling*. ISBN: 3639734718. URL: http://irh.inf.unideb.hu/user/jsztrik/publications/books/GlobeEdit_Basic_Queueing_Theory_Sztrik_2016.pdf (visited on 05/10/2019) (cit. on pp. 44, 104).

[195] Amirreza Tahamtan et al. "A Cloud Repository and Discovery Framework Based on a Unified Business and Cloud Service Ontology". In: *2012 IEEE Eighth World Congress on Services*. IEEE, June 2012, pp. 203–210. ISBN: 978-1-4673-3053-4. DOI: 10.1109/SERVICES.2012.42. URL: http://ieeexplore.ieee.org/document/6274052/ (cit. on pp. 27, 29).

[196] *The Real Problem of Facebook Inc's WhatsApp Acquisition*. http://www.fool.com/investing/general/2014/03/24/two-real-problems-of-facebooks-whatsapp-acquisitio.aspx. Accessed: 27-Mar-2014 (cit. on p. 76).

[197] *The Server Labs Case Study: Amazon Web Services*. URL: http://aws.amazon.com/solutions/case-studies/the-server-labs/ (visited on 09/09/2013) (cit. on p. 132).

[198] *The Unified Code for Units of Measure*. URL: http://unitsofmeasure.org/ucum.html (visited on 01/24/2019) (cit. on p. 57).

[199] R. Tiwari and R. Kumar. "G-TOPSIS: a cloud service selection framework using Gaussian TOPSIS for rank reversal problem". In: *The Journal of Supercomputing* 77 (2020), pp. 523–562 (cit. on p. 137).

[200] Chris Tofallis. "Add or Multiply? A Tutorial on Ranking and Choosing with Multiple Criteria". In: *INFORMS Transactions on Education* 14.3 (May 2014), pp. 109–119. ISSN: 1532-0545. DOI: 10.1287/ited.2013.0124. URL: http://pubsonline.informs.org/doi/abs/10.1287/ited.2013.0124 (cit. on p. 35).

[201] Chris Tofallis. "Add or Multiply? A Tutorial on Ranking and Choosing with Multiple Criteria". In: *INFORMS Transactions on Education* 14.3 (2014), pp. 109–119. DOI: 10.1287/ited.2013.0124. eprint: https://doi.org/10.1287/ited.2013.0124. URL: https://doi.org/10.1287/ited.2013.0124 (cit. on p. 87).

[202] *tpcc-mysql*. URL: https://github.com/Percona-Lab/tpcc-mysql (visited on 06/25/2019) (cit. on p. 111).

[203] Evangelos Triantaphyllou and Stuart H. Mann. "An examination of the effectiveness of multi-dimensional decision-making methods: A decision-making paradox". In: *Decision Support Systems* 5.3 (Sept. 1989), pp. 303–312. ISSN: 0167-9236. DOI: 10.1016/0167-9236(89)90037-7. URL: https://www.sciencedirect.com/science/article/pii/0167923689900377?via%7B%5C%%7D3Dihub (cit. on p. 33).

[204] *Turtle syntax*. URL: https://en.wikipedia.org/wiki/Turtle_(syntax) (cit. on p. 13).

[205] Mike Uschold and Michael Gruninger. "Ontologies: Principles, methods and applications". In: *The knowledge engineering review* 11.2 (1996), pp. 93–136. URL: http://www.aiai.ed.ac.uk/publications/documents/1996/96-ker-intro-ontologies.pdf (cit. on p. 12).

[206] Magesh Vasudevan. "Semantic Discovery of Cloud Service Catalog Published Over Resource Description Framework". In: 2014. URL: https://www.semanticscholar.org/paper/Semantic-Discovery-of-Cloud-Service-Catalog-Over-Vasudevan-Iyengar/c2352469813c768af886b277110b1f9870f88ec3 (cit. on pp. 24, 25).

[207] *Viber*. http://en.wikipedia.org/wiki/Viber. Accessed: 27-Mar-2014 (cit. on p. 76).

[208] Lizhe Wang et al., eds. *Cloud computing : methodology, systems, and applications*. CRC Press, 2011. ISBN: 9781315217208. URL: https://www.crcpress.com/Cloud-Computing-Methodology-Systems-and-Applications/Wang-Ranjan-Chen-Benatallah/p/book/9781439856413 (cit. on p. 2).

[209] Lizhe Wang et al. "MapReduce across Distributed Clusters for Data-intensive Applications". In: *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*. May 2012, pp. 2004–2011. DOI: 10.1109/IPDPSW.2012.249 (cit. on p. 76).

[210] *We're moving from Azure to Google Cloud Platform | GitLab*. URL: https://about.gitlab.com/2018/06/25/moving-to-gcp/ (visited on 06/05/2019) (cit. on p. 6).

[211] *Web pages are getting more bloated, and here's why - Pingdom Royal*. URL: https://royal.pingdom.com/2011/11/21/web-pages-getting-bloated-here-is-why/ (visited on 10/19/2018) (cit. on p. 132).

[212] *Weighted product model*. URL: https://en.wikipedia.org/wiki/Weighted_product_model (visited on 02/03/2019) (cit. on p. 34).

[213] Joe Weinman. "As time goes by: The law of cloud response time". In: *online] April* (2011) (cit. on p. 76).

[214] Joe Weinman. *Axiomatic cloud theory*. Tech. rep. Working paper, 2011 (cit. on p. 4).

[215] *What is AHP? – BPMSG*. URL: https://bpmsg.com/ahp-introduction/ (visited on 06/11/2019) (cit. on p. 36).

[216] *What is the difference between RDF and OWL?* URL: https://www.quora.com/What-is-the-difference-between-RDF-and-OWL (cit. on p. 16).

[217] *What's All The Fuss About WhatsApp? China's WeChat Is a Worthy Rival - TIME.com*. http://time.com/8873/whats-all-the-fuss-about-whatsapp-chinas-wechat-is-a-worthy-rival. Accessed: 27-Mar-2014 (cit. on p. 76).

[218] *What's New in RDF 1.1*. URL: https://www.w3.org/TR/rdf11-new/ (visited on 06/09/2019) (cit. on p. 14).

[219] John Wheal and Yanyan Yang. "CSRecommender: A Cloud Service Searching and Recommendation System". In: *Journal of Computer and Communications* 03.06 (May 2015), pp. 65–73. ISSN: 2327-5219. DOI: 10.4236/jcc.2015.36007. URL: http://www.scirp.org/journal/doi.aspx?DOI=10.4236/jcc.2015.36007 (cit. on p. 32).

[220] *Wikidata*. URL: https://www.wikidata.org/wiki/Wikidata:Main%7B%5C_%7DPage (visited on 06/05/2019) (cit. on pp. 8, 53).

[221] *Wikimedia*. URL: https://www.wikimedia.org/ (visited on 06/20/2019) (cit. on p. 108).

[222] Daya C. Wimalasuriya and Dejing Dou. "Ontology-based information extraction: An introduction and a survey of current approaches". In: *Journal of Information Science* 36.3 (2010), pp. 306–323. DOI: 10.1177/0165551509360123. eprint: https://doi.org/10.1177/0165551509360123. URL: https://doi.org/10.1177/0165551509360123 (cit. on p. 137).

[223] *WordNet*. URL: https://en.wikipedia.org/wiki/WordNet (cit. on pp. 24, 25).

[224] *WSMO*. URL: https://www.w3.org/Submission/WSMO/ (cit. on p. 50).

[225] Zichuan Xu and Weifa Liang. "Minimizing the Operational Cost of Data Centers via Geographical Electricity Price Diversity". In: *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. June 2013, pp. 99–106. DOI: 10.1109/CLOUD.2013.94 (cit. on pp. 9, 44).

[226] Yuli Yang, Nanyue Yu, and Yongle Chen. "Trusted Cloud Service Selection Algorithm Based on Lightweight Intuitionistic Fuzzy Numbers". In: *IEEE Access* 8 (2020), pp. 97748–97756 (cit. on p. 137).

[227] Zhen Ye, Athman Bouguettaya, and Xiaofang Zhou. "QoS-Aware Cloud Service Composition Based on Economic Models". In: *Service-Oriented Computing*. Ed. by Chengfei Liu et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 111–126. ISBN: 978-3-642-34321-6 (cit. on p. 41).

[228] *Yelp Case Study - Amazon Web Services (AWS)*. URL: https://aws.amazon.com/solutions/case-studies/yelp/ (visited on 10/19/2018) (cit. on p. 132).

[229] Lamia Youseff, Maria Butrico, and Dilma Da Silva. "Toward a Unified Ontology of Cloud Computing". In: *2008 Grid Computing Environments Workshop*. IEEE, Nov. 2008, pp. 1–10. ISBN: 978-1-4244-2860-1. DOI: 10.1109/GCE.2008.4738443. URL: http://ieeexplore.ieee.org/document/4738443/ (cit. on p. 50).

[230] Ahmed E. Youssef. "An Integrated MCDM Approach for Cloud Service Selection Based on TOPSIS and BWM". In: *IEEE Access* 8 (2020), pp. 71851–71865 (cit. on p. 137).

[231] Tao Yu, Yue Zhang, and Kwei-Jay Lin. "Efficient algorithms for Web services selection with end-to-end QoS constraints". In: *ACM Transactions on the Web (TWEB)* 1.1 (May 2007), 6–es. ISSN: 15591131. DOI: 10.1145/1232722.1232728. URL: http://dl.acm.org/citation.cfm?id=1232722.1232728 (cit. on p. 73).

[232] Miranda Zhang et al. "A Declarative Recommender System for Cloud Infrastructure Services Selection". In: *Economics of Grids, Clouds, Systems, and Services*. Ed. by Kurt Vanmechelen, Jörn Altmann, and Omer F. Rana. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 102–113. ISBN: 978-3-642-35194-5 (cit. on p. 53).

[233] Miranda Zhang et al. "An Infrastructure Service Recommendation System for Cloud Applications with Real-time QoS Requirement Constraints". In: *IEEE Systems Journal* PP.99 (2015), pp. 1–11. ISSN: 1932-8184. DOI: 10.1109/JSYST.2015.2427338. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7124433 (cit. on p. 87).

[234] Miranda Zhang et al. "An ontology-based system for Cloud infrastructure services' discovery". In: *CollaborateCom*. 2012. URL: http://ieeexplore.ieee.org/document/6450944/ (cit. on pp. 53, 54).

[235] Miranda Zhang et al. "Investigating Techniques for Automating the Selection of Cloud Infrastructure Services". en. In: *INTERNATIONAL JOURNAL OF NEXT-GENERATION COMPUTING* 4.3 (2013). ISSN: 0976-5034. URL: http://perpetualinnovation.net/ojs/index.php/ijngc/article/view/227 (cit. on p. 54).

[236] Nian Zhang and Guiwu Wei. "Extension of VIKOR method for decision making problem based on hesitant fuzzy set". In: *Applied Mathematical Modelling* 37.7 (Apr. 2013), pp. 4938–4947. ISSN: 0307-904X. DOI: 10.1016/J.APM.2012.10.002. URL: https://www.sciencedirect.com/science/article/pii/S0307904X12006075%7B%5C#%7Dt0010 (cit. on p. 40).

[237] Ying Zhang et al. "On P2P-Based Semantic Service Discovery with QoS Measurements for Pervasive Services in the Universal Network". In: *Journal of Computers* 9 (Aug. 2014). DOI: 10.4304/jcp.9.8.1914-1921. URL: https://www.researchgate.net/publication/273212719_On_P2P-Based_Semantic_Service_Discovery_with_QoS_Measurements_for_Pervasive_Services_in_the_Universal_Network (cit. on pp. 28, 29, 66).

[238] Laiping Zhao et al. "Flexible service selection with user-specific QoS support in service-oriented architecture". In: *Journal of Network and Computer Applications* 35.3 (2012). Special Issue on Trusted Computing and Communications, pp. 962–973. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2011.03.013. URL: http://www.sciencedirect.com/science/article/pii/S1084804511000671 (cit. on p. 34).

[239] Huiyuan Zheng et al. "QoS Analysis for Web Service Compositions with Complex Structures". In: *IEEE Transactions on Services Computing* 6.3 (July 2013), pp. 373–386. ISSN: 1939-1374. DOI: 10.1109/TSC.2012.7. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6158633 (cit. on p. 73).

[240] Z. Zheng et al. "QoS Ranking Prediction for Cloud Services". In: *IEEE Transactions on Parallel and Distributed Systems* 24.6 (June 2013), pp. 1213–1222. ISSN: 1045-9219. DOI: 10.1109/TPDS.2012.285 (cit. on pp. 41, 42).

[241] D Zormpa and C Evangelides. "Multiple Criteria Decision Making Using VIKOR Method. Application in Irrigation Networks in the Thessaloniki Plain". In: URL: https://cest2015.gnest.org/papers/cest2015_00039_oral_paper.pdf (cit. on p. 40).

Index