University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Marin Šilić

RELIABILITY PREDICTION OF CONSUMER COMPUTING APPLICATIONS

DOCTORAL THESIS

Zagreb, 2013

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Marin Šilić

RELIABILITY PREDICTION OF CONSUMER COMPUTING APPLICATIONS

DOCTORAL THESIS

Supervisor: Professor Siniša Srbljić, Ph.D.

Zagreb, 2013

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Marin Šilić

PREDVIĐANJE POUZDANOSTI PRIMJENSKIH PROGRAMA POTROŠAČKOG RAČUNARSTVA

DOKTORSKI RAD

Mentor: Prof. dr. sc. Siniša Srbljić

Zagreb, 2013.

Doctoral thesis was made at the University of Zagreb,

Faculty of Electrical Engineering and Computing ,

Department of Electronics, Microelectronics, Computer and Intelligent Systems

Supervisor:

Professor Siniša Srbljić, Ph.D.

Doctoral thesis contains: 179 pages.

Doctoral thesis number: _____

About the Supervisor:

Siniša Srbljić was born in Velika Gorica in 1958. He received B.Sc. degree in EE and M.Sc. and Ph.D. degrees in CS from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), in 1981, 1985, and 1990, respectively.

From February 1982, he is working at the Department ZEMRIS at FER. He was working with Prvomajska, R&D Dep., Croatia (1984-86) and he was visiting scientist at the University of Toronto, Canada (1993-1995), at the AT&T Labs, USA (1995-99), at the UC, Irvine, USA (2000,08-09), at the GlobalLogic Inc, USA (2011), and at the US Huawei, USA (2011). In March 2007, he was promoted to tenured professor. He coordinated 1 scientific program, led 1 scientific project, 1 technological project, and participated in 9 scientific projects. He led 3 research projects financed by companies from Croatia and USA, led 2 projects in USA, and participated in 3 projects in USA and Canada. He coordinates scientific program "Distributed Systems, Methods, and Applications" and leads scientific project "Computing Environments for Ubiquitous Distributed Systems" financed by the MZOS RH. He is author of 2 textbooks, more than 60 papers in journals and conference proceedings, and two 2 patents in USA in the area of distributed computing systems and consumer computing.

Prof. Srbljić is a member of IEEE, ACM, and HATZ. He received Silver medal "Josip Lončar" from FER for Ph.D. thesis in 1990 and Vratislav Bedjanič Award, Iskra, Ljubljana, for M.Sc. thesis in 1985.

O mentoru:

Siniša Srbljić rođen je u Velikoj Gorici 1958. godine. Diplomirao je u polju elektrotehnike, a magistrirao i doktorirao u polju računarstva na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 1981., 1985. odnosno 1990. godine.

Od veljače 1982. godine radi u Zavodu ZEMRIS FER-a. Bio je zaposlen u tvornici Prvomajska, odjel Istraživanje i razvoj (1984.-86.), a gostujući znanstvenik bio je na University of Toronto, Kanada (1993.-95.), u AT&T Labs, SAD (1995.-99.), na UC, Irvine, SAD (2000., 2008.-09.), u GlobalLogic Inc (2011.) i u US Huawei, SAD (2011.). U ožujku 2007. godine izabran je u trajno znanstveno-nastavno zvanje redovitog profesora. Koordinirao je 1 znanstveni program, vodio 1 znanstveni projekt, 1 tehnologijski projekt i sudjelovao na 9 znanstvenih pro-

jekta MZOS RH. Vodio je 3 istraživačka projekta financirana od kompanija iz Hrvatske i SAD, vodio 2 projekta u SAD i sudjelovao na 3 projekta u SAD i Kanadi. Koordinira znanstveni program "Raspodijeljeni sustavi, metode i primjene" i vodi znanstveni projekt "Računalne okoline za sveprisutne raspodijeljene sustave" koje financira MZOS RH. Autor je 2 udžbenika, više od 60 radova u časopisima i zbornicima konferencija i 2 patenta u SAD u području raspodijeljenih računalnih sustava i potrošačkog računarstva.

Prof. Srbljić član je IEEE, ACM i HATZ. Primio je srebrnu plaketu "Josip Lončar" FER-a za doktorsku disertaciju (1990.) i nagradu Vratislav Bedjanič, Iskra Ljubljana, za magistarski rad (1985.).

Patience is the art of hoping.

(*Marquis De Vauvenargues*, $1715 - 1747$)

To my parents, Divna and Miroslav. Thank you for your endless love and support that you gave me...

Acknowledgments

I would like to take this opportunity to express my personal gratitude to some important people who made this dissertation possible. All PhD candidates should be aware that it takes a relatively long period of time to graduate. From this perspective, I see my graduation as a long race with time, in which patience is your crucial ally for final success. All along this road, you will go through good and bad times, and you will meet a lot of outstanding people who will give you support and strength to advance towards the finish line.

First of all, I would like to thank my supervisor, Professor Siniša Srbljić, PhD, head of the Consumer Computing Laboratory (CCL) and full time professor at the University of Zagreb (UniZg), Faculty of Electrical Engineering and Computing (FER). During my PhD studies, Professor Srbljić gave me support and encouragement through numerous discussions, suggestions and personal advices that helped me in my research process and accomplished scientific results. In particular, I would like to emphasize his strong will to accept me as a PhD student back in 2007. This provided me a chance to fulfill my personal goals and finish my PhD. Professor Srbljić, my personal gratitude!

I think it is important to note that this thesis is not just my personal success, but also a success of my whole family. Special thanks go to my parents, Divna and Miroslav, who gave me endless support in every sense and not just during my studies, but through my entire life. I want to express my gratitude to my aunt, Seka, who I personally see as my second mother, and my cousin Goran who often, and especially during some hard times, endeavored to convince me that is worth to hold on and graduate in the end.

I would also like to thank my friend and college Goran Delac, who is a PhD candidate and research assistant at the UniZg, FER. Most of my graduation time, I was working with Goran who provided significant contributions through important discussions and brainstormings to all of my achieved results. Next, I would like to thank my friend Ivo Krka, who graduated his PhD at the University of Southern California, LA. Ivo's help was essential for me, especially in the beginning while I was trying to focus and converge my research ideas.

Furthermore, I would like to thank my friends and colleges at the UniZg who are members of the CCL lab including Miro Popović, Ivan Budiselić, Ivan Žužak, Dejan Škvorc, Klemo Vladimir, Ivan Gavran and Zvonimir Pavlić. Thank you all for your help and advices during my research and teaching activities, and various industry projects. In addition, I want to mention

colleges at the UniZg who are not members of CCL lab, but who were also helpful during discussions and relaxing times. Hence, special thanks to Artur Šilić, research assistant at UniZg, FER, and Željko Ilić, associate professor at UniZg, FER. I will certainly remember everyone of you along with all the good times during my PhD at the UniZg, FER.

Special thanks go to an associate member of CCL lab Boris Debic who is currently working in Google Inc., Mountain View, California. Boris helped me to apply for the internship in Google Inc., wherein some of my key concepts and research ideas were born back in 2008. During my internship in Google Inc. in NYC, I had a privilege to collaborate with many extraordinary individuals. Thus, I would like to name some of them here: Johnathan Rochelle, Micah Lemonik and Fuzzy Khosrowshahi.

At last, but certainly not the least, I would like to thank my personal friends who are not primarily associated with computer science, but who were very supportive during this six years, particularly through some refreshing breaks in between hard working periods. Special thanks go to members of Folklore Ensemble "Zagreb Markovac", Slavo Jakša, Ivan Lukić, Josip Soče, Veljko Srzić and all others!

Abstract

Consumer Computing is a novel consumer oriented methodology for development of component based applications. Consumers use existing applications as building blocks and compose them into more complex applications with the aim to support the additional desired functionalities. Besides the appropriate functionality, application's nonfunctional properties such as reliability and availability might significantly impact the perceived quality of the application. The reliability of the application depends on various number of parameters that describe the context of the application usage. The additional challenge while determining the reliability of applications is a substantially large number of applications in Consumer Computing and a very limited subset of applications with known reliability values. To address this challenge, new statistical methods for prediction of applications reliability need to be proposed. Besides the accurate reliability predictions, the proposed methods should also support real-time prediction performance in order to be used during the selection of components while creating composite applications in Consumer Computing.

Keywords: Consumer computing, Service-oriented computing, Dynamic software, Reliability, Prediction, Statistical models, LUCS, CLUS

Prošireni sažetak

Predviđanje pouzdanosti primjenskih programa potrošačkoga računarstva

Potrošačko računarstvo je nova metodologija razvoja primjenskih programa zasnovanih na komponentama. Potrošači koriste postojeće primjenske programe kao osnovne gradivne elemente i povezuju ih u složenije primjenske programe s ciljem ostvarenja dodatnih željenih funkcionalnosti. Osim ispravne funkcionalnosti, znatan utjecaj na doživljaj primjenskog programa imaju i nefunkcijska svojstva, kao što su pouzdanost i dostupnost. Pouzdanost komponenti ovisi o velikom broju parametara koji određuju kontekst korištenja primjenskog programa. Budući da u ekosustavu potrošačkog računarstva postoji veliki broj potrošača i primjenskih programa, vrijednosti pouzdanosti poznate su tek za mali podskup komponenata. Cilj disertacije je oblikovati metodu za predviđanje pouzdanosti komponenti kojima vrijednosti pouzdanosti nisu poznate. Osim preciznog predviđanja pouzdanosti, oblikovana metoda treba zadovoljiti svojstvo razmjernog rasta i rada u stvarnom vremenu da bi se koristila tijekom izbora komponenti u procesu izgradnje složenih potrošačkih programa.

U prvom poglavlju (1 "*Introduction*") doktorske disertacije iznosi se kratak pregled razmatranog područja kao i motivacija za provedeno istraživanje. Nadalje, opisani su glavni ciljevi istraživanja i izložen je kratak sažetak sadržaja doktorske disertacije. Drugo poglavlje (2 "Consumer Computing") opisuje potrošačko računarstvo. Detaljno se opisuje motivacija, ostvarivost, metodologija izgradnje potrošačkih primjenskih programa te okolina potrošačkog računarstva. U nastavku poglavlja razlaže se arhitektura primjenskih programa u potrošačkom računarstvu s ciljem razumijevanja posebnih zahtjeva za oblikovanje njihove pouzdanosti.

Poglavlje (3 "*Reliability in Consumer Computing*") prikazuje oblikovanje pouzdanosti primjenskih programa potrošačkog računarstva. Na početku poglavlja opisuju se osnovni teorijski koncepti i standardni pojmovi područja oblikovanja pouzdanosti programske potpore. U nastavku poglavlja, potrošački primjenski programi prikazuju se kao dinamički programski elementi kao što su usluge na Internetu. Na taj se način posebni izazovi koji su prisutni u postupku oblikovanja pouzdanosti u sustavima zasnovanim na uslugama prenose u područje potrošačkog računarstva. Da bi se prevladali spomenuti izazovi i odredila pouzdanost potrošačkih primjenskih programa potrebno je skupiti podatke o uspješnosti pojedinih primjenskih programa i koristiti metode predviđanja. Konačno, potrebno je oblikovati intuitivne mehanizme koji će

potrošačima pružati informacije o pouzdanosti potencijalnih kandidata u procesu odabira komponenti s ciljem optimiranja pouzdanosti složenih primjenskih programa.

Poglavlje 4 (4 "*State-of-the-art Models for Prediction of Application's Reliability*") donosi pregled do sada najuspješnijih postojećih pristupa predviđanja pouzdanosti usluga koji se zasnivaju na tehnici suradničkog filtriranja. Postoje tri osnovna tipa suradničkog filtriranja: zasnovan na memoriji, zasnovan na modelu i hibridni tip. Uz pregled svih tipova suradničkog filtriranja izlažu se i reprezentativni pristupi kao i glavni nedostatci i prednosti za svaki tip. S aspekta primjene u potrošačkom računarstvu, razmatraju se samo pristupi suradničkog filtriranja zasnovani na memoriji jer su hibridni i tip suradničkog filtriranja zasnovan na modelu računalno složeniji za ostvarenje te često zahtijevaju dodatne unutarnje informacije o sustavu. Iako postojeći pristupi zasnovani na tehnici suradničkog filtriranja postižu dobre rezultate predviđanja vrijednosti pouzdanosti, spomenuti pristupi pokazuju ozbiljne nedostatke kao što su nepodržavanje svojstva razmjernog rasta i slaba preciznost predviđanja u dinamičkim okolinama.

U poglavlju 5 (5 "*LUCS - Model for Prediction of Application's Reliability*") uvodi se *LUCS*, model za predviđanje pouzdanosti web usluga predložen u sklopu disertacije. *LUCS* procjenjuje vrijednost pouzdanosti usluge primjenom skupljenih podataka o prethodnim pozivima usluge. Model zasniva svoje predviđanje na osnovi vrijednosti sljedećih parametara: lokacija korisnika, lokacija usluge, opterećenje i klasa usluge (određuje računalnu složenost usluge). Model *LUCS* poboljšava postojeće pristupe zasnovane na tehnici suradničkog filtriranja na način da: (1) uvodi nove parametre opterećenje i klasa usluge u model i grupira zapise o prethodnim pozivima usluga prema parametrima modela te (2) provodi algoritam suradničkog filtriranja otkrivanjem sličnih entiteta s obzirom na svaki parametar modela. Konačna procijenjena vrijednost pouzdanosti računa se kao linearna kombinacija pojedinih utjecaja svakog parametra modela.

Budući da preciznost predviđanja *LUCS* modela znatno ovisi o izravnoj dostupnosti vrijednosti parametara modela, u sklopu disertacije predložen je fleksibilniji model predviđanja pouzdanosti web usluga *CLUS*, detaljno opisan u poglavlju 6 (6 "*CLUS - Model for Prediction of Application's Reliability*"). *CLUS* model procjenjuje vrijednost pouzdanosti usluge prema dostupnim podacima o pouzdanosti prethodnih poziva usluga. Postojeći pristupi koji su zasnovani na tehnici suradničkog filtriranja u procesu predviđanja posredno razmatraju samo parametre poziva usluge svojstvene korisniku i usluzi. S ciljem poboljšanja preciznosti predviđanja, model *CLUS* uvodi parametre poziva usluge svojstvene okolini koji opisuju stanje opterećenja u sustavu. U skladu s tim, postojeći podaci o pouzdanosti prethodnih poziva usluga raspršeni su kroz

dodatnu dimenziju koja odgovara stanju opterećenja okoline s obzirom na stanje opterećenja u okolini u trenutku kad su podaci prikupljeni. S ciljem poboljšanja svojstva razmjernog rasta, model *CLUS* smanjuje količinu suvišnih podataka grupiranjem korisnika i usluga u odgovarajuće uzorke korisnika i usluga s obzirom na njihovu vrijednost pouzdanosti korištenjem K-means clustering algoritma.

Poglavlje 7 (7 "Evaluation") donosi detaljne i iscrpne rezultate vrednovanja koji analiziraju različite kvalitativne aspekte predloženih modela u usporedbi s postojećim pristupima koji su zasnovani na tehnici suradničkog filtriranja. Na početku poglavlja opisuju se postavke provedenih eksperimenata. U nastavku se opisuje utjecaj različitih parametara na svojstva preciznosti i razmjernog rasta predviđanja za sve razmatrane pristupe predviđanja. Postupkom analize složenosti svih razmatranih modela predviđanja potvrđuju se rezultati vrednovanja s obzirom na svojstvo razmjernog rasta koji svjedoče o boljem svojstvu razmjernog rasta modela predloženih u sklopu disertacije. Poglavlje se završava s kratkim sažetkom svih rezultata vrednovanja na jednom mjestu. Poglavlje 8 (8 "Conclusion") razmatra ostvarene izvorne znanstvene doprinose.

Ključne riječi: Potrošačko računarstvo, Računarstvo zasnovano na uslugama, Dinamička programska potpora, Pouzdnost, Predviđanje, Statistički modeli, LUCS, CLUS

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Consumer Computing is a new research area in a contemporary Computer Science, introduced by the members of the *Consumer Computing Lab* (*CCL*) [1] at the Faculty of Electrical Engineering and Computing, University of Zagreb. The main goal of Consumer Computing is to provide *consumers* the ability to express their knowledge while creating applications for their consumption devices. Consumers are modern people without formal education in Computer Science and without any skills or previous experience in programming. Although they do not possess any theoretical or practical knowledge about programming, they are very experienced in using state-of-the-art technology including both physical devices such as tablets, smartphones and smart TVs, etc., and popular software products such as web tools, social networks, software widgets, mobile applications, etc.

Contemporary consumers use a variety of different applications on various consumption devices. For instance, consumers use web applications such as social network sites, search engines, multimedia sites, blogs, e-banking, etc. which makes them very skillful in consuming content on the Web. In fact, consumers use the same kinds of applications in a very similar manner on their physical devices such as tablets or smartphones. Even though they are the majority in nowadays digital world, the applications they use are designed and created by the digital world's minority group – software developers.

The current approach for application development gives consumers only a secondary role and it clearly reflects the inequality that is present in a digital world between consumers and developers [2]. More specifically, this approach is very often initiated by developers who prepare a number of potentially successful applications. In the next step, they conduct a case study, in which they challenge consumers to provide feedback and impressions regarding the proposed

applications. Finally, according to the collected feedback, developers create potentially the most profitable applications.

By contrast, Consumer Computing aims to propose a different approach for application creation. The idea is to empower consumers to create their own applications according to their knowledge, needs and preferences. In Consumer Computing, consumers create component-based applications using basic building blocks. The basic building blocks are the existing applications which are combined in order to support the desired functionalities. The process of components selection is very important during applications creation in component-based systems. The selection of the appropriate functional component is mandatory for the correct functioning of the application. However, besides functional requirements while selecting components, nonfunctional properties such as reliability, availability, maintainability, etc. also need to be consider due to their significant impact on the perceived quality of the application.

This dissertation is studying the reliability property of applications in Consumer Computing. According to their architecture, consumer applications are component-based systems. In order to develop reliable component based system, it is necessary to know the reliability of each basic component. Consumer applications provide their functionality through a simple and intuitive user interface and hide the implementation details underneath. However, they often require some data retrieval or information processing on the Internet. Hence, consumer applications can be viewed as a dynamic software artifact such as services. The modeling of services reliability on the Web is very challenging task since the variety of parameters determine the service invocation context.

The researchers have proposed plenty of different models for reliability modeling of traditional software systems [3–11], but these approaches are not applicable for services reliability modeling. The variations in perceived reliability occur due to different perspectives of service users and service providers. The reliability of the service can be computed from the available data about previous invocations as the ratio of number of successful invocations against the total number of invocations [12]. The accuracy of the computed reliability value depends on the quality of the previous invocation sample. The process of acquiring comprehensive invocation sample proves to be a difficult task. This is particularly the case in situations in which a user has not previously used the service or the usage frequency has been low. Further obstacles such as service cost and performance issues can hamper the efforts to accurately estimate the service reliability [13].

A possible solution is to gain partial but relevant invocation sample by collecting feedback both from service users and providers, and to utilize prediction methods to estimate the reliability for the missing records. The most successful approaches for services reliability prediction on the Web [14–17] are based on the *collaborative filtering* [18] technique. There exist three types of collaborative filtering: *memory-based*, *model-based* and *hybrid*. Since the model-based and hybrid collaborative filtering are more complex and costly for implementation, the discussion is focused on memory-based collaborative filtering underlying the state-of-the-art recommendation systems [19–23]. The memory-based collaborative filtering extracts information or patterns using statistical collaboration among multiple entities such as agents, viewpoints, and data sources. The similarity relations among different entities are calculated using specialized statistical methods. The benefit of collaborative filtering is that it can be applied in situations in which specific data that is lacking can be predicted using the available data from the most statistically similar entities.

Although the existing collaborative filtering based approaches provide promising results in services reliability prediction, they demonstrate some serious disadvantages such as *scalability* issues and poor prediction *accuracy* in dynamic environments. The aim of this doctoral thesis is to propose a new method for prediction of services reliability that can be applied in Consumer Computing to assess consumer applications reliability. Besides the *accurate* prediction of applications reliability, the proposed method should also support *scalable* and *real-time* performance.

The rest of the dissertation is organized as follows. Chapter 2 presents the Consumer Computing. The underlying motivation, feasibility, programming methodology and environment of fully supported Consumer Computing are described. Additionally, the architecture of applications in Consumer Computing is examined in order to understand the specific requirements for their reliability modeling.

Chapter 3 presents the reliability modeling of applications in Consumer Computing. First, some basic theoretical concepts and taxonomy of software reliability are described. Next, consumer applications are considered as dynamic software artifacts such as services on the Internet. In such manner, specific obstacles present in reliability assessment in service-oriented systems are introduced into Consumer Computing. In order to overcome these obstacles and asses the reliability of applications in Consumer Computing, the feedback regarding the selected components needs to be collected, and prediction methods need to be used. In addition, to enable

consumers to optimize the reliability while creating their applications, the appropriate consumer intuitive mechanisms need to be proposed to represent the reliability of potential selection candidates.

Chapter 4 overviews so far most successful approaches for prediction of services reliability which are based on the collaborative filtering technique. As already stated, there are three different types of collaborative filtering: memory-based, model-based and hybrid. As part of the overview, each type of collaborative filtering is presented along with its representative approaches, main advantages and drawbacks. Regarding the appliance in Consumer Computing, only memory-based collaborative filtering approaches are considered since model-based and hybrid approaches are computationally heavier, more difficult to implement and often require some additional internal information from the system. Although the memory-based collaborative filtering approaches demonstrate promising results, they suffer from potential scalability issues and poor accuracy in dynamic environments.

Chapter 5 introduces *LUCS* [24], a model for reliability prediction of web services that is proposed as part of this dissertation. *LUCS* estimates the reliability for an ongoing request using the collected data about previous invocations. The model bases its prediction considering following parameters at the invocation time: user's location, service's location, service load and service class (describing service internals regarding its computational complexity). The *LUCS* approach improves the existing collaborative filtering approaches by: (1) extending the model with parameters service load and class and grouping the service invocations records according to the model parameters, and (2) performing collaborative filtering by discovering similar entities regarding each model parameter. The final prediction is computed as a linear combination of different parameters impacts.

Since the *LUCS* approach appears to be very dependent on the explicit availability of model parameters, a more flexible *CLUS* [25] approach is proposed in Chapter 6. The *CLUS* approach predicts the reliability for an ongoing request based on the available records about past invocations reliability. The existing collaborative filtering approaches implicitly consider only user– and service–specific parameters of the service invocation context. To improve the prediction accuracy, the *CLUS* approach introduces environment–specific parameters that describe current load conditions in the system. In such manner, the collected past invocation data is dispersed across the additional dimension that corresponds with the respected environment conditions. To improve the scalability, the model reduces the redundant data by grouping users and ser-

vices into respected user and service clusters according to their reliability performance using *K-means* clustering algorithm [26, 27].

Chapter 7 brings the exhaustive and detailed evaluation results which analyze different quality aspects of proposed models in comparison with the existing collaborative filtering based approaches. In the beginning, the experiment setup is described. In addition, different impacts on prediction accuracy and computational performance are considered for each competing approach. Furthermore, the analysis of complexity is given to support evaluation results regarding computational performance and to bear out claims in favor of better scalability of the models proposed in this dissertation. The end of the chapter brings a brief summary of all evaluation results in one place. Chapter 8 discusses the original scientific contributions and finally concludes the dissertation.

Chapter 2

Consumer Computing

Consumer Computing is a novel consumer-oriented methodology for application development which enables the average consumers, ordinary people without formal education in computer science and prerequisite coding skills, to create their own applications. The term *Consumer Computing* was introduced as part of the research in the *Consumer Computing Lab* (*CCL*) at the University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb [1]. As defined in various publications of the members of *CCL* laboratory [1, 28–32], a *consumer* is an average individual who has no formal education in computer science and no necessary programming knowledge, but who is highly experienced in consuming contemporary technology including popular web tools such as web browsers and software widgets. Consumers are also very familiar with modern physical devices such as tablets, smart-phones and other similar gadgets.

The underlying motivation for Consumer Computing is primarily the increased demand for various applications development on the market. The most recent projections on number of experts formally educated in computer science indicate that exclusively software engineers will not be able to satisfy the rising demand for applications on the market. The supporters of Consumer Computing aim to empower the average people to create and share their own personalized applications. In such manner, the increased demand for new applications on the market would be more easily overcome thanks to the collective power of the crowd of numerous consumers. The additional benefit of Consumer Computing is hidden in domain specific applications developed by certain experts in their field. By creating consumer composite application, each individual transforms personal tacit knowledge into a process that can be executed on a machine and saved as a new consumer applications which can be further used and modified by

others. In this way, both various domain specific experts and ordinary people as well, help disseminate their personal knowledge into a collective knowledge archived within the developed consumer applications.

The members of the *Consumer Computing Lab* (*CCL*), at the University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb have implemented a consumer oriented programming tool *Geppeto* [33], which enables consumers to create applications. Although *Geppeto* still needs additional modifications and extensions to fully support Consumer Computing, it is a closest implementation of Consumer Computing environment so far.

The rest of the chapter is organized as follows. Section 2.1 explains the goals of Consumer Computing and introduces motivation for consumer application development. Section 2.2 describes the programming methodology for consumer applications development in Consumer Computing and presents the programming methodology that is used in *Geppeto*. Section 2.3 provides detailed architecture of Consumer Computing Environment and describes different entities that are present in Consumer Computing. Finally, Section 2.4 analyses the underlying architecture of applications in Consumer Computing and studies different properties of applications in Consumer Computing.

2.1 Feasibility of Consumer Computing

Different parts of this section try to explain motivation, feasibility, goals and challenges of Consumer Computing. Section 2.1.1 introduces motivation and explain why is Consumer Computing needed in a contemporary digital world. In addition, Section 2.1.2 brings out the arguments that support thesis that the current *state-of-the-art* technology has reached a level when it is quite possible to provide the average consumers tools to create their own applications. Section 2.1.3 describes the additional benefits of Consumer Computing such as knowledge automation, elimination of digital world divide and achievement of digital world equality, scalability and sustainability. Finally, Section 2.1.4 presents some most important challenges which need to be addressed in order to achieve Consumer Computing.

2.1.1 Motivation for Consumer Computing

The main motivation for introduction of Consumer Computing is the rising demand for different applications creation. Nowadays, consumers use different kinds of applications on their

personal physical devices such as desktop machines, tablets or smart-phones. Most recent technological innovations such as invention of tablets and smartphones have even more increased the demand for a new kind of applications called situational applications. Situational applications are novelty due to a fact that tablets and smartphones are a personal computers which can stand in a pocket and be carried everywhere to be used in different situations. The rising need for new application development is clearly demonstrated in a rapid growth of number of applications in different application stores. Table 2.1 captures the number of applications available in two most popular application stores, *Google Play* [34] and *App Store (iOS)* [35], for the period from 2009. till 2013.

Year	Number of applications		
	App Store (iOS)	Google Play	Total
2009.	25,000	2,300	27,300
2010.	150,000	30,000	180,000
2011.	400,000	250,000	650,000
2012.	650,000	600,000	1,250,000
2013.	1,000,000	1,000,000	2,000,000

Table 2.1: The number of mobile applications in two most popular application stores

It is obvious from the data presented in the table that the number of applications in stores is constantly growing which is closely related to the two facts: (*i*), technology is still improving and new physical features in modern devices are offering more and more possibilities for new applications, and (*ii*), consumers demand new kinds of applications on the market. The most recent research shows that the average number of applications per smart-phone in U.S. has grown from 32 in 2011 to 41 in 2012 (an increase of 28%), which indicates that demand for new applications is rapidly rising [36]. Another important indicator of even greater demand for new applications in future are the most recent projections of *U.S. Bureau of Labor Statistics* [37] on job openings in software engineering field. According to the projections, software developers occupation will have a growing rate of 32.4%, which places it among the occupations with the fastest growing rates for the period from 2010 to 2020. All above mentioned arguments and facts support thesis that average consumers should be given a chance to create their own applications and express their creativity.

Consumer Computing aims to endorse consumers to create their own applications by providing consumers intuitive tools and technology for application creation. For instance, the current approach for application development is to conduct a case study among consumers or collect

their feedback, and then to hire the professional experts in programming such as software developers to implement the applications with desired functionalities according to the studies. This approach reflects obvious inequality and divide in a today's digital world [2]. On one side of the digital world's gap are the applications developers which are clearly the minority group trying to serve all the needs for the applications consumers, which are the majority group on the other side of the digital world's gap. Consumer Computing is about to propose an alternative approach for application development by providing the sole consumers ability to create applications. In such an approach, the consumers choose which applications to create and actually drive the process of application development.Note that by letting consumer to create their applications, software developers still keep their role in the process of applications production. The general idea is to let consumers combine the existing applications into a new composite applications and implement more advanced functionalities. However, the basic application modules and components still need to be implemented by individuals formally educated in computer science and who have prerequisite programming skills.

2.1.2 State-of-the-art Technology for Consumer Computing

The previous section provides arguments and underlying motivation for Consumer Computing. However, another important question is the feasibility of Consumer Computing. In particular, how can the existing technology be utilized in order to support applications creation by the average consumers?

The recent technological innovation have led to creation of smartphones such as *iPhone*, *Google Phone* and *Android HTC*, which are a pocket *PC*-s that can access the Internet, process information almost as effective as desktop machines, and more important, their physical features and characteristics get constantly improved. The importance of smartphones appearance on the market is crucial for Consumer Computing. As already stated, a smartphone is a powerful machine that stands in a pocket, and consequently can be carried everywhere and can be used in many different situations, which demands new context-based applications development for smartphones (such as location-based, event-based, social-based etc). The similar things happened with the recent invention of tablets, a new set of tablet-specific applications were developed and offered to consumers (*iPad*, *iPad-mini*, *Samsung Galaxy Tab* etc). The smartphones and tablets present the applications in a very similar manner as tiles on the working surface. The most recent versions of web browsers also present commonly used web applications as tiles on

the working surface (*Google Chrome*, *Internet Explorer 10* etc). Finally, contemporary versions of popular desktop operation systems such as *Windows 8* and *Ubuntu Tablet* also utilize the tile-oriented style for applications presentation on a desktop surface.

The average consumers are using applications in a similar manner on different physical devices and getting familiar with tile-application look. Once the applications is started and activated, regardless of the hosting physical device, it provides its specific functionality, but still application's graphic user interface remains very familiar and intuitive to consumers. Consumers are familiar with plenty of features, such as adding new applications, removing the obsolete applications, running and using applications, and also consumers know how to use applications in chain. For example, it is often required to obtain the exact mailing address of the destination location, which can be obtain by using a *Search* application, before the address is utilized in a *Driving Directions* application.

All these mentioned facts and remarks indicate that the existing applications can be used as the basic building components for new applications creation in Consumer Computing. In fact, the existing applications are semantically intuitive to consumers and represent natural building blocks for composite applications development. The programming methodology which is used by consumers to create new applications in Consumer Computing is described in details in Section 2.2.

2.1.3 Additional Benefits of Consumer Computing

The most important benefit of Consumer Computing is related to the Consumer Computing motivation. The process of applications creation would become much smoother and more effective if consumers them self could create applications. This methodology would shorten the *time to market* period for the variety of interesting consumer applications. The advanced consumers would express their creativity by creating applications and bootstrap the other less motivated consumers, but also software developers as well, to create similar more efficient and more reliable applications. The most skilled individuals among consumers could create interesting applications that could be popular on the application market and gain profit for their's creators.

Another important benefit of Consumer Computing is related to the divide that is present in todays digital world [2]. The minority group in the digital world divide are the individuals formally educated in computer science with excellent programming skills. Those individuals

use the variety of creation tools and devices to provide the applications for the other group of the digital world divide. The other group in the digital world divide is consisted of consumers - the individuals without coding skills and formal education in programming, who simple consume the existing applications provided by software developers. The Consumer Computing idea would enable consumers to create applications on their own, which would eliminate the existing inequality that is present today in a digital world and bridge the gap between consumption and creation.

The additional benefit of Consumer Computing is related to the transformation of each individual's tacit knowledge into a collective procedural knowledge stored in created consumer applications. For instance, if a domain specific experts create their own applications to solve domain specific problems in their field, the respectable knowledge those people posses gets archived and stored as part of saved applications they create. In such manner, Consumer Computing enables the individual's tacit knowledge to be transferred to other consumers who can reveal that knowledge by simple using applications certain individuals created. Although the greatest potential lies in a domain-specific knowledge, the tacit knowledge of each average consumer should not be underestimated. In fact, most creative average consumers may improve specific use case applications such as *Trip Planner*, *Party Organizer* etc., due to their familiarity with specific topics, and offer better applications than the one that are available, or even create some specific use case applications that are not available yet.

There are plenty of more side-effect benefits of Consumer Computing such as encouraging ordinary people to use modern technology, letting people to express their creativity by automating every day tasks as part of the applications they create, enabling average consumers to adopt basic programming skills which could be helpful and beneficial while organizing different real life activities etc.

2.1.4 Challenges in Consumer Computing

The goals, motivation and ideas of Consumer Computing are definitely interesting and sounds promising for improvement of todays digital world. However, there are still challenges that need to be addressed in order to achieve Consumer Computing.

For instance, to achieve digital world *equality*, each consumer should be given a chance to create applications, which is a very difficult task. In fact, the appropriate consumer intuitive technology needs to be offered. The recent technological improvements such as invention of

tablets, smartphones, modern web browsers and most recent versions of popular desktop operating system with their similar tile-representation of applications sound promising, but still the appropriate methodology for application creation should be proposed. Note, however, that the technology is still improving and another even more consumer intuitive device, a *Google Glass* [38], is on the horizon. Perhaps a *Google Glass* can offer a better way to compose the existing applications in a composite process to support more advanced features.

Another important challenge of Consumer Computing is related to the digital world *sustainability*. The appropriate technology and suitable methodology for consumer applications creation are not sufficient to achieve sustainable application development. According to their structure, consumer applications are component based systems. Besides having a *know how* to compose the existing applications, consumers still need help to find the appropriate functional components and connect them correctly. Hence, consumers need different types of assistants that help them during the process of application creation. The role of consumer assistants is to output real time recommendations during the application creation. For instance, there should be an assistant that analyses current application structure, compares it with the existing applications contained in the archive, and proposes consumer the next step in development process such as which component applications to include, which parts in the existing applications to connect etc. In addition, there should be assistant that analyses the current application's components, finds other consumers that used those components, and suggests the active consumer a list of consumers that could be helpful with their advices during the application creation process. There should be also assistants that help consumer to optimize nonfunctional properties of the composite application. In fact, besides the appropriate logical functioning, nonfunctional application's properties such as reliability, availability, security etc., highly impact the perceived quality of the application.

Finally, another important aspect that needs to be considered for effective Consumer Computing is the *scalability* of the digital world. As already stated, consumer applications are component based systems, where the existing applications are used as basic components for new composite applications creation. In addition, new composite applications can be further used as basic components for some other composite applications, which poses the challenge of scalability of consumer applications in Consumer Computing. Although such an hierarchical application composition, where each application may be consisted of several applications on each level of composition seems practical for applications creation, it questions the possibility

of effective real time execution [39]. To address this drawback and overcome scalability issues, new composite applications need to be packed tightly and deployed as homogeneous systems, or alternatively, their execution should be optimized and performed in the cloud.

2.2 Programming Methodology in Consumer Computing

There are several aspects that define each programming methodology as described in [28]. In order to present the programming methodology in Consumer Computing within this dissertation, only most important characteristics are considered: *programming elements*, *programming language*, *programming technique* and *application's software architecture*. *Programming elements* are the basic building blocks that programmers, or individuals who create computer programs and applications, use to implement desired functionalities. For instance, programmers who prefer *Object Oriented Programming* (*OOP*) programming methodology use objects as basic building blocks to create their computer programs. *Programming language* is an artificial language that programmers, or other individuals who create computer programs and applications, use to instruct the behavior of their programs to the computer. For example, the programmers who prefer already mentioned *OOP* programming methodology usually use general purpose languages such as *C++*, *Java* or *C#*. *Programming technique* defines the procedure how to transfer a sequence of operations or tasks representing computer program or application to the computer. For instance, experienced programmers usually directly write code instructions to transfer the program to the computer. Software architecture denotes a high level structure of a certain software artifact. However, some problems are commonly occurring in software architecture, which isolates general and reusable solutions as architectural patterns [40]. There are plenty of architectural design patterns such as *component based*, *layered*, *client-server*, *data driven*, *event driven* etc. The architecture of applications in Consumer Computing follows component based design pattern, and it is described in details in Section 2.4.

The rest of the section is organized as follows. Section 2.2.1 explains the programming elements used by consumer in Consumer Computing to create applications. Section 2.2.2 describes the programming language while Section 2.2.3 presents the programming technique used in Consumer Computing.

2.2.1 Programming Elements in Consumer Computing

Contemporary consumers use various types of applications on different sophisticated devices such as smartphones, tablets, smart TVs etc. In addition, all those modern devices present applications in a similar manner as tiles on the working surface. Consumer are very familiar with application's graphic user interface which provides simple representation of application's functionality and also hides the implementation details from consumers. Thus, consumers are familiar with applications at least as much as the professional developers are familiars with the programming elements they use. For instance, consumers know how to add, remove, use and combine different applications on their physical devices. Further, consumers often use different applications in chain in order to achieve some additional functionalities. All these mentioned arguments substantiate reasoning that the existing applications are the ideal choice for programming elements in Consumer Computing.

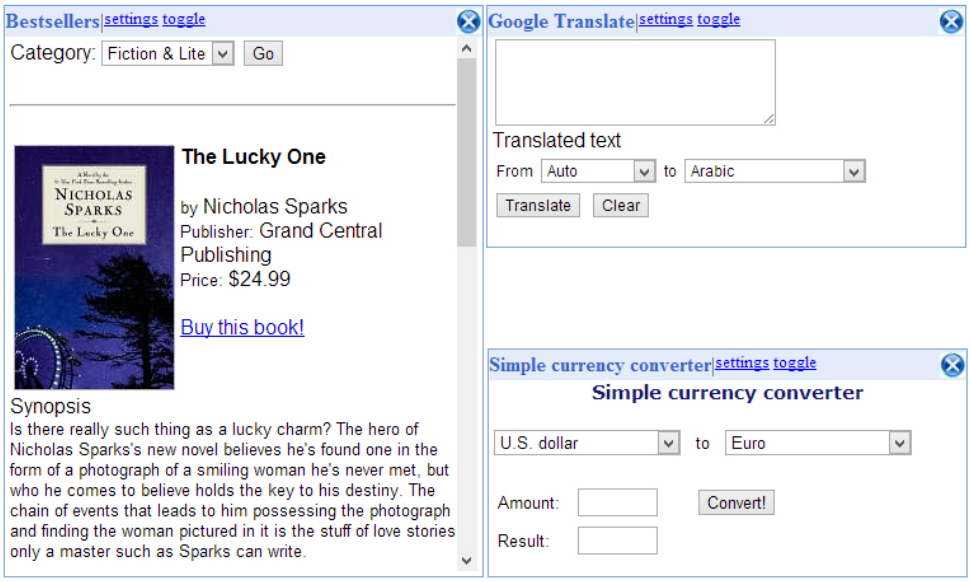


Figure 2.1: Software widgets presenting applications in *Geppeto*

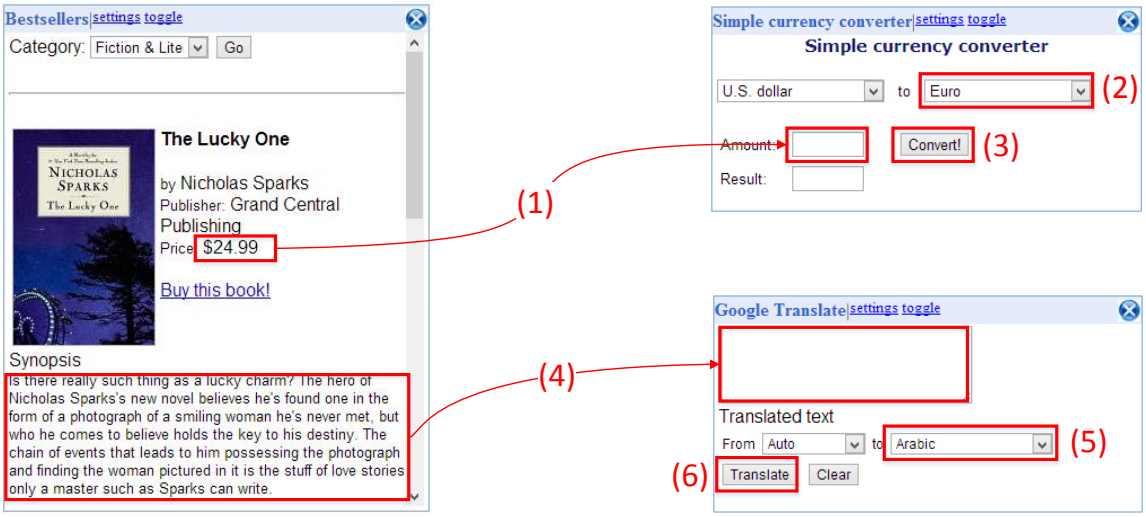
As already stated, the researchers at the *Consumer Computing Lab* have implemented a tool *Geppeto* [33], which is so far the closest implementation of Consumer Computing. *Geppeto* is a web tool that enables consumers to create their own applications by composing the existing applications. Applications in *Geppeto* are software widgets that provide their various functionalities through a simple unified graphic interface consisted out of a set of controls supported in *HTML* language [41]. Software widgets and *HTML* controls were chosen for applications representation due to facts that software widgets are semantically intuitive to consumers who are very familiar with web browser's *GUI* controls. Figure 2.1 presents three software widgets

representing applications in *Geppeto*. More detailed description of programming elements in Consumer Computing can be found in [28]. The following sections describe the programming language and technique utilized in Consumer Computing.

2.2.2 Programming Language in Consumer Computing

General purpose programming languages used by professional developers for computer programs creation are too abstract and difficult for average consumers. In order to provide consumers the ability to create applications, consumer intuitive and familiar programming language need to be used. In fact, in his thesis [28], Skvorc introduces the principles of *equality of consuming and programming*. The principle states that consumers need a programming language that employs the programming paradigm that is not different than the paradigm which is employed while using the programming elements. Hence, to connect the programming elements in Consumer Computing, the language used for consumption of programming elements needs to be used. The language of programming elements consumption is consisted out of GUI level actions that consumers perform to instruct the behavior of applications such as *type the text into the input field*, *press the control to trigger certain process*, *transfer the data from one field to another* etc.

In the most successful Consumer Computing implementation, *Geppeto*, the programming language is consisted out of GUI level actions that can be performed over the *HTML* web controls. In such manner, so far supported GUI level action are: *Type In*, *Copy*, *Paste*, *Check*, *Uncheck*, *Select*, *Click* and *Double Click*. These actions enable various manipulations with the software widgets and provide fully support for composite applications creation process. For instance, let the reader consider a composite process containing the existing programming elements presented in Figure 2.1. The widget *Bestsellers* provides the list of most popular books in a selected category containing each book's basic information such as title, author, publisher, price in US dollars and a brief book's synopsis in English. The consumers in Croatia would prefer the book's synopsis to be displayed in Croatian language and book's price to be in Croatian currency. In order to achieve this preferences, the average consumers can use two other software widgets: *Google Translate* and *Currency Converter*. As depicted in Figure 2.2, consumers can achieve the desired preferences by performing a list of simple GUI level actions on the widgets. Note that average consumers are very familiar with this type of actions. In fact, consumers perform these actions everyday while using different web applications, which makes



Consumers need to perform the following list of simple GUI actions:

(1) Copy the price in US dollars on the *Bestsellers* widget and paste it into the input field on the *Currency Converter* widget
(2) Select the Croatian currency on the checkbox on the *Currency Converter* widget
(3) Click the *Convert!* button on the *Currency Converter* widget
(4) Copy the synopsis text on the *Bestsellers* widget and paste it into the input field on the *Google Translate* widget
(5) Select the Croatian language on the checkbox on the *Google Translate* widget
(6) Click the *Translate* button on the *Google Translate* widget

Figure 2.2: Programming language consisted out of GUI actions used in *Geppeto*

them a reasonable choice for programming language in Consumer Computing.

2.2.3 Programming Technique in Consumer Computing

The most common programming technique used by professional developer is writing code instructions in a dedicated development environment such as *Eclipse* or *Microsoft Visual Studio*. Note, however, that such an programming technique is too complex for average consumers because of numerous rules and constraints that need to be followed while writing code. Obviously, writing code instructions is not likely to be adopted for application creation in Consumer Computing. In Consumer Computing, consumers need a programming technique that requires no more concentration than using the existing applications. There are few possible options that accommodate these conditions such as *Programming by Demonstration* [42], *Programming by Example* [43] or *Visual Programming using GUI* [44]. Another benefit of these programming techniques is that they are less error prone than classical programming technique such as writing code instructions.

In *Geppeto*, the *Visual Programming* technique is used. Consumers define GUI actions

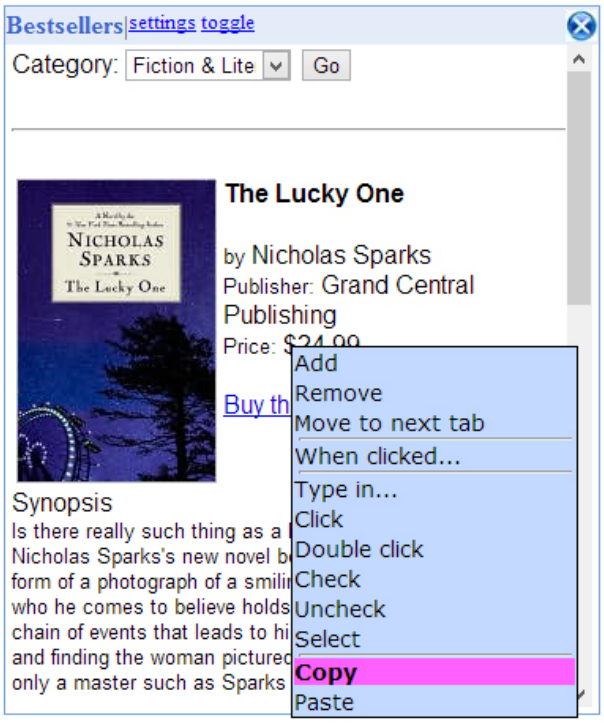


Figure 2.3: *Floating context menu* programming technique used in *Geppeto*

on widgets controls by using a special *floating context menu* as depicted in Figure 2.3. The floating context menu gets activated once a consumer performs a right mouse button click over a certain control. The activated context menu provides a list of available GUI actions that can be automated as part of the consumer application. By repeatedly defining actions for different controls, consumer defines control and data flow for the consumer application.

2.3 Consumer Computing Environment

The previous sections introduce the motivation for Consumer Computing and describe used programming methodology which arguments the feasibility of Consumer Computing. In this section, the entities that assure digital world's equality, sustainability and scalability in Consumer Computing are presented. As already stated, to enable sustainable applications development, various assistants that help consumers during the creation process need to be developed. In addition, to enable composite applications development and tackle consumers to create applications, the variety of basic components applications need to be developed by professional developers. The intuitive programming methodology is not sufficient to fully achieve Consumer Computing. Another important requirement for scalable composite applications development is related to the consumer composite applications transparency. The composite applications

created by consumers need to provide their functionalities through the same interfaces as basic component applications. Additionally, composite applications need to be available for further compositions as basic components for some other applications as well.

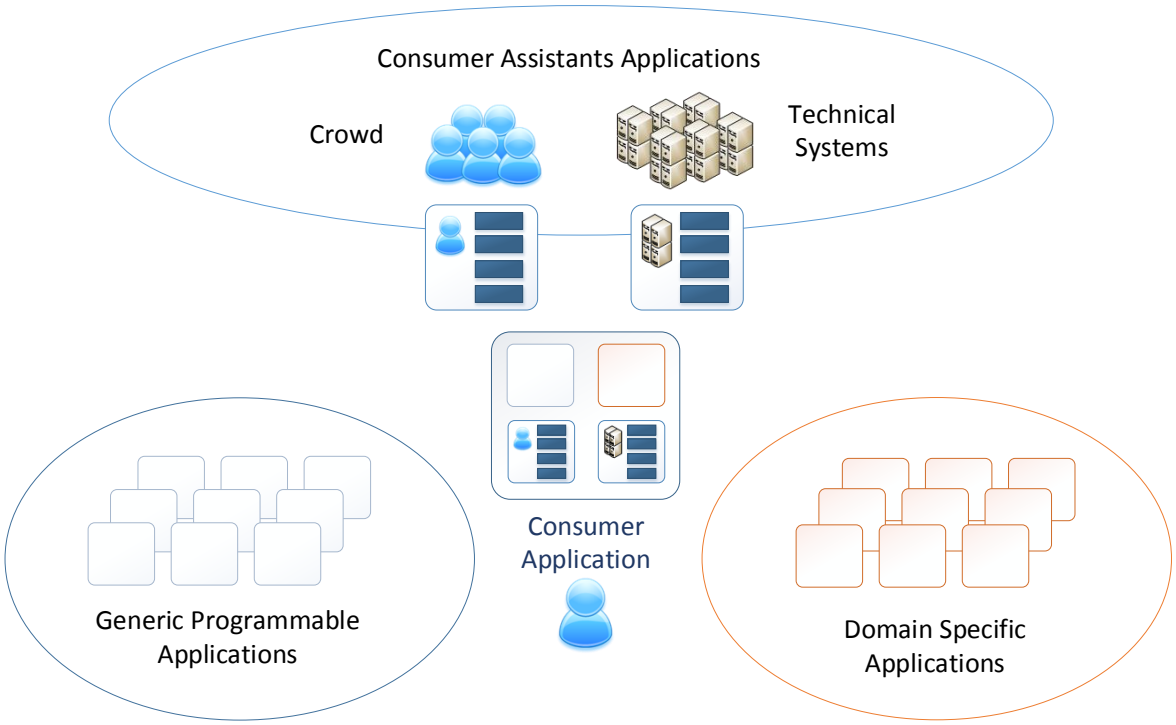


Figure 2.4: Consumer Computing environment

The Consumer Computing environment containing different entities which assure Consumer Computing feasibility is presented in Figure 2.4. As can be seen in the figure, three different types of applications assure completely supported Consumer Computing environment: *Consumer assistants applications*, *Generic programmable applications* and *Domain specific applications*. Section 2.3.1 describes domain specific applications which are used as basic functional components for new composite applications creation. Section 2.3.2 explains generic programmable applications that are used to create, encapsulate and deploy composite application's logic into a new application. Finally, Section 2.3.3 overviews consumer assistants applications that provide help for different functional and nonfunctional composite application's properties adjustment during the process of application creation.

2.3.1 Domain Specific Applications

Domain specific applications are determined by their functional properties and they are used as basic functional components for new applications creation. Consumers compose the

existing domain specific applications in order to support more advanced desired features using programming methodology described in Section 2.2. Composite applications expose their functionalities in the same manner as the existing applications which enables sustainable growth of domain specific applications base.

In *Geppeto*, there exist several domain specific applications sets implemented to demonstrate the appliance of Consumer Computing in different fields. For instance, there exist a special set of software widgets intended for planning and controlling the *ROV* during autonomous underwater diving missions. Another interesting set of domain specific applications developed for analysis of climate and oceanic data provided by the *National Oceanic and Atmospheric Administration* (*NOAA*) is depicted in Figure 2.5. There are plenty of other specific sets of applications implemented such as set for statistical data analysis, set for studying the financial stocks data, set for chemistry, set for math etc.

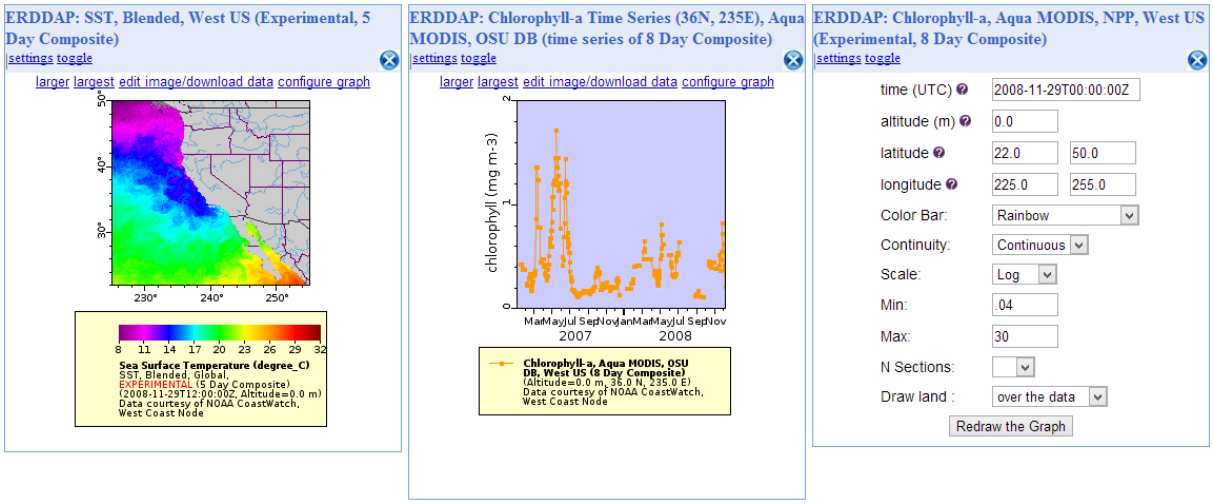


Figure 2.5: Domain specific widgets for *NOAA* data analysis

The general idea is to provide as much as possible domain specific sets of applications to employ Consumer Computing in different fields. These sets should be initially consisted out of basic functional components designed considering domain specific experts' advices, and implemented by professional software developers. Using such developed basic functional components, consumers with special interest for specific domains should create more advanced composite applications according to their needs.

2.3.2 Generic Programmable Applications

In Section 2.2, the programming methodology for application creation in Consumer Computing is described. However, the detailed explanation of application's logic encapsulation and new application deployment is not provided. In order to support sustainable applications development and increase of domain specific applications base, new composite applications need to be deployed using the same technology as the existing applications. In fact, new composite applications need to provide their functionalities using the same interfaces as the existing applications do. To accommodate these requirements, generic programmable applications are used. Before they start defining control and data flow for their composite applications, consumers create a new, blank generic programmable application, in which data flow, control flow and user interfaces for the new application are stored.

Figure 2.6: Generic programmable *Geppeto TouchMe* widget

For instance, there are few different types of generic programmable software widgets supported in *Geppeto*. The most commonly used generic programmable widget is *Geppeto TouchMe*, a widget in which a new composite application's logic is encapsulated and deployed. The widget also supports definition of GUI controls representing input and output fields for a new application. The composite application encapsulated in a *Geppeto TouchMe* that implements the example scenario shown in Figure 2.2 is presented in Figure 2.6. The name *TouchMe* indicates that the new application's execution is triggered by consumer when pressing the programmable control. In this case, the application is triggered by pressing the button *Go*.

There are more generic programmable widgets supported in *Geppeto*, such as *Geppeto TriggerMe* which is triggered by signaling particular external event, *GeppetoTickMe*, a type of a programmable widget that is triggered at the specific time, *Geppeto LocateMe*, a special widget intended for mobile devices triggered once a device is found in a predefined area, *message-triggered* programmable widgets that support *synchronization* and *communication* among different consumer applications (introduced by Miroslav Popovic and described in details in his PhD thesis [29]), etc.

2.3.3 Consumer Assistants Applications

The consumer oriented programming methodology is not enough to bootstrap the massive consumer applications creation. The process of appropriate component selection sometimes can be very difficult task even for the experienced professional software developers. First, it is crucial to select the functional components that accommodate the requirements coming out of consumer application's desired behavior. Even when exact desired behavior of the consumer application is accomplished, the application may be suboptimal when nonfunctional properties such as reliability, availability, security etc. are considered. In fact, even professional software development environments such as *Eclipse* or *Microsoft Visual Studio* nowadays help developers by activating various wizards that lead them through the process of application creation. Further, one of the requirements for Consumer Computing feasibility is related to the substantially large number of various domain specific applications which emphases the problem of appropriate component application selection.

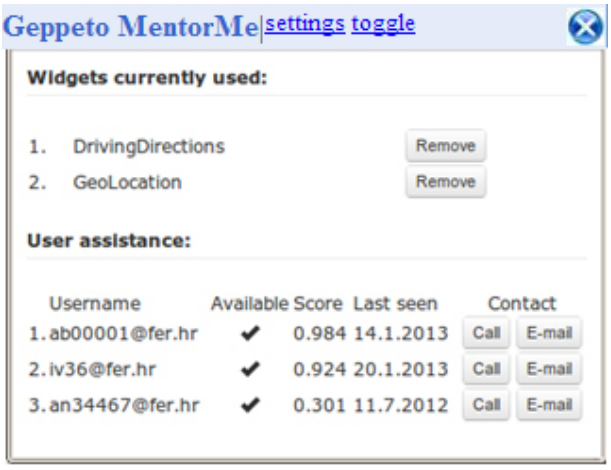


Figure 2.7: *Geppeto MentorMe* assistant widget

In order to help consumers during the process of application creation, a special entities, con-

sumer assistants applications, are introduced in the Consumer Computing environment. These elements are the primary focus of most recent *Consumer Computing Lab*'s research activities. Consumer assistants are intended to help consumers during the process of selection of domain specific applications. For instance, a special consumer assistant application helps consumer to select the next component in the application creation process by analyzing current state of the application and comparing it with composite applications archive [30]. Another interesting assistant, already implemented in *Geppeto*, *Geppeto MentorMe* [31], analyzes the current state of the application and finds the list of consumers that have already created similar applications and whose advices might be beneficial and helpful during the application creation. The *Geppeto MentorMe* assistant is depicted in Figure 2.7.

So far mentioned assistants are considering exclusively correct functioning of the applications. However, application's nonfunctional properties such as reliability or availability also influence the perceived quality of the application. Thus, assistants that help consumers optimize different nonfunctional aspects of their applications should be proposed.

2.4 Architecture of Applications in Consumer Computing

The primary focus of this dissertation is prediction of consumer applications reliability which is a nonfunctional property of a software system. Some of the most commonly considered nonfunctional properties of software systems are: fault tolerance, backward compatibility, extensibility, maintainability, reliability, availability , security, usability etc. In order to analyze nonfunctional properties of a certain software system, it is crucial to understand the architecture of the respected software system. The classical old-fashioned approaches in software architecture design were usually driven by functional requirements and data flow of the system [45]. Besides the correct functional operation, nonfunctional qualities are very important for contemporary software systems. The system's architecture highly impacts its nonfunctional properties and that is the reason why modern quality-driven approaches in software design synthesize software architecture of the system out of previously defined nonfunctional properties requirements [46].

When consumer applications architecture is considered, there should be distinguished two types of applications according to their structure: *composite* and *atomic* consumer applications. The architecture of atomic consumer applications is presented in Section 2.4.1. Composite

applications are built out of atomic applications and their architecture is briefly presented in Section 2.4.2.

2.4.1 Architecture of Atomic Consumer Applications

The atomic consumer applications provide basic domain specific functionalities and they are usually developed by professional developers. They are considered atomic and homogeneous due to fact that there is no additional information about their inner structure or organization. These applications provide their functionalities through a simple consumer intuitive graphic user interface and hide the implementation details under the hub. However, they often require some data retrieval and information processing on the server side over the unpredictable Internet.

In *Geppeto*, so far most closest Consumer Computing implementation, the atomic applications are implemented as software widgets. Software widgets provide their functionalities via consumer intuitive graphic user interface consisted out of *HTML* supported web browser's controls. Part of the widget's code is executed locally on the hosting machine within the browser, but most widgets usually contain the other part of the code that requires server side data processing or retrieval over the Internet. In such manner, this other part of the widget's code can be viewed as dynamic software artifacts such services described in *Service Oriented Architecture* (*SOA*) [47, 48].

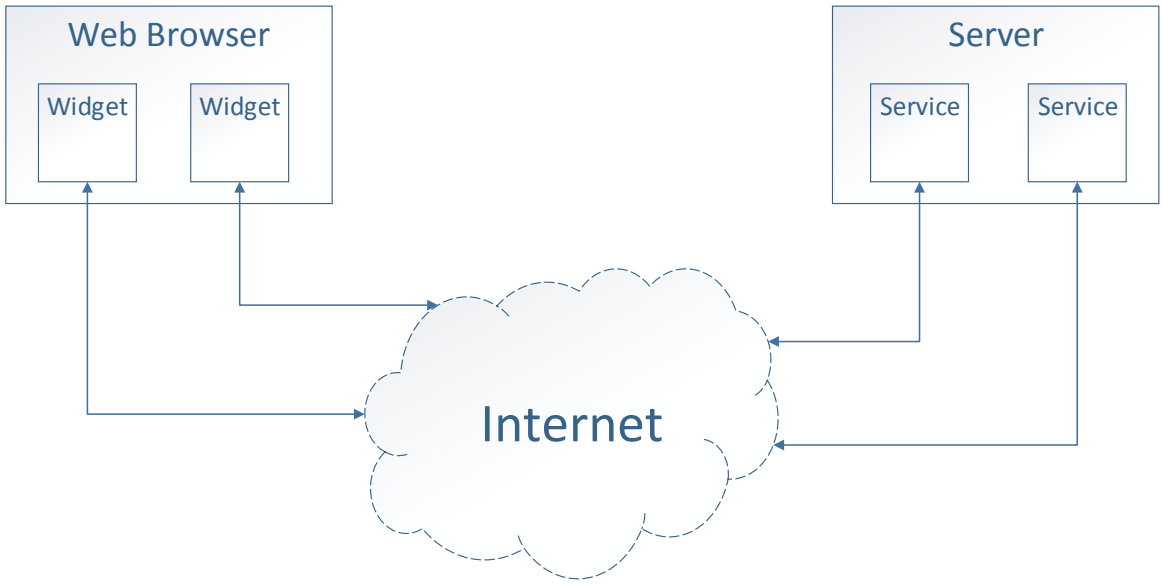


Figure 2.8: Architecture of Atomic Consumer Applications

This widget's service-characteristics are very important, especially for consideration of non-functional application's properties in Consumer Computing. The part of the code that is executed locally on the hosting machine is considered less interesting in terms of nonfunctional qualities modeling than the other part of the code that is executed over the Internet due to Internet's highly dynamic and unpredictable features. Accepting all afore mentioned arguments and requirements, the architecture of atomic applications in Consumer Computing can be presented as depicted in Figure 2.8.

2.4.2 Architecture of Composite Consumer Applications

Composite consumer applications are built out of atomic consumer applications that are orchestrated and coordinated in order to provide the additional functionalities. In general, composite applications are created by consumers and their functionality is encapsulated into a new applications using a dedicated type of a generic programmable applications which are described in Section 2.3.2. The composite applications encapsulation enables further utilization of such applications as basic basic components for more composite applications creation. In this manner, composite applications may be consisted out of atomic and composite applications on lower composition levels.

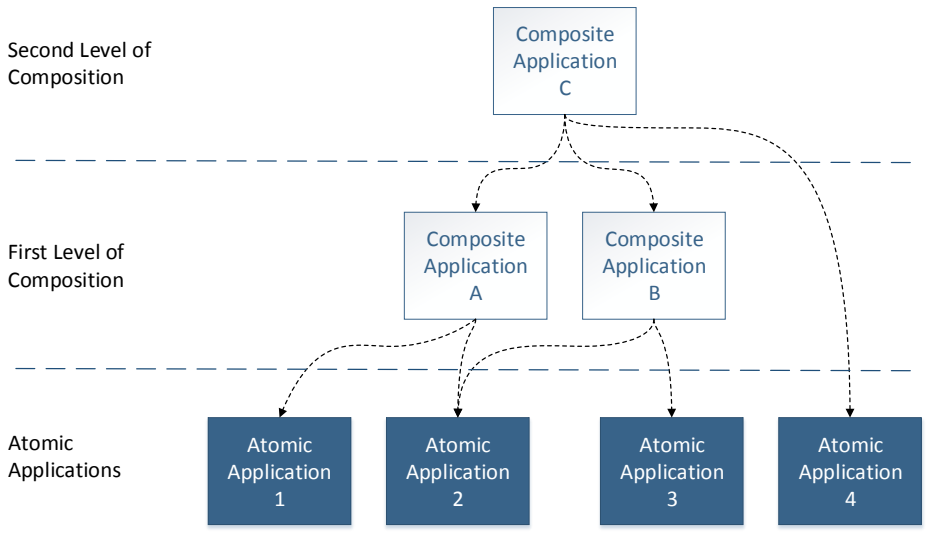


Figure 2.9: Architecture of Composite Consumer Applications

The architecture of composite applications is presented in Figure 2.9. The figure also describes the process of applications composition, which is organized hierarchically. Hence, composite applications may appear at different level of composition. For instance, composite ap-

plication *A* at the second level of composition is consisted out composite applications *B* and *C*, and atomic application *4*. Composite application *B* at the first level of composition contains atomic applications *1*, *2*. Note that each application, regardless of being atomic or composite, can be contained in more than one composite application. For example, atomic application *2* is contained in composite applications *B* and *C*.

Any additional information about internal architecture of some software artifact can be very important and helpful when assessing its nonfunctional properties. The next chapter is focused on reliability modeling of applications in Consumer Computing. Although the thesis of this dissertation is primary focused on reliability modeling of atomic applications (note the application is considered atomic in case no information about its internal structure is available), the reliability of composite applications Consumer Computing is briefly presented.

Chapter 3

Reliability in Consumer Computing

The previous chapter describes Consumer Computing, a novel methodology that enables consumers and encourages them to create their own component based applications. The basic components for composite application creation in Consumer Computing are the existing applications. In order to implement additional functionalities, consumers compose the existing applications and create new composite applications that can be used later for further compositions. It is extremely important for consumer to select the appropriate functional components while creating a new application. In fact, in case consumer selects the wrong functional components it is likely that the applications will not operate according to the desired behavior. To address this potential drawback, the consumer assistant applications were introduced in Consumer Computing environment. Consumer assistants are implemented as wizards that lead consumers and help them during the process of application creation.

However, another important aspect that should be considered in component based systems is the optimization of nonfunctional application's properties. Nonfunctional properties such as reliability, availability, security, etc. might have a significant impact on perceived quality of the application. More specifically, nonfunctional properties are crucial for efficient execution of component based applications. Thus, during the process of basic components selection and composition, the optimization of nonfunctional application's properties should be incorporated as well. The research in this dissertation is studying reliability of applications in Consumer Computing. When modeling reliability in component based systems, there should be distinguished two types of applications: atomic and composite applications. As already stated in Section 2.4.1, the applications in Consumer Computing can be considered as dynamic software artifacts such as services and SOA model can be soundly applied for studying reliability of con-

sumer applications. There has been proposed the variety of approaches for composite services reliability modeling, but only few models consider reliability of atomic services.

The main topic of this research is to asses the reliability of atomic services underlying atomic consumer applications. It is very challenging to assess the reliability of the service because of its dynamic nature. Services are software artifacts whose nonfunctional properties i.e. reliability change depending on *service invocation context*. The service invocation context is determined by the set of *user*–, *service*– and *environment*–specific parameters. To address this challenge, leveraging users and providers feedback should be utilized to collect partial but comprehensive past invocations sample, and then, prediction methods should be used to estimate the reliability of atomic services.

Finally, the last but certainly not the least important aspect that needs to be taken into consideration is consumer's perception of reliability. Consumers need a simple and intuitive way to analyze and optimize applications reliability. Hence, there should be a special dedicated consumer assistant widget intended for application's reliability optimization during the process of components selection. The assistant should provide consumers help and information about the reliability of components they consider as selection candidates for composition.

The rest of the chapter is organized as follows. Section 3.1 provides basic theoretical software reliability concepts and taxonomy. Section 3.2 analyzes Consumer Computing applications from the aspect of their execution and justifies the adoption of SOA model while assessing reliability in Consumer Computing. Section 3.3 presents specific challenges while modeling reliability in service oriented systems. Finally, Section 3.4 describes the system used for reliability prediction of Consumer Computing applications according to the adopted SOA model.

3.1 Software Reliability Basics

According to the commonly adopted taxonomy brought in [49], the reliability is one of the aspects of broader dependability area. Other aspects of dependability are availability, safety, maintainability and integrity. The reliability is related to the continuity of a correct service delivered by a system. There are two most common definitions of reliability that can be found in literature:

(i) the probability that the system performs its required functions under stated conditions for a specified period of time [3] and,

(ii) the probability that the system successfully completes its task when it is invoked (this definition is known as "reliability on demand" in literature) [50].

As stated in [51], the definition (i) refers to the "never ending systems" that must operate correctly during the entire given mission (e.g., the on-board flight control system of an airplane should not fail during the entire flight). The definition (ii) is related to the systems offering services that, once invoked, must successfully complete response. Both of these definitions can be applied to any software systems regardless of its granularity level (e.g., distributed software artifact, software component, software service etc.). It is only required that the correct behavior of the given software system can be unambiguously defined. A correct behavior is a "failure-free" behavior, in which the system produces the expected output for any input that accommodates system's specifications and requirements.

"Failure-free" behavior is only related to what can be observed at the system output, which means that the system might experience a degree of incorrect functioning without demonstrating any failure at the output. In order to understand and explain these claims, three fundamental reliability concepts are introduced: *fault*, *error* and *failure*. A fault is a wrong statement introduced somewhere in the software system. An error is an unexpected state in which the system might get upon it executes a fault statement (e.g., a local variable gets assigned an unexpected value). A failure is the propagation of an error up to the system output (e.g., an output variable gets assigned an unexpected value).

Even if the fault is introduced somewhere in the systems, that does not mean that the system will encounter an error. Hence, the system might not execute the fault statement for a long period of time, depending on the structure of the code or sequence of inputs given to the system. In the case the system gets in an erroneous state, it does not imply that the failure will be manifested at the system output. For instance, the error may be masked thanks to some other operations executed between the erroneous state and the output production.

Different classification of failures are present with the respect to different attributes. According to the failures manifestation they can be classified as follows:

- Regular failure – A failure that manifests itself as an unexpected value at the system output.
- Crash failure – A failure that causes immediate stop of the system operation; this type of failure is common to systems known as *fail-and-stop* systems.

- Looping failure – A failure that prevents the system from producing any (correct or incorrect) output; this type of failure is unpleasant due to the fact that it takes time to notice that the system is experiencing such a failure.

According to their severity, failures can be classified as follows:

- Repairable failure – A failure that can be repaired without restarting the whole system.
- Unrepairable failure – A failure that requires the restart of the system in order to restore its correct behavior.

These classifications are important in the terms of understanding which types of failures are allowed to be experienced by the system in certain reliability models. As can be expected, the less restrictive the requirements on types of allowed failures are, the more complicated is the model formulation. The reader should note that the attributes specified in different classifications are not independent of each other. For example, a crash failure is obviously not repairable.

3.2 Adoption of SOA Model in Consumer Computing

The architecture of applications in Consumer Computing was described in Section 2.4. As already stated, the crucial part of the code of consumer applications gets executed over the Internet, which enables modeling of consumer applications as dynamic software artifacts that provide their functionalities on the Internet such as services described in SOA [47,48]. This section justifies the adoption of SOA model while considering nonfunctional properties of applications in Consumer Computing. In order to justify the adoption of SOA model, the execution process of Consumer Computing applications needs to be studied carefully. As described in Section 2.4, there should be distinguished two kinds of consumer applications: atomic and composite applications. Also, this section presents the composing methodology used in Consumer Computing as a simple extension of service composition tools used in service-oriented computing.

The architecture of atomic consumer applications is described in Section 2.4.1. Atomic consumer applications are executed as separate frames within the web browser hosted on consumer's machine. However, each consumer application contains dynamic part of the code that communicates with remote resources on the Internet. For instance, consumer applications often require some data retrieval or information processing which is usually achieved by performing dynamic invocations to the remote services on the Internet. Hence, regarding their execution

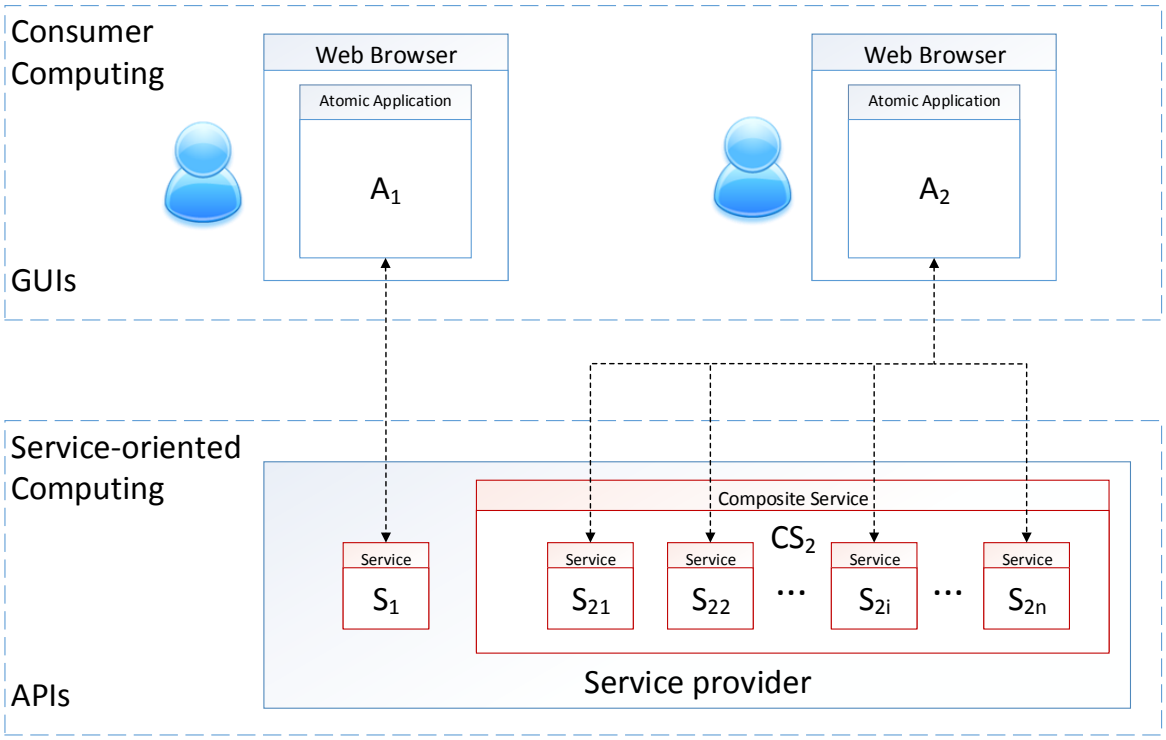


Figure 3.1: The process of execution of atomic consumer applications

manner, atomic consumer applications can be viewed as dynamic software artifacts that rely on one or more remote services on the Internet as presented in in Figure 3.1. Based on this assumption, reliability of a particular atomic consumer application is determined by the reliability values of services underlying the associated consumer application.

Since atomic consumer applications usually provide basic functionalities they can be seen as the user interface extensions that provide graphic representations for the underlying services. Hence, in most common case, each atomic consumer applications is corresponding to a single service on the Internet, and it is certainly justified to associate the reliability of the consumer application to the reliability of the underlying service. For instance, as depicted in Figure 3.1, atomic consumer application A_1 is relying on the service S_1. In the second case, atomic consumer application is depending on more different services. For instance, atomic consumer application A_2 is depending on services S_{21}, S_{22}, ..., S_{2i}, ... and S_{2n}. In general, if such an application fails, it is not possible to exactly identify the failure reason from the consumer's perspective. The application failure can be caused by the failure of any of services S_{21}, S_{22}, ..., S_{2i}, ... or S_{2n}. However, services S_{21}, S_{22}, ..., S_{2i}, ... and S_{2n} can be viewed as a composite service CS_2 and the failure of the application A_2 can be addressed to the failure of the composite service CS_2. From the consumer's perspective, such a composite service can be seen

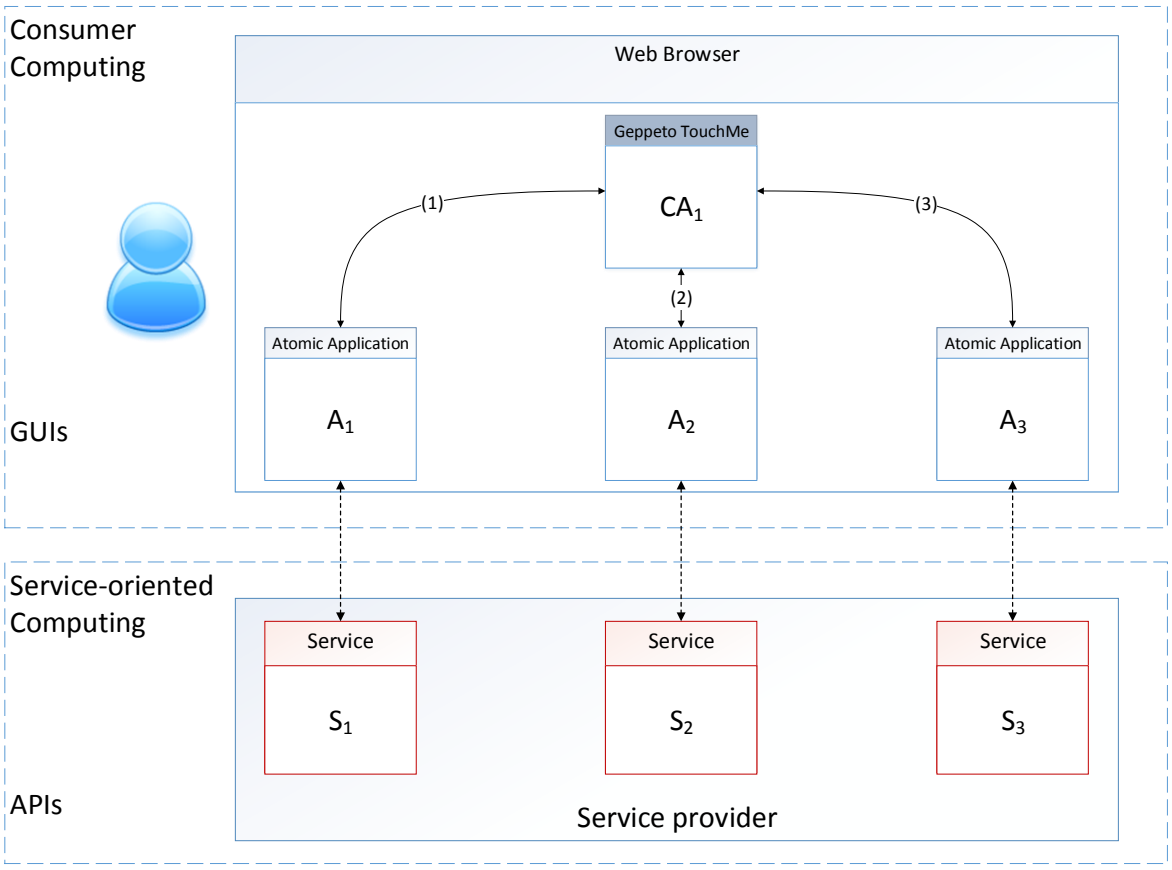


Figure 3.2: The process of execution of composite consumer applications

as an atomic service whose internal structure and dynamic execution properties are generally unknown. Thus, for the second case, it is also justified to correlate the reliability of the atomic consumer application A_2 to the reliability of the service CS_2.

Although composite consumer applications are not the primary topic of this dissertation, SOA model needs to be applicable for their execution as well in order to fully justify its adoption in Consumer Computing. There is a separate research dedicated to the reliability improvement of composite consumer applications [32], which is conducted as part of the research activities in the *CCL* lab. Composite consumer applications are built out of more atomic consumer applications in order to support more advanced functionalities. As depicted in Figure 3.2, composite consumer application CA_1 is comprised out of atomic consumer applications A_1, A_2 and A_3 each relying on its service S_1, S_2 and S_3 respectively. Composite consumer applications are executed in the web browser by performing GUI level actions defined by consumers while creating composite applications. The logic of the composite application is encapsulated in a dedicated generic programmable widget (see Section 2.3.2) as a set of GUI level actions that needs to be performed in order to support the composite application's functionality. It is impor-

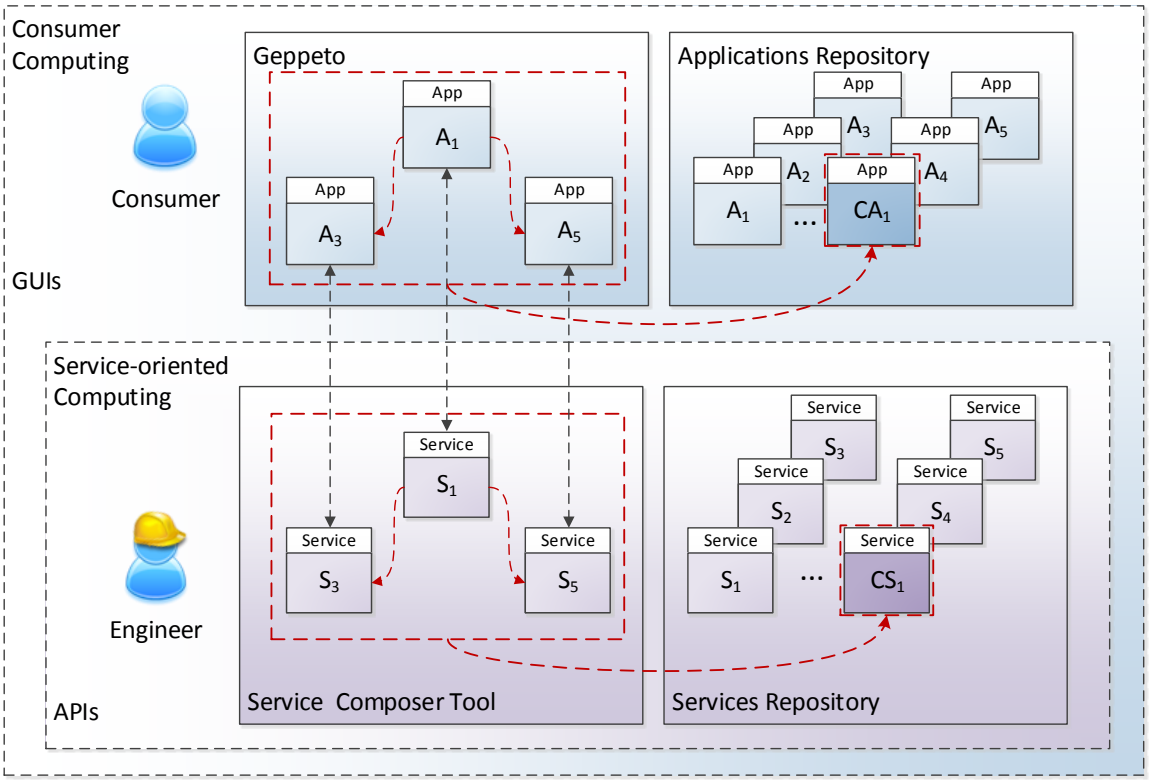


Figure 3.3: Consumer Computing as an extension of service oriented computing

tant that each atomic application (being part of the composition) needs to be present within the browser for the composite application to function correctly. In fact, composite application simply performs orchestration and coordination by executing GUI level actions in the web browser. For instance, composite application CA_1 invokes application A_1, waits for it to process its task, then invokes application A_2, waits for it to process its task, and finally invokes application A_3. Since all of the comprising atomic applications rely on their respected services, it is obvious that the reliability of the composite consumer applications is depending exclusively on reliability of services underlying atomic consumer applications (assuming the orchestration and coordination mechanisms are implemented correctly). Hence, it is justified to correlate the reliability of the composite consumer application to the reliability of services underlying comprising atomic applications.

Knowing the execution process of consumer applications, the composing methodology in Consumer Computing can be seen as an consumer oriented extension of the service composition process which is used in service oriented computing. As presented in Figure 3.3, in service oriented computing an engineer creates a new service CS_1 that supports the additional functionality by composing the existing services S_1, S_3 and S_5. The services are composed by

manipulating the *API*s using special service composition languages. The new service is then deployed on the Internet and added into the services repository and it can be used for further compositions. On the other hand, in Consumer Computing, consumer creates a new application CA_1 out of the existing applications A_1, A_3 and A_5 by defining the set of *GUI* level actions that need to be performed to support the desired functionality. The new application's logic is then encapsulated in a special generic programmable widget which is added to the applications repository and which can be used for further compositions. Consumer Computing can be viewed as an extension for service oriented systems development. In fact, while consumers create new consumer applications without having any knowledge about programming or *API*s, the system hides the implementation details from consumers and actually creates new service compositions beneath.

To summarize, the arguments presented in previous paragraphs uphold the efforts to soundly adopt the SOA model for consideration of reliability of applications in Consumer Computing. Hence, in further sections the term consumer application and service are interchangeably used when nonfunctional properties are considered, due to a fact the each consumer application can be seen as a special representation of the relying service. According to the adopted model, the next section provides some specific challenges that are present while modeling reliability in service oriented systems.

3.3 Reliability Challenges In SOA

The arguments that justify sound adoption of service oriented model for reliability consideration of consumer applications are presented in the previous section. According to the adopted model, each consumer application can be considered as a dynamic software such as service that provides its functionality on the Internet. However, by adopting service oriented model, special challenges that are present while modeling reliability in service-oriented systems are introduced. Service-oriented systems posses specific characteristics that make them unique when nonfunctional properties are considered. The researchers have proposed a variety of different approaches for modeling the reliability of traditional software systems [3–11]. However, these approaches are not suitable for modeling reliability in service-oriented systems due to a specific reliability challenges that come out of special services features. In this section specific issues for modeling the reliability in service-oriented systems are discussed.

As stated in previous section, there exist two most common definitions of reliability in literature. For service-oriented systems, the "reliability on demand" definition (ii) suits better to the nature of services since service invocations can be observed as a discrete, reasonably sparse events. According to the adopted reliability definition the following sections discuss specific issues in service-oriented environments as follows. Section 3.3.1 studies additional information that needs to be available to support reliability analysis in service-oriented systems. Section 3.3.2 examines possibilities how additional information necessary for reliability modeling can be obtained. Section 3.3.3 provides detailed overview of service invocation context parameters that significantly impact experienced service reliability. Finally, Section 3.3.4 describes the adopted failure model for reliability modeling in service-oriented systems.

3.3.1 Information to Support Reliability Analysis in SOA

Service-oriented systems are built according to the set of rules and principles defined in *SOA* [47, 48]. In *SOA* environments, each service should publish the information which is necessary to perform successful service invocation. This information, described using a dedicated language such as WSDL [52], usually provides the names of the supported operations, and the names and types of their input and output parameters. In order to support analysis of some non-functional property such as service reliability, each service should publish the reliability related information as well.

It is very important to isolate which information should be published in order to support reliability analysis. One of the basic principles of SOA paradigm is that each services composition can be used as a service for further composition. Hence, there should be distinguished two types of services:

(i) *Atomic service*, an indivisible software artifact that is too granular and executes fewer technical functionalities without requiring any other resources to perform its tasks.

(ii) *Composite service*, built as a composition of other selected services which are required for the composite service to carry out its task. The services are organized and orchestrated using a special dedicated workflow description language such as *BPEL* [53], *CL* [54] or *SSCL* [55].

The reader should note that there is a mapping between services and consumer applications, meaning that the atomic consumer applications can be considered as atomic services, while

composite consumer applications can be viewed as composite services.

In terms of reliability, the difference between atomic and composite services is that the service provider of an atomic service can publish complete reliability information that can be used by clients for reliability modeling. On the other hand, the provider hosting a composite service can only publish the part of the reliability information that is concerning the part of the service implementation which is under the provider's direct control. This part of the reliability information is called *internal reliability* in literature [51]. However, this information needs to be combined with the reliabilities of other selected services to obtain the entire reliability of the composite service.

It is also challenging to combine these reliabilities properly to assess the reliability of the composition. This means that the right weight should be assigned to each service's reliability. Thus, a rarely invoked service has a smaller impact on reliability than the frequently used one. Besides having the information about the reliability of each service within the composition, it is also important to obtain the information about the structure and dynamic execution properties of the composition which is often called *usage profile* in literature [51].

When considering the reliability properties of service-oriented systems, most of the researchers commonly focus on studying the reliability of service compositions. In such manner, plenty of different approaches for prediction of the composite services reliability have been proposed [51, 56–63]. These proposed approaches usually assume that the reliability values of the atomic services are available or scarcely indicate how can they be acquired.

By contrast, this dissertation is focused on atomic services reliability modeling. The aim is to determine the reliability values for atomic services when service users are heterogeneous and varied. The following sections describe the difficulties and challenges that need to be overcome in order to accurately asses the atomic services reliability.

3.3.2 Obtaining Reliability Information

The ideal scenario for obtaining the information about service reliability is the case when the information is provided along with the service description at the time the provider publishes and deploys the service. In a more realistic scenario, the service reliability data can be obtained by monitoring the service.

As already, stated, the primary focus of this dissertation is atomic services reliability modeling. However, when composite services are considered, the structure of the composite workflow

is described using special service composition languages. The description provides the information about possible invocation patterns of services comprised within the composition. Note that sometimes is not possible to determine the exact execution flow due to the ambiguities introduced by presence of some specific control flow statements. For instance, if there is a branch statement in the composition workflow, it is not possible to determine which of the branches will get executed. However, by monitoring the relative frequencies of the different branches, and collecting such data over an adequate number of different invocations, the probability of different patterns can be estimated.

Regarding the atomic services, the only necessary information is the *internal reliability* of the service, which corresponds with the actual reliability of the service. According to the adopted reliability definition (ii), the reliability value can be computed from the past invocations sample as the ratio of the number of successful service invocations against the total number of invocations. The accuracy and relevance of the computed reliability value depends on the quality and quantity of the past invocations sample.

However, gaining a numerous and diverse past invocations sample proves to be a very difficult task in practice. There is a difference in a reliability perception from the user's and service provider's perspective due to the variabilities stemming from the service invocation context [51]. The reliability value computed considering exclusively the data obtained by the service provider could be incorrect for a specific user due to oscillations caused by a variety of parameters that influence the invocation context. For example, users in different geographic locations might experience different reliabilities while using the same service. From the service user's perspective, further obstacles are related with the service usage cost and performance issues. For example, collecting the invocation sample by performing service reliability testing can be extremely expensive for the services that are not free of charge. On the other hand, conducting "stress testing" can significantly impact the performance of the service and also make the measured data irrelevant [13].

One possible approach to address these obstacles is to obtain *partial* but relevant history invocation sample by leveraging human feedback regarding service usage and collecting as much as possible data from the service providers (1), and to utilize *prediction* methods for the estimation of missing reliability records (2).

3.3.3 Parameters of the Service Invocation Context

As already stated in previous section, acquiring a comprehensive past invocation sample appears to be a challenging task. Furthermore, even if one could gain a comprehensive past invocation sample, there are still difficulties when estimating the service reliability due to reliability oscillations produced by the service invocation context parameters.

There are plenty of various factors that might impact and determine service's nonfunctional properties such as reliability, availability, etc. in service-oriented systems. However, it is useful to systematize this matter in order to create an accurate and efficient reliability prediction model. Hence, parameters that define the service invocation context can be grouped in three basic categories: *user–*, *service–* and *environment–*specific parameters. Most of the existing approaches for prediction of atomic services reliability (described in next Chapter 4) implicitly incorporate only user– and service–specific parameters. Note, however, that services exist on the Internet, which is a very dynamic environment by its nature. To propose an accurate service reliability prediction approach, environment–specific parameters also need to be combined.

The user–specific parameters of the service invocation context include any potential factors, introduced by user or caused by some user's attributes, that might produce the variabilities in the perceived service reliability. For instance, some user–specific parameters are user's geographical location at the time of the invocation, quality of the Internet connection that a user is subscribed to or physical device's capabilities a user is using to perform the invocation etc.

The service–specific parameters of the service invocation context are related to any possible cause of variabilities in perceived service reliability introduced by service itself or some service's characteristics. Some of the service–specific parameters that that cause service reliability variabilities are: geographical location of service, computational complexity of the service, quality of the service implementation, system resources dedicated to the service by the provider such as CPU or RAM etc.

The environment–specific parameters are related to the current conditions in the environment comprising users and services that introduce fluctuations in the perceived service reliability. The environment conditions are very important aspect for nonfunctional properties estimation in dynamic environments such as Internet. The environment–specific parameters include environment properties such as network performance or service provider load at the time of the service invocation.

3.3.4 Failure Model for Service-oriented Systems

As stated in Section 3.1, failures can be classified as regular, crash and looping according to their manifestation, and as repairable and non-repairable according to how severe they are.

Crash failures cause the system to stop functioning which means that these failures are the simplest to model from the reliability point of view. Some authors support claims that Internet based systems, including also service-oriented systems, should be design as "crash-only" in order to be more reliable and efficient [64].

Regular failures generate incorrect values at the output of the system. The issue with these type of failures is related to the fact that incorrect output value, produced by a certain service within the service composition, does not need necessary to propagate at the output of the system. In fact, some other service on the path to the output might mask the error. Thus, in order to consider regular failures for reliability modeling, the *error maskability* factor, a capability for a service to map any incorrect input to a correct output, should be included in the model. The reliability modeling can be simplified by assuming that each regular failure in an inner service always propagates to the composition output.

Another important remark regarding the adopted failure model is related to the assumption of independence of atomic services within the composition. There is a possibility that originally independent services are composed in such manner that they rely on some common service and become no longer independent. In this research the impact of common service sharing on reliability is not considered.

3.4 Reliability Prediction System in Consumer Computing

On the basis of analysis provided in previous sections there can be identified few distinct entities that are necessary to support the reliability modeling of consumer applications in Consumer Computing. These entities and relationship between them form a reliability prediction system in Consumer Computing. The following remarks identify and briefly describe each entity that needs to be present for reliability prediction in Consumer Computing.

The discussion in Section 3.2 justifies the sound adoption of service-oriented model for presenting the execution of consumer applications which is a crucial process for consideration of application's nonfunctional properties such as reliability. The conclusions drawn in this discussion confirm that consumer applications can be viewed as a dynamic software artifacts

that provide their functionalities over the Internet such as services defined in SOA. As stated in Section 3.3.2, one possible approach to asses the reliability of services underlying consumer applications is to use *prediction models*. Each prediction model utilizes the actual measured *data*, collected during previously performed tasks, to produce the predictions for the ongoing tasks. The prediction system produces predictions using a prediction *algorithm*. It is obvious that the quality of produced predictions depend both on quality of the prediction algorithm and the quality of the collected data.

Another important aspect of the reliability prediction system is related to the collecting *feedback*. In order to have a comprehensive data sample, the prediction system needs to collect the feedback data about selected components both from consumers and service providers. In fact, the past invocation sample needs to be maintained, it needs to be updated with the most recently experienced reliability data, and also the expired past reliability data needs to be removed from the sample.

In order to help consumers to optimize the reliability of components for their consumer applications, the functionality of the prediction system needs to be exposed and incorporated in a special consumer assistant application. This assistant application, called *Geppeto ReliabilityOptimizeMe*, needs to provide *recommendations* for consumers during the components selection by sorting the same functional components according to their predicted reliability values. For the purpose of collecting consumers feedback on selected components an appropriate consumer intuitive rating system should be proposed.

According to the afore mentioned remarks, the architecture of the reliability prediction system in Consumer Computing is depicted in Figure 3.4. The following sections describe each entity in details. Thus, Section 3.4.1 describes the *Prediction System* and the requirements it needs to accommodate to be successfully applied in Consumer Computing. Section 3.4.2 describes the *Feedback Management System* that collects feedback and maintains the past invocations sample. Section 3.4.3 provides a brief overview of most significant rating systems used in moder recommendation systems on the Internet and analyzes the aspect of their potential adoption and adjustment in Consumer Computing. Finally, Section 3.4.4 presents the consumer assistant application *Geppeto ReliabilityOptimizeMe* that provides consumers recommendations regarding the reliability of potential candidates during the process of selection.

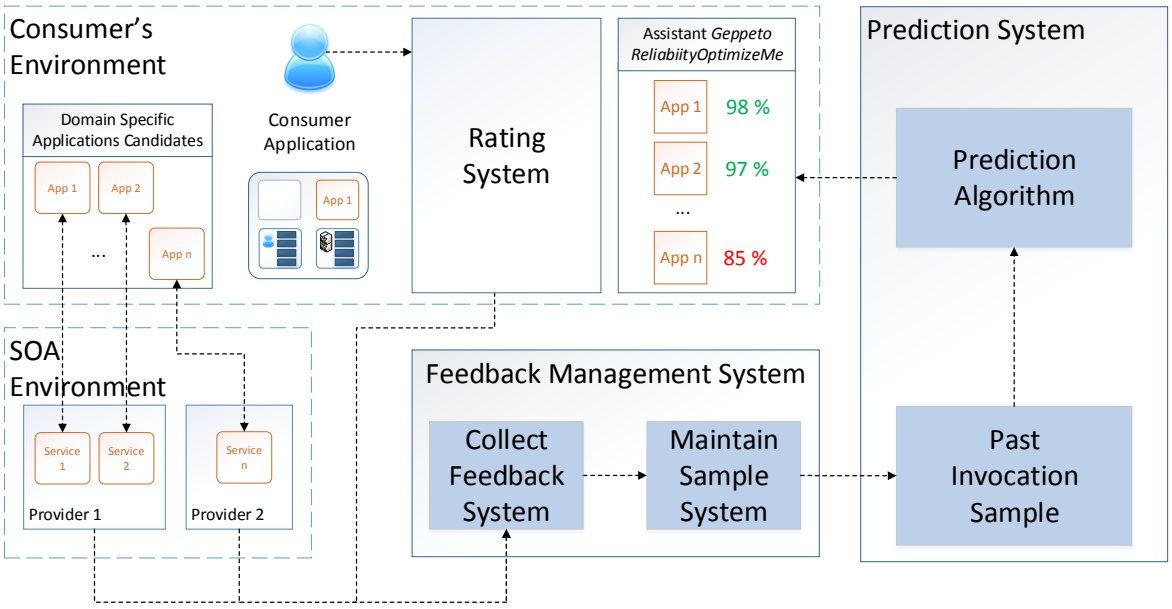


Figure 3.4: Architecture of the reliability prediction system in Consumer Computing

3.4.1 Prediction System

As depicted in Figure 3.4, the *Prediction System* is consisted out of *Prediction Algorithm* and *Past Invocations Sample*. The *Prediction Algorithm* module implements the chosen prediction method and produces predictions for ongoing service invocations using the collected data about previous invocation stored in the *Past Invocations Sample*.

As already described the quality of the prediction is depending on both quality of the algorithm and quality of the past invocations sample. However, past invocations sample is maintained by the *Feedback Management System* which is described in the following section (see Section 3.4.2 for details), while this section is focused on the prediction model qualities.

Real service-oriented systems, which are underneath the consumer applications in Consumer Computing, comprise a substantially large number of users and services. However in such systems, each user visits only a limited subset of services and also new users and services emerge in real time. As a consequence, reliability values are known only for the limited set of service invocation contexts, and there is a number of invocation contexts for which the reliability values are missing and need to be predicted. Besides the *accurate* reliability prediction for the missing invocation contexts, the designed method should support *scalable* and *real time* reliability prediction performance. This is an very important requirement from the Consumer Computing point of view. In fact, from consumer perspective, the algorithm that takes too long to produce the predictions can make consumers less enthusiastic while creating their

applications.

The researchers have proposed several prediction models that leverage the available past invocations records to estimate predictions for the ongoing invocations and they are described in the next chapter (see details in Chapter 4). The proposed state-of-the-art prediction models are based on collaborative filtering technique, often used in recommendation systems. Although the existing collaborative filtering based approaches achieve promising performance, they demonstrate disadvantages primarily related to the prediction accuracy in dynamic environments and scalability issues influenced by the invocations sample size.

When prediction accuracy is concerned, collaborative filtering provides accurate recommendations in static environments where the collected data records are relatively persistent. This means that the records remain up to date for a reasonably long period of time (e.g. *songs ratings*, *band recommendations*). On the other hand, SOA systems are deployed on the Internet, a very dynamic environment in which service providers register significant load variations during the day [65–67]. In such a dynamic environment, user perceived service reliability may considerably change depending of the actual time of invocation. Furthermore, collaborative filtering approaches store reliability values for each user and service pair. Having millions of users and a substantially large number of services, these approaches do not scale.

As part of this dissertation two different prediction algorithms are proposed in order to address the drawbacks of the existing approaches. The first proposed approach, called *LUCS* (service **L**oad, **U**ser location, service **C**lass, **S**ervice location) according to the model parameters, is described in details in Chapter 5. The *LUCS* aims to improve scalability and accuracy of the existing collaborative filtering approaches by: (1) extending the model with the parameters that describe the environment and the internals of a service and grouping the past invocation data into discrete sets across several dimensions according to the model parameters, and (2) performing collaborative filtering based on the set membership considering impact of each model parameter for the ongoing invocation. The final reliability prediction is computed using a linear combination of each model parameter's impact.

The *LUCS* approach prediction accuracy is highly dependent on the explicit availability of the model parameters which are sometimes difficult to gain. To address this drawback, a second approach called *CLUS* (**CLUS**tering) is proposed. The *CLUS* (described in details in Chapter 6) aims to improve the performance of the state-of-the-art approaches by: (1) considering all parameters of the service invocation context: user–, service– and environment–specific param-

eters (the existing approaches implicitly consider only user– and service– specific parameters, while this approach introduces additional parameter that describes the condition of the environment), and (2) reducing the redundant data by grouping users and services into respected user and service clusters according to their reliability performance using K-means clustering algorithm. The authors in [68–70] show that different nonfunctional qualities of a service are influenced by specific service characteristics such as internal complexity or service location. In fact, the existing collaborative filtering based approaches support claims that similar users and services obtain similar reliability values, which can be utilized to aggregate the redundant data and improve scalability.

3.4.2 Feedback Management System

In order to produce accurate predictions, the past invocations data sample needs to be maintained up to date. The role of *Feedback Management System* is to maintain the past invocations sample as depicted in Figure 3.4. Although the challenge of collecting feedback and maintaining the invocations sample is not the primary concern of this dissertation, some basic remarks are presented in the following text.

As presented in Figure 3.4, the *Feedback Management System* is consisted out of two separate modules: *Collect Feedback Module* and *Maintain Feedback Module*. The role of *Collect Feedback Module* is to collect feedback regarding the selected components both from consumers and service providers. The role of *Maintain Feedback System* is to manipulate with the reliability data records in the past invocations sample. The *Collect Feedback Module* passes the collected feedback to the *Maintain Feedback System* which than directly manipulates with the data.

In such manner, new recently experienced reliability values, collected either from consumers or providers, need to be included in the sample. On the other hand, some old, expired reliability records need to be removed from the sample. For instance, the service provider might decide to improve the implementation of the service because of the poor reliability performance. In this case, past reliability records need to be removed from the sample because these records are not relevant for a new service implementation. In addition, some service might become obsolete and service provider might decide to stop offering it. In this case, the reliability records regarding that service need to be removed from the sample because they impact the accuracy of the prediction. Also, in the case a very new service arises in the system, the collected reliability

records regarding a new service need to be added to the sample.

Furthermore, the existing user might change the Internet provider or choose some other service plan that offers better network performance. In this case, the reliability records associated with that users become obsolete and need to be excluded or replaced with the new one. Similar like with the services, if a very new user appears in the system, the past invocation sample needs to be updated with the reliability records associated with a new user. Different strategies can be applied regarding the existing users that were active in the system sometimes in the past. One possible approach is to keep those records, because the user might appear in the system anytime. However, although such an approach seems to be fair to the users, it is questionable from the prediction accuracy and performance point of view. First, it is not efficient to keep the old records because of the size of the invocations sample. The new users arise in real time, keeping the old records posses the scalability issues for the prediction algorithm. Second, the old records obviously impact the prediction accuracy. However, the relevance and validity of those records is questionable. The other possible approach is to dismiss the data from the sample once it expires.

Regarding the expiration time and update frequency of reliability data records, different strategies can be applied to maintain the invocation sample. One possible strategy is to update the data as quick as possible, perhaps on a hourly or a daily basis, depending on the properties of the environment comprising users and services. Such a strategy would assure the invocations sample to be up to date and it would definitely improve the prediction accuracy. On the other hand, this strategy demands significant processing and memory resources and it may disturb the prediction performance which is closely related to the requirements of the prediction in real-time. Another possible strategy would be to update the invocations sample less frequently on a weekly or a monthly basis, again depending on the environment properties. This strategy would not effect the performance of the prediction, but it could compromise the prediction accuracy in the case the environment changes more frequently than the invocation sample is updated.

It is obvious that the optimal strategy is a compromise, somewhere "in the middle" between the two edge cases. The optimal strategy requires the presence of the monitoring system within the *Maintain Sample System*. The role of the monitoring system is to compare new, recently collected data with the old records within the sample. By comparing old and new records, the monitoring system can discover potential changes in the environment and determine the change frequency for the environment. Hence, this strategy with the monitoring system is an effective

trade-off between the accuracy and performance. From the performance point of view, the strategy with the monitoring system requires far less resources than the strategy that employs frequent update of the sample. From the accuracy perspective, the monitoring system ensures quick detection of significant changes in the environment and consequently triggers the eventual update of the sample.

3.4.3 Rating System

The rating model is another very important aspect of each recommendation/prediction system that needs to be considered in order to produce the quality recommendations/predictions. There are two possible solutions to gain ratings regarding the reliability of components in Consumer Computing. The first approach is to collect the data in service oriented environment by analyzing the available service providers logs about the service usage. There has been proposed a variety of approaches for monitoring nonfunctional properties in service oriented systems [71–76], which can be used to retrieve the information about the service usage from providers. However, the data collected exclusively from the provider's side is not comprehensive due to a significant number of reliability violations that appear on the user's side while delivering services. Hence, to gain a comprehensive data sample for the prediction system, the users feedback should be also incorporated and equally combined along with the data obtained form the providers logs as presented in Figure 3.4.

The majority of the researchers focus on improving prediction algorithms when enhancing performance of the recommendation systems. However, some most recent efforts explore the quality of ratings and their impact on prediction performance [77–82]. First, it is crucial to design the appropriate rating model from users point of view. The purpose of rating systems is to collect users feedback regarding items of interests in order to obtain data for future predictions. Hence, it is important that users perceive the rating model as interesting in order to provide as much as possible feedback. On the other hand, from recommender's standpoint, it is important to choose the rating model that is suitable for the particular prediction domain. For instance, different prediction domains might prefer different granularity of ratings to produce the most accurate predictions.

The existing literature provides ratings classifications regarding several aspects [83, 84]. According to the ratings relevance, there should be distinguished two kinds of ratings: *positive* ratings and *negative* ratings. Users provide positive ratings to the items they are interested in,

while negative ratings are usually given to the items that are not of interest. According to the ways the ratings are obtained, the ratings can be classified as *implicit*, *explicit* or *hybrid* ratings.

Implicit ratings are those ratings that are obtained by monitoring users activities and actions rather then directly asking them about their preferences regarding particular items. For instance, if a user views the specifications and characteristics of a particular product, the conclusion that a user is interested in the product can be drawn. There are some additional actions that can be tracked for the web documents, such as enlarging images, scrolling, printing, etc. In addition, the time a user spends while examining a certain product can be measured and associated with the user's interest in a particular product. The benefit of implicit ratings is that they can be easily obtained in lack of explicit ratings. However, the amount of data which collected by employing implicit ratings is very large, and it often needs extensive transformation to replace the explicit ratings.

By contrast, the *explicit* ratings are those that are collected by directly asking users to express their opinion on particular items. There are several kinds of different rating scales used in contemporary recommendations systems. Social network sites such as *Facebook* and *Google+* often use *unary* rating scales in which users can only express their positive attitude regarding particular items (for example, users can "like" feeds they find interesting). Further, some web sites such as *YouTube* and *Digg* use binary rating scales in which users can provide positive ratings to the items they find interesting, but also negative ratings to the items that are not of interest (for instance, users can "like" and "dislike" particular videos on *YouTube*). Additionally, some web sites such as *Amazon* bookstore and *IMDB* movie database site enable users to rate items on a discrete numerical scales (for instance, *IMDB* offers users to rate movies on a 10-star rating scale). Finally, some web sites support explicit ratings by enabling users to post textual comments regarding particular items (*Movielens* site enables ratings of movies via textual comments). The most significant drawback of explicit ratings is related to the fact that the Internet users are reluctant to invest personal time to provide feedback. However, there are some motivators that could stimulate raters such as personal contributions to the community advancement, access to some more advanced website functionality, satisfaction about having their personal opinion voiced and valued, etc.

It is obvious from the arguments presented in previous paragraphs that both implicit and explicit ratings have some advantages and disadvantages. As it often appears to be in many different technologies, the best performance is produced by combining several existing ap-

proaches. In such manner, implicit and explicit ratings could be combined in order to create the best resulting – *hybrid* approach. While implicit ratings reduce users efforts and mental activities, explicit ratings provide accurate inputs to the recommendation system. In such a hybrid approach, explicit ratings can be used to obtain some precise and direct data from users, while implicit ratings can be used to actually confirm the validity of the ratings which are collected explicitly. For instance, if a user explicitly rates a particular product, there should be a implicit evidence that a user has examined the product. In case the evidence is missing, the confidence in user's explicit rating should be reduced.

Regarding the adoption of some particular rating system in Consumer Computing, several approaches are possible. First, implicit ratings could be gained by analyzing data retrieved from the service providers about how different consumers use particular services (underlying consumer applications). However, those implicit ratings consider exclusively the reliability data from the service provider's standpoint which is not sufficient for accurate reliability prediction (see Section 3.3.2 and Section 3.3.3). Hence, in order to collect the quality data sample for the prediction algorithms, implicit ratings need to be combined with the explicit ratings obtained directly by asking consumers to provide feedback regarding components they used. Another open question is related to the rating scale that should be used while acquiring explicit ratings. At first glance, it seems that the best possible choice is a binary scale, in which consumers can provide positive ratings for components that operate successfully or negative ratings for components that fail. The reason why a binary scale seems to be promising in this particular domain is related to the fact that there are two possible outcomes of the service (underlying consumer application) invocation: either it gets executed successfully or it fails. Note, however, that rating scales are not the primary concern of this dissertation. In order to make conclusions, a more deeper research should be conducted, in particular; how different rating scales impact the prediction accuracy and performance.

3.4.4 Consumer Assistant *Geppeto ReliabilityOptimizeMe*

This section describes consumer assistant application *Geppeto ReliabilityOptimizeMe* which is mandatory for reliability management in Consumer Computing. *Geppeto ReliabilityOptimizeMe* provides consumers assistance regarding potential candidate's reliability while selecting components.

As already presented in Figure 3.4, consumers create their applications by composing the

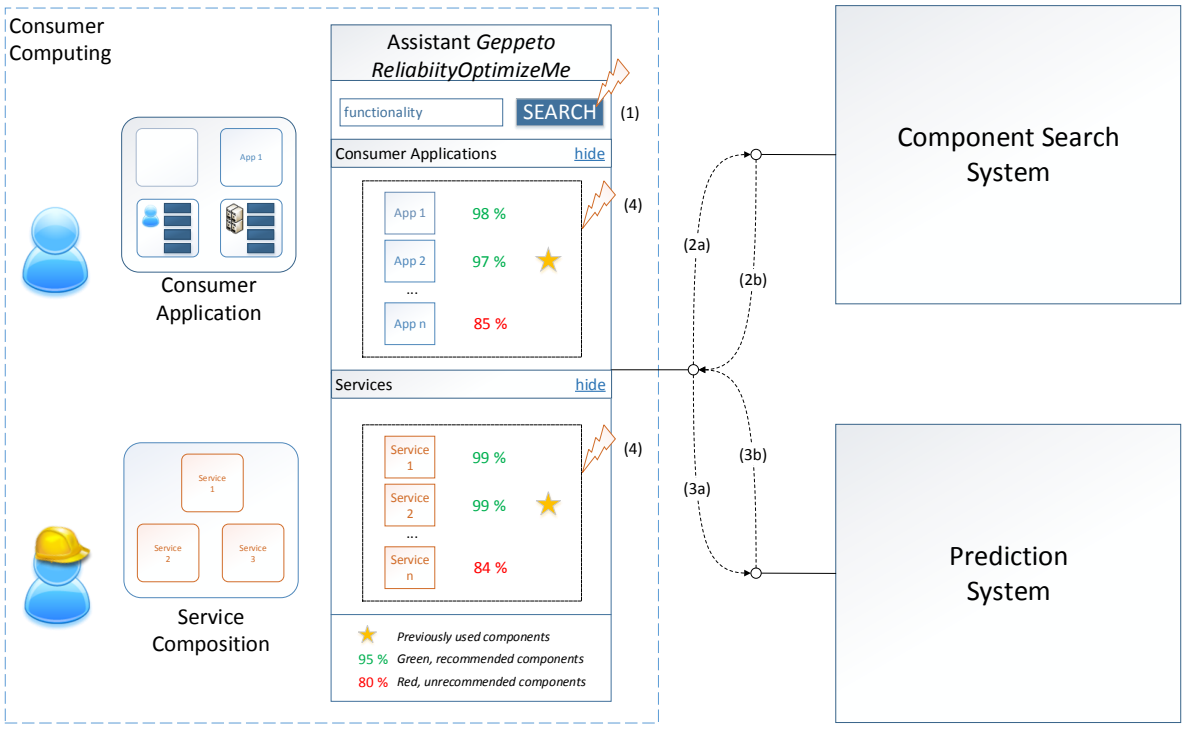


Figure 3.5: *Geppeto ReliabilityOptimizeMe* assistant and a use case scenario

existing applications as basic components. However, while selecting components, besides logical functioning, consumer should also consider nonfunctional properties such as reliability of potential selection candidates. The role of *Geppeto ReliabilityOptimizeMe* consumer assistant is to provide reliability ratings for functional components which are considered for selection by consumer. For the purpose of this dissertation, the description is focused on assistant's user interface which should be intuitive and contain all necessary information for consumers to select the adequate candidate. The *Geppeto ReliabilityOptimizeMe* consumer assistant and its use case scenario as well are depicted in Figure 3.5.

As presented in Figure 3.5 consumers have the ability to search for a specific functional component by textually describing the functionality of interest. Once the consumer describes the desired functionality (1), the special part of the system (called *Component Search System* in Figure 3.5) retrieves the list of components that match the desired functionality ($2a - 2b$).

Note that this part of the system, which is responsible for searching and retrieving functional components, is not considered in this dissertation. As part of the *Consumer Computing Lab*, separate research is conducted to perform efficient searching and fetching of components that match the desired functionality. The first approach analyzes the morphological structure of the composite consumer application created so far, and recommends the next functional component

[30]. The other approach inspects which components have been added to the composition so far and finds most prominent consumers who have already used those components in the past and who could be helpful during the process of application creation [31].

However, once the list of functional components is retrieved, the *Geppeto ReliabilityOptimizeMe* assistant sends the request containing list of components to the *Prediction System* to asses the reliability values for each component within the list ($3a-3b$). Once the prediction system produces predictions, the consumer assistant can display the components sorted according to the predicted reliability values (4).

The reader should note that the prediction results are displayed in a consumer intuitive manner. The reliability predictions of recommended components are presented in green color which is often used to present something that is allowed, especially in western civilization (e.g., *traffic lights*, etc.). On the other hand, the reliability predictions of unrecommended components are marked in red color which is often used to present restrictions in western civilization (e.g., *restricted area*, etc.). Also, previously used components, which were rated positively by consumer, are marked with a special *star sign* which is often used to symbolize favorite choices on the Internet.

Another important note regarding the consumer assistant *Geppeto ReliabilityOptimizeMe* is that it can be utilized by engineers and professional developers as well as presented in Figure 3.5. In particular, while creating different composite tasks in service oriented environment, engineers can also use the assistant to select the most eligible service candidates from the reliability point of view.

Chapter 4

State-of-the-art Models for Prediction of Application's Reliability

In the previous chapter (see Chapter 3), Consumer Computing environment and nature of consumer applications were described. Also, the reasons and specific challenges which indicate that traditional approaches for software reliability modeling are inappropriate for predicting the reliability of consumer applications were described. This chapter provides a detailed overview of most relevant existing *state-of-the-art* approaches for predicting the reliability of web applications.

As already described in previous chapters (see details in Chapter 3 and Chapter 2), Consumer Computing environment is a very dynamic and heterogeneous ecosystem. Modeling reliability of applications in the Consumer Computing ecosystem is very challenging due to a variety of parameters that have influence on nonfunctional properties. First, the ecosystem contains consumers located in different locations worldwide while building various domain-specific applications. More specifically, consumers use various physical devices, obtain different network capabilities and behave quite uniquely according to their personal usage profiles. Second, consumer applications are accessible through a simple consumer graphic interface (web interface, mobile interface, desktop interface, tablet interface etc.), but often require information retrieval and data processing over the Internet. Hence, each consumer application can be viewed as a component that provides its functionalities through a publicly accessible interface on the Internet. However, behind the simple interface, there is an underlying dynamic software which implements the desired functionality (see details in Figure 3.4). Finally, service invocation context parameters that determine consumer application's nonfunctional properties are

quite unstable and time dependent in a dynamic environment such as Internet.

All afore mentioned arguments bear out the assertions that consumer applications reliability can be seen as the reliability of dynamic software artifacts such as services described in *Service Oriented Architecture* (*SOA*) [47, 48].

The rest of the chapter is organized as follows. First, classical techniques commonly used in statistics are presented. Section 4.1 describes the *UMEAN* approach for predicting the reliability of services underlying consumer applications, while Section 4.2 provides the description of the *IMEAN* approach for predicting the reliability of consumer applications.

In addition, most successful approaches for predicting the reliability of atomic services [14–17] based on collaborative filtering [18] technique are presented.

There exist three types of collaborative filtering in literature [18]:

- *memory-based*,
- *model-based* and
- *hybrid*.

Since model-based and hybrid collaborative filtering are more complex and costly for implementation [18], the thesis is focused on memory-based collaborative filtering underlying the state-of-the-art recommendation systems [19–23].

Further sections of this chapter overview different types of collaborative filtering by presenting most relevant and distinguished approaches of each type. Note, however, that hybrid and model-based approaches were not considered in the evaluation process (see Chapter 7) due to their greater computational complexity which makes them less scalable. Section 4.3 provides an overview of memory-based collaborative filtering approaches. Section 4.4 presents most relevant model-based collaborative filtering approaches. Section 4.5 reviews most common hybrid collaborative filtering techniques. Finally, Section 4.6 provides a brief overview of all collaborative filtering types presenting each type's major characteristics and challenges.

4.1 The UMEAN Approach

The *user-mean* (*UMEAN*) approach, commonly used in statistics, estimates the result of the current web service invocation considering the collected past invocation data about the user performing the invocation. This approach calculates the average success rate for the associated

user while using different services in the past. In [85], the authors define user-perceived service availability and they calculate it using *UMEAN* approach.

Similarly, the reliability of the service $\overline{p_u}$ can be calculated using *UMEAN* approach as follows:

$$\overline{p_u} = \frac{\sum_{a \in S(u)} p_{u,a}}{|S(u)|}, \tag{4.1}$$

where $S(u)$ represents the set of all services used by user u in the past, $p_{u,a}$ is the success rate achieved by user u while using service a while $|S(u)|$ is the cardinal number of the set $S(u)$ i.e. the number of services used by user u.

The advantage of this approach is that it can be applied in environments in which services gain similar performance while users by them self introduce the oscillations in perceived service reliability. The reliability oscillations may be caused by various reasons such as specific usage profile or the specific set of services which are used. In addition, nonfunctional properties may depend on a physical device or network infrastructure which is used while accessing the services. For instance, while accessing service through a *dial-up* network connection, similar reliability is obtained regardless of the invoked services.

The main disadvantage of this approach is that it does not consider different impacts of services regarding the operation they execute, the infrastructure they use or the processing power they posses. For instance, the service internal complexity caused by the implementation or required functionality may impact the perceived quality of nonfunctional properties while accessing the service.

4.2 The IMEAN Approach

The *item-mean* (*IMEAN*) approach, which is also commonly used in statistics, estimates the outcome of the current service invocation considering the collected reliability data about previous service invocations. This approach calculates the average success rate for the associated service using past invocations performed by different users. In [12], the authors define service reliability as *successful execution rate* and they use the *IMEAN* statistical approach to calculate the reliability value.

According to the *IMEAN* approach, the reliability of the service $\overline{p_i}$ is calculated as follows:

$$\overline{p_i} = \frac{\sum_{a \in S(i)} p_{a,i}}{|S(i)|}, \tag{4.2}$$

where $S(i)$ represents the set of users that have used the associated service i in the past, $p_{a,i}$ is the success rate obtained by user a while accessing service i and $|S(i)|$ is the cardinal number of the set $S(i)$ i.e. the number of users that have accessed the service i.

The advantage of this approach is that is applicable in environments where user specific parameters obtain similar values (environments with broadband Internet speed and users with physical devices that posses sufficient network and other infrastructure to accept and process different service responses), but services by themselves cause the variability in perceived service reliability. Note that the variability may be related to the processing power or some other physical resources of the provider hosting the associated service. Also, the variability can be related to some service specific heavy computation that is required for the service to execute its task.

The main disadvantage of this approach is that it does not consider user's impact on non-functional properties. In general, all users neither have the same network infrastructure, nor they use the same physical devices. The user specific variability is not considered at all in this approach. However, it may considerably impact perceived quality of service's nonfunctional properties such as reliability.

4.3 Memory-Based Collaborative Filtering Approaches

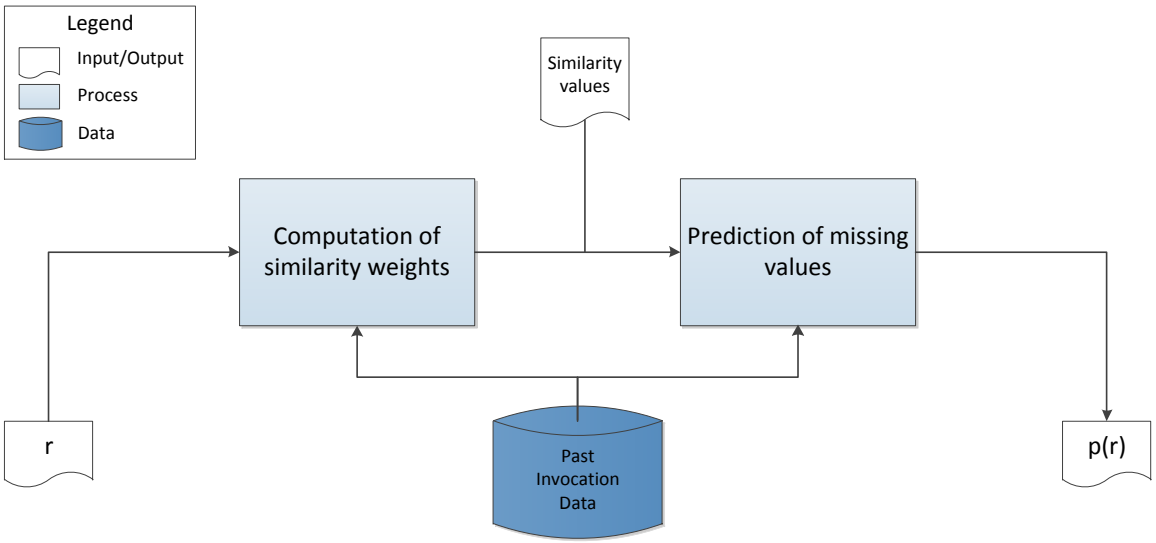


Figure 4.1: Two basic phases in memory-based collaborative filtering.

The memory-based collaborative filtering extracts information or patterns using statistical

collaboration among multiple entities such as agents, viewpoints, and data sources. The main characteristic of memory-based *CF* algorithms is usage of the entire user-item database to generate predictions. Special statistical methods and techniques are used in order to group entities (usually users and items) into groups of people with similar interests. The benefit of this approach is that it can be applied in situations in which specific data that is lacking can be predicted using the available data from the most statistically similar entities. For instance, the preferences of an active user on some particular item can be predicted by considering preferences of active user's so-called neighbors on that particular item.

The *neighborhood-based CF* algorithms, the most common memory-based *CF* approaches, operate according to the following phases as depicted in Figure 4.1:

- calculate the *similarity* or *weight*, $w_{i,j}$, representing weight, distance, correlation or similarity, between two users or items, i and j;
- produce the prediction for an active user by considering the weighted average of all the ratings of the user or item on a particular item or user, or using a simple weighted average [15].

For the case when the task is to produce a top-*N* recommendation, the algorithm should find *k* most similar entities (users or items) after computing the similarities, and then suggest top-*N* most suitable items as the recommendation.

4.3.1 Similarity Computation

Similarity computation between two entities is the critical step in memory-based *CF* algorithms. In order to compute the similarity between two items i and j, the *item-based CF* approaches first find all users who have rated both of these items, and then apply similarity a relation to determine similarity, $w_{i,j}$, between two co-rated items. On the other hand, the *user-based CF* approaches calculate the similarity, $w_{u,v}$,between two users u and v who have both rated the same items.

There are numerous different methods to compute similarity or weight between users or items.

Correlation-Based Similarity

In this case, the *Pearson correlation* similarity relation is used to compute the similarity or weight between different entities ($w_{u,v}$ between two users u and v and $w_{i,j}$, between two items i and j). *Pearson correlation* measures the extent to which two variables linearly relate with each other [86]. The appliance of the *Pearson correlation* relation provides *Pearson correlation coefficient* representing the similarity measure between two entities. For the *user-based PCC* (*UPCC*) approach, the similarity between users u and v is computed as:

$$w_{u,v} = \frac{\sum_{i \in I} (r_{u,i} - \overline{r_u}) \times (r_{v,i} - \overline{r_v})}{\sqrt{\sum_{i \in I} (r_{u,i} - \overline{r_u})^2} \sqrt{\sum_{i \in I} (r_{v,i} - \overline{r_v})^2}}, \tag{4.3}$$

where I presents the set of items that both users u and v have rated, $r_{u,i}$ and $r_{v,i}$ are users u and v ratings on item i respectively, while $\overline{r_u}$ and $\overline{r_v}$ represent the average rating for users u and v computed considering all items respected users have rated.

For the *item-based PCC* (*IPCC*), the similarity between two items i and j is computed as:

$$w_{i,j} = \frac{\sum_{u \in U} (r_{u,i} - \overline{r_i}) \times (r_{u,j} - \overline{r_j})}{\sqrt{\sum_{u \in U} (r_{u,i} - \overline{r_i})^2} \sqrt{\sum_{u \in U} (r_{u,i} - \overline{r_j})^2}}, \tag{4.4}$$

where U presents the set of users that have rated items i and j, $r_{u,i}$ and $r_{u,j}$ are user's u ratings on items i and j, while $\overline{r_i}$ and $\overline{r_j}$ are the average ratings for items i and j respectively.

There can be found some variations of user-based and item-based *Pearson correlations* in the literature [87]. Other correlation-based similarities are *constrained Pearson correlation*, a variation that uses midpoint instead of mean rate; *Spearman rank correlation*, similar to basic *Pearson correlation*, except that ratings are ranks; and *Kendall's τ correlation*, similar to the *Spearman rank correlation*, but instead ranks themselves, only the relative ranks are used to make the prediction [88, 89].

Note, however, that the basic *Pearson correlation*-based *CF* algorithm is a very popular and representative memory-based *CF* algorithm which is commonly used in the research community. The number of users and items in the computation process is usually called neighborhood size for the active user or item, and the similarity based *CF* is regarded as *neighborhood-based*

CF.

Vector Cosine-based Similarity

To calculate similarity between two documents text mining techniques often present each document as a vector of word frequencies, and then compute the cosine of the angle formed by the frequency vectors [90]. This technique can be applied for collaborative filtering tasks by presenting users or items instead documents and ratings instead of word frequencies.

The matrix R, shown in Figure 4.2, is the *user-item* $n \times m$ matrix which stores ratings of n users on m items. The similarity between to items i and j is defined as the cosine of the *n*-dimensional vectors corresponding to the i^{th} and j^{th} matrix columns which store users ratings on those items.

Vector cosine similarity between items i and j is defined as:

$$w_{i,j} = cos(\vec{i}, \vec{j}) = \frac{\vec{i} \bullet \vec{j}}{||\vec{i}|| \times ||\vec{j}||}, \tag{4.5}$$

where "$\bullet$" presents the vector-product of two vectors. For instance, for two 2-dimensional vectors $\vec{A} = x_1, y_2$ and $\vec{A} = x_2, y_2$, the vector cosine similarity between A and B is computed as:

$$w_{A,B} = cos(\vec{A}, \vec{B}) = \frac{\vec{A} \bullet \vec{B}}{||\vec{A}|| \times ||\vec{B}||} = \frac{x_1 \times x_2 + y_1 \times y_2}{\sqrt{{x_1}^2 + {x_2}^2}\sqrt{{y_1}^2 + {y_2}^2}} \tag{4.6}$$

The disadvantage of the *vector cosine similarity* is that it does not consider that different users may have different rating scales. To address this shortcoming, *adjusted cosine similarity* measure is used. The *adjusted cosine similarity* subtracts each user's rating by the average of the respected user over each rated pair. Note that in that case *adjusted cosine similarity* has the same formula as *Pearson correlation*.

Other Similarities

In [91, 92], the authors use *conditional probability-based* measure of similarity. This similarity measure is based on the conditional probability of rating one of the items given that the other item has been already rated. However, this similarity measure is not commonly and widely used in recommendation systems.

4.3.2 Prediction and Recommendation Computation

The prediction or recommendation computation is the most important phase in the memory-based *CF* system. In the case of a neighborhood-based *CF*, a set of most statistically similar neighbor entities is chosen, and a prediction or recommendation for the active user is produced based on the ratings of most similar entities using their weighted average [93].

Weighted Sum of Other's Ratings

In order to make the prediction for the active user a, on a particular item i, collaborative filtering can be applied on the data stored in the matrix r shown in Figure (4.2) in two different ways.

The first approach, the *user-based* approach (*UPCC*) [14], employs the average rating of the active user over all rated items, and the ratings of all other users in according to the extent of their similarity with the active user. Using the *UPCC* approach, the prediction is computed as follows [86]:

$$p_{a,i} = p_{uPCC} = \overline{p_a} + \sum_{u \in U} \omega_{a,u} \times (p_{u,i} - \overline{p_u}), \tag{4.7}$$

where U is the set of most similar users for the active user a, $p_{u,i}$ is the rating of the user u on the item i, $\overline{p_a}$ and $\overline{p_u}$ are the average ratings of the active user, a, and the user u, computed over all rated items, while $\omega_{a,u}$ is the weight factor that enables more similar users to contribute more to the prediction computation. It is calculated as follows:

$$\omega_{u,a} = \frac{w_{u,a}}{\sum_{b \in U} w_{b,a}}. \tag{4.8}$$

where $w_{u,a}$ and $w_{b,a}$ are the similarity measures between the active and the associated user.

The second approach, called *item-based* approach (*IPCC*) [15], employs the average rating on the active item which is computed considering each user's rating on that item, and the ratings of the active user on all other rated items according to the extent of similarity of those items with the active item. The *IPCC* approach produces prediction according to the following formula:

$$p_{a,i} = p_{iPCC} = \overline{p_i} + \sum_{j \in I} \omega_{j,i} \times (p_{a,j} - \overline{p_j}), \tag{4.9}$$

where I is the set of most similar items for the active item i, $p_{a,j}$ is the rating of user a on item j,

$\overline{p_i}$ and $\overline{p_j}$ are the average ratings on items i and j respectively, computed considering each user's rating on those items, and $\omega_{j,i}$ is the weight factor that enables more similar items to contribute more to the prediction computation. It is calculated as follows:

$$\omega_{j,i} = \frac{w_{j,i}}{\sum_{k \in I} w_{j,k}}. \tag{4.10}$$

where $w_{j,i}$ and $w_{j,k}$ are the similarity measures between the active and the associated item.

The main drawback of these approaches is that they exclusively employ either *UPCC* or *IPCC*. I this manner, some of the valuable information gets neglected either way. To address this drawback and improve the prediction accuracy, the researchers have proposed the *Hybrid* approach to make predictions [16, 17]. The *Hybrid* approach considers both the impact of similar users and similar items, and predicts the missing *user-item* records by utilizing a linear combination of *UPCC* and *IPCC* approaches. In this approach, the prediction is produced using the following formula:

$$p_{a,i} = \lambda \times p_{uPCC} + (1 - \lambda) \times p_{iPCC}, \tag{4.11}$$

where λ is a model parameter, having $0 \leq \lambda \leq 1$, and p_{uPCC} and p_{iPCC} are the values predicted by user- and item-based approach respectively. They conducted the series of experiments and their evaluation results demonstrate that better prediction accuracy is obtained when the *Hybrid* approach is used.

Simple Weighted Average

Alternatively, instead of using the *weighted sum of other's ratings*, the *simple weighted average* can be used to calculate the prediction.

For the *user-based* approach prediction using the *simple weighted average*, the prediction for user u on item i, $p_{u,i}$, is computed as:

$$p_{u,i} = \frac{\sum_{binU} p_{b,i} \times w_{b,u}}{\sum_{binU} |w_{b,u}|}, \tag{4.12}$$

where the summations are over all users $b \in U$ that have rated item i, $w_{b,u}$ is the similarity between users b and u, and $p_{b,i}$ is the rating of user b on item i.

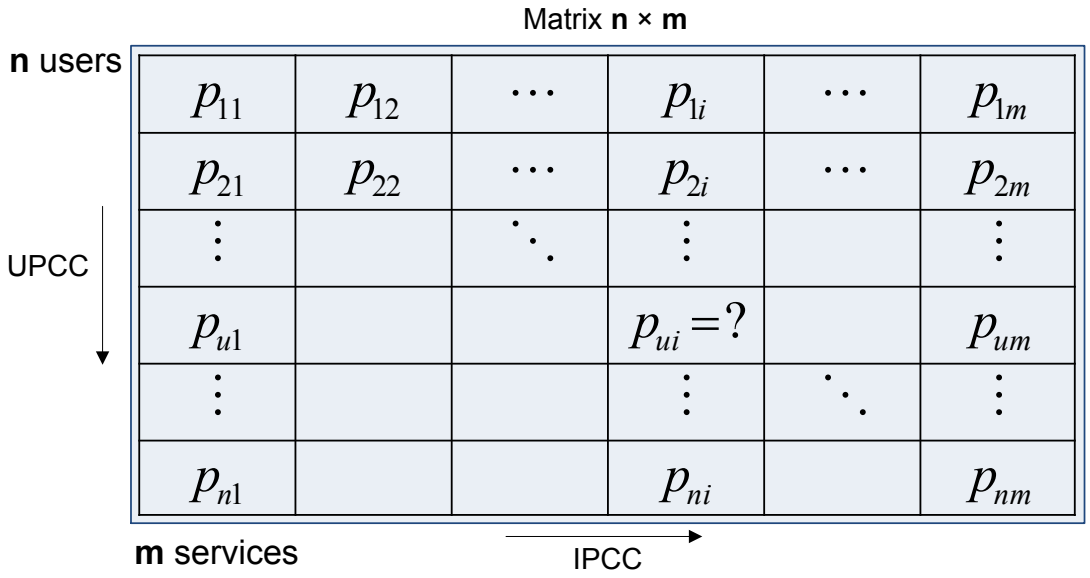


Figure 4.2: The user-item matrix used for collaborative filtering.

Similarly, for the *item-based* approach, the prediction using *simple weighted average* is obtained as follows:

$$p_{u,i} = \frac{\sum_{kinI} p_{u,k} \times w_{k,i}}{\sum_{kinI} |w_{k,i}|}, \tag{4.13}$$

where the summations are over all items $k \in I$ rated by user u, $w_{k,i}$ is the similarity between items k and i, and $p_{u,k}$ is the rating of user u on item k.

4.3.3 Top-N Recommendation

Top-*N* recommendation aims to produce a set of N top-ranked items that will be of interest to a particular user. For instance, each time a user logs in the social network site such as *Facebook*, *Google+* etc., the user gets recommendations to connect with new people, or each time user logs into a *Ebay* account, the list of products that might be of interest gets recommended. Top-*N* recommendation techniques analyze the user-item matrix data in order to discover the relations between different users or items, and then utilize them to produce the recommendations.

User-based Top-N Recommendation Algorithms

User-based Top-*N* recommendations algorithms find the k most statistically similar users (closest neighbors) to the active user using either *Pearson correlation* or *vector cosine* [14, 94]. Each user is regarded as a vector in the m-dimensional item space and similarities between users are computed by calculating distance between representing vectors. Once the set of most similar users K is identified, the rows of matrix R, which correspond to the selected set of users K, are aggregated to discover the set of items C, which are purchased by the users within the set K with their respected purchase frequencies. Finally, the N most frequent items which have not been purchased by the active user yet, are selected as the recommendation system output. The serious disadvantage of user-based Top-N recommendation algorithms is that they suffer from potential scalability issues and real-time performance [91].

Item-based Top-N Recommendation Algorithms

In order to address scalability issues of Top-N *user-based* recommendation algorithms, the Top-N *item-based* recommendation algorithms were proposed. These algorithms first discover k most similar items for each item according to some similarity measure, and then identify the set C, which is the union of each item's k most similar items. At the same time, the items that the active user has already purchased are excluded into a special set U. In the next step, the similarity between each item of the set C and the set U is calculated. Finally, the Top-N most similar items from the C will be recommended as the results list [91]. The drawback of this approach is that in the case the joint distribution of a set of items differs from the distributions of each single item the approach provides suboptimal recommendation. To tackle this drawback, the authors [92] propose higher-order item-based Top-N recommendation algorithms that use all combinations of items up to a particular size when recommending items to the active user.

4.3.4 Extensions to Memory-Based Collaborative Filtering Algorithms

Default Rating

Many collaborative filtering approaches [86, 94] base their predictions on the intersection of items that both users have rated. However, this approach will not be confident in the case there are to few available values to compute the similarities. Also, focusing exclusively on the intersection of items ignores the global rating behavior of the active user which is registered

in the user's rating history. In such situations, the researchers have empirically shown that prediction accuracy can be improved if some *default rating* values for the missing ratings are assumed. In [93], the authors reduce the weight of users that have small intersections containing fewer than N items in common. Chee et al. [95] use the average of the group as a default rating to extend each user's rating history.

Inverse User Frequency

The *inverse user frequency* idea [90] aims to emphasize the impact of less common rated items rather than the impact of universally rated items since the less common rated items are more useful for determining similarities. The inverse frequency can be defined as:

$$f_i = \log(\frac{n}{n_i}), \tag{4.14}$$

where n_i is the number of users that have rated the item i and n is the total number of users. If everyone have rated some item i than f_i is zero. To apply the inverse frequency in collaborative filtering, for the similarities computation phase, the item with higher f_i should be given more weight in the similarity calculation, and for the prediction computation phase, the transformed ratings should be used which are simply the original ratings multiplied by the f_i factor [14].

Case Amplification

The case amplification idea is about to transform the weights used in the basic collaborative filtering in a way that higher similarities are emphasized and lower similarities are disfavored. The transformed weights are calculated as follows:

$$w'_{i,j} = w_{i,j} \times |w_{i,j}|^{\rho-1}, \tag{4.15}$$

where ρ is the *case amplification* power, $\rho \geq 1$, and a typical choice of ρ is 2.5 [96]. Case amplification is use to reduce the noise in the data, it tends to favor high similarities while low similarities values raised to a power become negligible.

Imputation-based Collaborative Filtering Algorithms

The experiments have shown that *Pearson correlation*-based *CF* algorithms have problems with producing accurate predictions for the extremely sparse data. To address this drawback,

Su et al. [97, 98] proposed *imputation-based CF* algorithms (*IBCF*). The *IBCF* approach first uses the imputation technique to fill in the missing data, and then uses a traditional *Pearson correlation*-based *CF* algorithm on this completed data to make predictions for a specific rating. The authors investigated a variety of standard imputation techniques such as mean imputation, linear regression imputation, predictive mean matching imputation [99], Bayesian multiple imputation [100], and as well machine learning classifiers [98] such as naive Bayes, *SVM*, neural network, decision tree, lazy Bayesian rules to complete the data in *IBCF* and found that proposed *IBCF* can perform effectively. More specifically, they found that *IBCF* using Bayesian multiple imputation, *IBCF-NBM* (a mixture *IBCF* that uses naive Bayes for denser datasets and mean imputation for sparser ones) [97], and *IBCF* using naive Bayes outperform the *content-boosted CF* algorithm which is a representative *hybrid CF* approach.

Weighted Majority Prediction

In [101], the authors proposed *weighted majority prediction* that makes prediction using the rows with measured data in the same column, weighted by the similarity between the rows with binary ratings. The similarity weights are initialized to the value of 1, and further computed using the following equation:

$$w_{i,\acute{i}} = (2-\gamma)^{C_{i,\acute{i}}}\gamma^{W_{i,\acute{i}}}, \tag{4.16}$$

where $0 < \gamma < 1$, $C_{i,\acute{i}}$ is the number of rows that have the same value as in row i, while $W_{i,\acute{i}}$ is the number of rows that have different values then the i^{th} row. The prediction on a certain item is produced using the value on that item in the row that has highest accumulated similarity weight with the active row. The algorithm can be extended to multi-class data [102]. The disadvantage of this approach is scalability, once the number of users or items, n, gets substantially large, it becomes impractical to update the values in the matrix which requires the computational complexity of $O(n^2)$.

4.4 Model-Based Collaborative Filtering Approaches

In order to improve the prediction accuracy of simple memory-based collaborative filtering techniques and provide more intelligent predictions, the researchers have proposed the design of models. The general idea is to use more advanced techniques such as machine learning and data mining algorithms to learn the prediction model to recognize complex patterns based on

training data, and then use the model to make predictions on the real-world data. The primary motivation for model-based *CF* algorithms is to address well known disadvantages and challenges in memory-based *CF* algorithms [14, 103]. Most commonly used models are Bayesian models, clustering models and dependency networks. Classification algorithms can be applied in the case user-item ratings are discrete and categorical, while regression models are usually used for numerical user-item ratings.

4.4.1 Bayesian Belief Net Collaborative Filtering Algorithms

A Bayesian belief net (*BN*) is a directed acyclic graph (*DAG*) define with a triplet (N, A, Θ). Each graph node $n \in N$ is a random variable, each directed arc $a \in A$ represents probabilistic relation between variables, and Θ is a conditional probability table specifying how each node depends on its parents [104]. Bayesian belief nets (*BNs*) are commonly used for classification tasks.

Simple Bayesian Collaborative Filtering Algorithm

In simple Bayesian *CF* algorithm a naive Bayes (*NB*) strategy is used to predict missing values for *CF* tasks. The probability that missing values belong to a specified class is calculated for each available class assuming the features are independent given the class. The class with the highest probability is chosen as the predicted class [105]. The incomplete data class value $class$ is computed using each complete data d from the set of collected complete data D:

$$class = \underset{j \in classSet}{\arg\max}\, p(class_j) \prod_{d \in D} P(X_d = x_d | class_j). \tag{4.17}$$

In order to calculate the probability and avoid conditional probability of 0, the *Laplace Estimator* is used:

$$P(X_i = x_i | Y = y) = \frac{\#(X_i = x_i | Y = y) + 1}{\#(Y = y) + |X_i|}, \tag{4.18}$$

where $|X_i|$ is the cardinal number of class set.

For example, if the binary class is considered, $P(X_i = 0|Y = 1)$ will be $(0 + 1)/(1 + 2) = 1/4$, and $P(X_i = 1|Y = 1)$ will be $(2 + 1)/(1 + 2) = 3/4$ using *Laplace Estimator*.

The majority of real-world *CF* data sets are multi class ones. In order to apply *NB CF* algorithm on the multi class data easier, the authors in [106] transform multi class data into binary class data and then perform the algorithm. However, the data conversion introduces

issues of scalability and information loss for multi class data.

In [107], the authors perform naive Bayes strategy directly on the multi class data and show that *NB* algorithm has worse prediction accuracy but better computational performance than *Pearson correlation* memory-based *CF* because the prediction process is less time consuming.

NB-ELR and TAN-ELR Collaborative Filtering Algorithms

Extended logistic regression (*ELR*) is a "discriminative learning" algorithm that maximizes *log conditional likelihood* (*CL*) rather than "generative learners" that maximize *log likelihood*. Generative learning algorithm would create a perfect model of the distribution that will perform optimally for any possible input. However, *CF* tasks include limited training data, which is not suitable for generative learners. Hence, the *ELR* algorithm aims to find the parameters that will perform well according to the collected data. The *ELR* uses logistic regression to seek parameters which maximize *CL* [108, 109].

Both, *TAN-ELR*, which is tree augmented naive Bayes [110] and *NB-ELR*, which is a naive Bayes optimized by *ELR*, have shown to perform significantly better than *NB CF* algorithms and consistently better than the *Pearson correlation* memory-based *CF* from the aspect of prediction accuracy. However, these algorithms require a longer time to train the modes. The solution is to run the time-consuming models training phase off-line, and then to deploy the optimal models to produce the real-time predictions.

Other Bayesian Belief Nets Collaborative Filtering Algorithms

Bayesian belief nets with decision trees at each node model has a decision tree at each node of the *BNs*, each node represents the item object and the states of each node represent each item's ratings [14]. The evaluation of the model shows that it achieves similar performance as the *Pearson correlation* memory-based *CF* regarding the prediction accuracy and better scalability than the vector cosine memory-based *CF*.

Baseline Bayesian model uses Bayesian belief nets with no arcs and makes predictions according to the overall item popularity [111]. However, the performance of the model is poor.

4.4.2 Clustering-Based Collaborative Filtering Algorithms

A data cluster contains data objects that are similar to each other within the same cluster and dissimilar to the data objects in other clusters [112]. There exist different measurements of

similarity between data objects defined using different metrics and similarity relations such as *Minkowski* distance and *Pearson correlation*.

The popular *Minkowski* distance for two data objects, $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$ is defined as follows:

$$d(X,Y) = \sqrt[q]{\sum_{i=1}^{n} |x_i - y_i|^q}, \tag{4.19}$$

where n is the dimension of the compared data objects, x_i, y_i are the values of the i^{th} dimension of data objects X and Y respectively, and q is a parameter of the distance relation that may be assigned with a positive integer value. For the case $q = 1$, $d(X,Y)$ is the *Manhattan* distance and for the case $q = 2$, $d(X,Y)$ is the *Euclidean* distance.

There exist three categories of clustering methods in literature [112,113]: partitioning methods, density-based methods, and hierarchical methods. The most popular partitioning method is *k*-means clustering [26], which is very easy to implement and very effective. Most common density-based clustering methods are *DBSCAN* [114] and *OPTICS* [115]. The density-based methods search for dense clusters of data objects isolated by sparse regions that represent boundaries among clusters. Hierarchical clustering methods split the data into clusters set by applying hierarchical decomposition using some given criterion. A typical representative of hierarchical clustering is *BIRCH* [116].

Usually, the clustering methods are performed as an intermediate step prior to further data processing such as prediction, classification or some other task. The clustering methods can be applied in various ways for the *CF* tasks. In [117, 118], the researchers use clustering methods to split the data set into smaller distinct clusters, and then they perform *Pearson correlation* memory-based *CF* algorithm to predict the missing values on the reduced data within clusters.

The *RecTree* method [95] recursively splits a large data set into two sub-clusters using *k*-means, $k = 2$, and it forms an unbalanced binary tree with leaf nodes representing data clusters. Each leaf node contains the data that belongs to its cluster and external nodes contain centroid information about their subtrees. The binary tree structure containing centroid information in external nodes is used to guide active user-item value prediction to the leaf node presenting the cluster that it belongs to, and then the prediction is made within the given cluster. The computational complexity required to form the binary tree is $O(n \log_2(n))$ and the complexity required to make on-line predictions is $O(b)$, where n is the data set size and b is a constant representing partition size. Their evaluation results show that *RecTree* method improves the

accuracy of *Pearson correlation* memory-based *CF* algorithm when an appropriate partition size is chosen.

In [119], the authors separately cluster users and items into respected users and items clusters. The users are clustered based on the items they rated and the items are clustered based on users that rated them. The prediction accuracy of the approach is good on the synthetic data, but it does not look promising on the real data.

Another approach, flexible mixture model (*FMM*) [120], uses the extension of existing clustering methods to group both users and items allowing each user and item to be in multiple clusters. The evaluation results show that *FMM* approach provides better prediction accuracy than *Pearson correlation* memory-based *CF* algorithm and the *aspect model* [121].

In general, clustering *CF* models have better scalability than the simple memory-based *CF* algorithm due to a fact that they make predictions on the reduced clustered rather than on the entire data set [94, 95, 122, 123]. These algorithms usually have time-consuming and expensive clustering phase which is performed off-line. However, the quality of produced predictions using these algorithms is poor. To improve the predictions, more advanced and fine tuned segments should be used. Note, however, that such mechanism degrade the performance and make the on-line prediction phase as expensive as finding similar entities using memory-based *CF* algorithms [124].

4.4.3 Regression-Based Collaborative Filtering Algorithms

The memory-based *CF* algorithms demonstrate a number of well known shortcomings. One of the main disadvantages for the memory-based *CF* is related to the fact that two rating vectors may have significantly different similarity rates depending on which similarity relation is applied. For example, two rating vectors may be very distant in the case the *Euclid distance* is used but they may have a very high similarity when the *vector cosine* or *Pearson correlation* similarity measure is applied. In order to address this issue, regression models are applied. Regression models are effective at making predictions on the numerical data sets which are very common in real-life *CF* prediction tasks.

A regression method uses an approximation of ratings to make predictions based on a regression model. Let $Y = (Y_1, Y_2, ..., Y_i, ..., Y_n)$ represents user preferences on n items. Then, $Y_i = (y_{i1}, y_{i2}, ..., y_{ij}, ..., y_{in})$ is the preference for the item i and each y_{ij} represents user j's rating on item i. Let $X_i = (x_{i1}, x_{i2}, ..., x_{ik})$ be a random variable representing preferences of

user k , which is a kind of user personal profile. For instance, in the case the data set contains movie ratings, each x_ik may be the user affinity for certain movie types such as crime, drama, thriller etc. It should be noted that $k << n$ is assumed. The linear regression model can be defined as follows:

$$Y = \Delta X + N, \tag{4.20}$$

where Δ is a $n \times k$ matrix. $N = (N_1, N_2, ..., N_n)$ is a random variable representing noise in user choices, X is a $k \times m$ matrix where each matrix column represents an estimate of the random variable X_i (user's personal profile in k-dimensional rating space) for a single user.

In [125], the authors propose a *sparse factor analysis*, that fills out the missing user-item pairs with *default voting* values (the average of some available values, either the average by rows, or by columns or by all) and uses linear regression to initialize the values for the *Expectation Maximization* (*EM*) iterations [126]. The experiments conducted on various data sets show that this approach scales better than the *Pearson correlation* memory-based *CF* algorithm and *Personality Diagnosis* (*PD*), which a hybrid *CF* representative algorithm [127], and manifests better prediction accuracy than *singular value decomposition* (*SVD*) [128]. Additional benefit of *sparse vector analysis* approach is that it supports privacy protection thanks to the ability to compute the encrypted user data.

Another regression based approach for dealing with the *CF* tasks on the numerical ratings data [129] searches for similarities between items, forms a collection of simple linear models, and combines them effectively to make predictions for missing user-item values. The authors used the *ordinary least squares* to estimate the parameters of the linear regression function. According to the evaluation results, the approach seems to be successful in addressing the sparsity, prediction latency and numerical prediction issues of memory-based *CF* algorithms.

A *slope one* regression based algorithm [130] is proposed to address the scalability of memory-based *CF* algorithms by improving the time that takes to produce the predictions.

4.4.4 MDP-Based Collaborative Filtering Algorithms

MDP-Based CF algorithms present collaborative filtering prediction tasks as the sequential optimization problems and use a *Markov decision process* (*MDPs*) formalism [131] to make predictions for *CF* tasks.

An *MDP* is a formal model for sequential stochastic decision problems often used in sit-

uations where each environment participant impacts its surroundings through decisions and actions. A *Markov decision process* can be defined as a four-tuple:

$$MDP = (S, A, R, P_r), \tag{4.21}$$

where, S is a set of process states, A is a set of actions (alternatively, A_s is a set of actions associated with the state s), R is an *expected* immediate reward function for each state/action pair, and P_r is a transition probability function between every pair of states given each action.

An optimal solution to the *MDP* is to maximize the function of its reward stream through the steps of iteration. The process starts with the initial policy that specifies the actions the decision maker will choose in state s:

$$\pi_0(s) = \arg\max_{a \in A} R(s, a). \tag{4.22}$$

At each step of iteration, the reward value function $V_i(s)$ is computed based on the previous policy, and the policy is updated with the new function value until the iteration will converge to an optimal policy [132, 133].

Shani et al. [134] use a *MDP* for the *CF* tasks where *MDP* states are k-tuples of items with some empty values representing missing values. The actions of the *MDP* correspond to the item recommendations and the rewards in the *MDP* are corresponding to item selling. The state following each recommendation is depending on weather the user buys recommended item, buys some other not recommended item, or buys nothing. It is assumed that the probability that a user buys an item depends on the current state, the associated item and weather the item is recommended or not, but it does not depend on other recommended items. The authors deployed *Mitos*, the recommendation system for on-line bookstore based on the designed *MDP*, and the recommender caused the bookstore to gain significantly higher profit than without using the recommender. Also, the *MDP*-recommender outperformed a simple *Markov chain* (*MC*) based recommender, which is a simplification of an *MDP* without actions.

The proposed *MDP* model can be described as approximation of a *partial observable MDP* (*POMDP*) [135]. Since the computational and representational complexity of *POMDPs* is very high, the researchers have proposed the approximate solutions to address these problems. The *POMDPs* approximate solutions can be classified in three very wide categories: *value function optimization* [136], *policy based optimization* [123, 137], and the most recent *stochastic sampling optimization* [138]. These strategies can be potentially applied to make predictions for *CF*

tasks.

4.4.5 Latent Semantic Collaborative Filtering Algorithms

A *Latent Semantic CF* algorithm is a statistical modeling technique that uses latent class variables in a mixture model with the aim to discover user communities and interest profiles. Basically, this approach decomposes user preferences using overlapping communities. This technique demonstrates higher prediction accuracy and better scalability over standard memory-based *CF* algorithms [139, 140].

The *aspect model* [121] is a probabilistic latent-space model that describes individual ratings as convex linear combination of rating factors. The latent class variable is associated with each particular user-item pair assuming that users and items are independent from each other given the latent class variable. The *aspect model* manifests much better performance that the clustering model working on *EachMovie* dataset [141].

A *multinomial model* is a simple probabilistic model for categorical data [14, 142] that assumes only one user type. On the other hand, *a multinomial mixture model* supports multiple types of users, and assumes that the rating variables are independent with each other and with the user identity given the user type [143]. The *user rating profile* (*URP*) [142] approach is a combination of the *multinomial mixture model* and the *aspect model* with a high-level generative semantics of generative probabilistic *Latent Dirichlet Allocation* (*LDA*) model [144]. The *URP* approach outperforms both the *multinomial mixture model* and the *aspect model* for *CF* prediction tasks.

4.4.6 Other Model-Based Collaborative Filtering Algorithms

Some applications require ordering of potentially desirable items for a specific user rather than classifying the items in the adequate clusters. In [145], the authors propose a two stage *order learning CF* approach which aims to optimize the order of recommended items. In this approach, the first stage learns a preference function by conventional means, and the second stage aims to order new set of instances by finding the total ordering that best approximates the preference function. Since the problem of finding the total ordering is *NP*-complete, a greedy-order algorithm is applied to approximate the optimal ordering function. The obtained evaluation results show that this approach performs better that a nearest neighbor *CF* algorithm and linear regression algorithm.

Another type of *CF* algorithms used for top-*N* recommendations rather than prediction and classification are *Association rule based CF* algorithms. In [94], the authors present their approach to find rules for developing top-*N* recommendation systems by using traditional *association rule mining* algorithm. In their approach, they choose all the rules that meet the thresholds for support and confidence values, sort items according to the confidence value of the rules in a way that the items predicted by the rules with higher confidence value are ranked higher, and finally select the first N highest ranked items as the recommended set. Fu et al. [146] proposed a web pages recommender that uses an apriori algorithm to mine association rules over the navigation history of the associated user. Leung et al. proposed another collaborative filtering framework based on fuzzy association rules and multi-level similarity [147].

A *maximum entropy approach* [148] clusters the data first, and then uses maximum entropy as an objective function to form a conditional maximal entropy model to make predictions.

A *dependency network* is a graphical model for probability relationships with a potentially cyclic graph. The probability component of a *dependency network* is a set of conditional distributions each belonging to a graph node given its parents. The advantage of the *dependency networks* over the *Bayesian belief nets* is that *dependency networks* make predictions faster. and require less memory and time to learn the model [111]. On the other hand, the *dependency networks* are less accurate than *Bayesian belief nets*.

Decision tree CF algorithms view the *CF* tasks as classification problem and they use a decision tree as the classifier [149].

Another graph-based model, *Horting*, uses a graph-based technique where each graph node presents user and edges between nodes represent similarity rate between users [150].

Multiple multiplicative factor models (*MMFs*) are the casual, discrete latent variable models that combine factor distributions multiplicatively and readily make predictions on missing data [151].

Probabilistic principal components analysis (*pPCA*) [125, 152] finds the principal axes of a set of observed data vectors using maximum likelihood estimation of parameters in a latent variable model similar to factor analysis.

To improve scalability and data sparsity issues of *CF* tasks, the researchers proposed *matrix factorization* based *CF* algorithms which are proven to be effective [153–155].

While developing their model-based collaborative filtering, Wang et al. proposed probabilistic relevance *CF* models [156, 157] that gain benefits from information retrieval theories

and models.

4.5 Hybrid Collaborative Filtering Approaches

Hybrid collaborative filtering approaches combine *CF* with other recommendation methods (mostly with content-based recommendation systems) to make predictions for *CF* tasks.

Content-based recommendation systems make predictions by analyzing the semantics of the content of textual information. The textual information is typically comprised in textual documents, news, feeds, URLs, messages, web logs, item's labels and descriptions, and profiles containing user's preferences, tastes and needs. All above mentioned textual information is analyzed in order to extract regularities in the content which can be used to make predictions [158]. The importance of the textual content is determined by various elements such as observed browsing features of the words or pages (e.g. term frequency and inverse document frequency), and similarity between items liked by user previously [159]. In this case content-based recommender uses heuristic methods and classification algorithms to make predictions [160]. The *start-up* problem reflected in a fact that the content-based recommender must have enough information to build a reliable classifier is common to all content-based techniques [160]. Another limitation of content-based approaches is the tight and explicit association with the objects they recommend and the information is sometimes hard to extract, while *CF* approaches can make predictions and recommendations without any explicit description and meta-data. The *overspecialization* problem is another common shortcoming of content-based techniques, that is, they can only recommend items that highly match along with user's profile or his/her rating history [161, 162].

There are other hybrid recommender systems such as *demographic-based* recommender systems, which use some personal user information such as gender, postcode, occupation, etc. to make user profiles [163], *utility-based recommender systems* and *knowledge-based recommender systems*, which both require the information about how a particular item can be useful to users [20, 164].

In general, hybrid *CF* algorithms aim to improve prediction performance and overcome the limitations of recommender systems by: adding content-based features to *CF* models, adding *CF* characteristics to content-based models, combining *CF* techniques with content-based or other models, or combining different *CF* algorithms [161, 165].

4.5.1 Hybrid Recommenders Combining Collaborative Filtering and Content-Based Features

The *content-boosted CF* algorithm uses *naive Bayes* to classify the content, and then fills in the missing values of the rating matrix with the predictions of the content predictor to form a *pseudo rating matrix*. Thereby, the observed values are preserved and the missing rating values are replaced by the predictions of the content predictor. In the next step, the algorithm makes predictions over the *pseudo rating matrix* using *weighted Pearson correlation*-based *CF* algorithm, which gives higher impact to the items rated by more users, and the ones rated by the active user [166]. The *content-boosted CF algorithm* has improved performance over some pure content-based algorithms and some pure memory-based *CF* algorithms as well. Additionally, this approach addresses the *cold start* issue and *sparsity* problem common to all memory-based *CF* approaches. In [108], the authors used *TAN-ELR* as the content predictor and directly applied *Pearson correlation based CF* instead of *weighted Pearson correlation*-based *CF* algorithm on the *pseudo rating matrix* to make predictions. Evaluated on the reasonably reduced data subsets instead on the entire data set, their approach improved *CF* prediction accuracy.

In [167], the authors propose a Bayesian preference model that statistically integrates various useful information for making predictions, such as user preferences, user and item profiles and expert remarks. In their approach they use *Markov chain Monte Carlo* (*MCMC*) methods [168] for sampling-based inference and their approach provides better prediction accuracy than pure memory-based *CF* algorithms.

Balabanovic et al. propose a recommender *Fab* [165] which maintains user profiles in web pages using content-based techniques, and then applies *CF* techniques to identify profiles with similar tastes. It can then recommend documents and web pages across user profiles. Sarwar et al. [169] implemented a set of knowledge-based *filter bots* representing artificial users which base their opinion using certain criteria. For instance, the example of *filter bot* is a *genre bot* that rates items according to their genre. The "jazz bot" would rate any CD in the jazz section with a high score, while any other CD that is not in the jazz section would get low rates. In [170], the authors use predictions from *CF* algorithms as the input for the content-based recommender. Condiff et al. [162] propose a Bayesian mixed model that integrates user ratings along item's features in a single unified framework to make predictions. The *CF* recommender *Ripper* uses both user ratings and item's features to provide predictions [103].

4.5.2 Hybrid Recommenders Incorporating Collaborative Filtering and Other Recommendation Systems

A *weighted hybrid recommender* incorporates various recommendation techniques by their weights, which are calculated using the approach employed in [20]. The combination can be linear, the weights can be fine tuned [171], and *weighted majority voting* [158, 172] or *weighted average voting* [173] can be used. The *P-Tango* system [171] gives *CF* and content-based technique equal weights at the beginning, but then gradually fine tunes the weights according to the feedback obtained from users while making on-line predictions. The *boosting* approach [174] applies the similar strategy as the one used in *P-Tango*.

A *switching hybrid recommender* switches between different recommendation approaches according to some criteria such as confidence levels from various used recommendation techniques. Once certain recommendation system can not produce predictions according to the required confidence level, then another recommendation technique is activated. Switching recommendation systems may improve predicting performance but also introduce the complexity of parametrization for the switching criteria [20].

Some other proposed recommendation systems in this category include *mixed hybrid recommenders* [175], *cascade hybrid recommenders* [20], *meta-level recommenders* [20, 158, 165, 176].

The evaluation results gained in the variety of experiments conducted to compare the performance of the pure *CF*, content-based methods and hybrid recommendation systems support the claims that hybrid-based recommendation systems may improve prediction accuracy, especially for the *new user* and the *new item* situations, in which pure *CF* approaches can not produce quality predictions. Note, however, that hybrid recommendation systems require additional external information that does not have to be available in the general case, and also, the evaluation results confirm that hybrid-based recommendations systems increase the computational complexity of the prediction [20, 158, 177].

4.5.3 Hybrid Recommenders Based on Combination of Other Collaborative Filtering Algorithms

The basic categories of *CF* approaches, memory- and model-based *CF* approach, can be combined to form a hybrid *CF* approach. In general, the prediction/recommendation perfor-

mance of such an approach is better when compared to the pure memory-based or model-based *CF* approach [127, 178].

Probabilistic memory-based collaborative filtering (*PMCF*) uses both memory-based and model-based techniques [178]. In this approach a mixture model is introduced on the basis of a set of stored user profiles, and then the posterior distribution of user rankings is used to produce predictions. The *new user* problem is tackled by adding an active learning extension to the *PMCF* that queries a user for additional information if an insufficient amount of information is available. The computational performance is addressed by selecting the small subset of user rankings called *profile space*. Then, the prediction is produced based on the limited profile space rather then considering the entire database of user rankings. The evaluation results show that *PMCF* has better accuracy than *Pearson correlaton* memory-based *CF* and the *model-based CF using naive Bayes*.

Personality diagnosis (*PD*) is another popular hybrid *CF* approach that mixes memory-based and model-based *CF* techniques and demonstrates some advantages against both algorithms [127]. In this approach, the active user is generated by employing the random selection of some other user and adding Gaussian noise to the selected user's ratings. The probability that the active user belongs to some "personality type", and the probability that the active user likes a certain item can be computed considering user's known ratings. The *PD* can be seen as a clustering method with exactly one user per one cluster. While performing predictions on *EachMovie* [141] and *CiteSeer* [179] data sets, *PD* demonstrated better prediction accuracy than *Pearson correlation* and *vector similarity* memory-based *CF*, and *Bayesian clustering* and *Bayesian networks* model-based *CF* as well [14].

In general, a hybrid *CF* approach can be used to improve prediction accuracy in situations in which an ensemble of classifier can produce more accurate predictions than a member classifier. Thus, a hybrid *CF* approach that combines more different *CF* techniques can be useful to improve the prediction accuracy for the *CF* tasks [173].

4.6 Characteristics and Challenges in Different Collaborative Filtering Approaches

The basic purpose and most important functionality of every prediction or recommendation system is to provide accurate and real-time predictions or recommendations, which will bring

CF Category	Representative Approaches	Advantages	Disadvantages
Memory-based CF	• Neighbor-based CF (item– or user–based CF with Pearson or vector cosine correlation) • Item– or User–based top-N recommendations	• easy implementation • simple adding of new data records • content independent recommendation of items • scale well for co-rated items	• dependent on human ratings • low prediction accuracy for sparse data • new item and new user issue • do not scale for large data sets
Model-based CF	• Bayesian belief nets CF • clustering CF • MDP-based CF • latent semantic CF • sparse factor analysis • SVD-based CF	• improve scalability and data sparsity issues • better prediction accuracy • sparse factor analysis • SVD-based CF	• expensive model building • trade-off between scalability and accuracy • lose information for dimensionality reduction techniques
Hybrid CF	• content-based CF • content-boosted CF • hybrid CF combining model– and memory–based CF algorithms	• addresses drawbacks of CF and content-based recommenders • improve prediction accuracy • provide solutions for data sparsity and gray sheep issues	• have increased complexity and expensive to implement • need additional external information usually not trivial to collect • provide solutions for data sparsity and gray sheep issues

Table 4.1: A brief overview of different Collaborative Filtering Types

satisfaction to the users of the system, and benefits to the company as well. Table 4.1 provides a brief overview of all collaborative filtering types, which are described in details in previous sections. The overview summarizes typical representative approaches, main advantages and disadvantages for each *CF* category.

The collaborative filtering based prediction systems are evaluated depending on how well they cope with challenges which are common to all collaborative filtering approaches. Some of the most common challenges of these approaches include: *data sparsity*, *scalability*, *accuracy in dynamic environments*, *synonymy*, *gray sheep* and *shilling attacks*. The rest of the section analyzes each of these challenges, especially from the aspect of collaborative filtering appliance for prediction of web services reliability.

4.6.1 Data Sparsity

In real-world recommendation systems, collaborative filtering is used to produce predictions or recommendations on a considerably large amount of data, including both substantially large number of users and items. In addition, each user in the system is determined with her/his personal interest, and uses a very limited subset of items according to her/his preferences. Also, in a real recommendation environment, new users and new items emerge in real-time. Thus, the user-item matrix is often very sparsely-populated with a number of missing cells capturing the expected ratings whose values needs to be predicted.

The *data sparsity* issue is manifested in different situations. More specifically, the data sparsity issue is related to the *cold start* problem, which occurs when a new user or a new item appears in the system. In such situations, it is very difficult to find similar entities and produce predictions for the new entity due to a lack of information about it (the cold start issue is also called *new user problem* or *new item problem* in the literature [161, 178]). The percentage of entities for which the recommendation algorithm can produce predictions is called *coverage*. The *reduced coverage* problem appears when the number of ratings is very small compared to the number of items in the system. In situations like this, it is impossible for the recommendation system to produce quality predictions. *Neighbor transitivity* is related to the data sparsity problem, in which users with similar preferences have not both rated the same items. In this case, the efficiency of the recommendation systems which are based on comparing users in pairs, is reduced.

To address the data sparsity challenge, many approaches have been proposed. *Singular Value Decomposition* (*SVD*) [128], is a dimensionality reduction technique that removes unrepresentative or insignificant entities to reduce the dimensionality in the user-item matrix. The representative and patented approach based on *SVD* is *Latent Semantic Indexing* (*LSI*) [180, 181], where similarities between users are computed by the representation of the users in the reduced space. In [88], the authors developed the *eigentaste*, which is based on *Principle Component Analysis* (*PCA*), closely related to the factor analysis technique first proposed by Pearson in 1901 [182], to reduce dimensionality. Note, however, that dimensionality reduction discards certain entities and useful information about them, which may impact the quality of predictions or recommendations [94, 124].

Hybrid-based *CF* approaches, such as *content-boosted CF* algorithm [166], are quite effective in addressing data sparsity issues. In such approaches, the external content information is

brought in the system to produce predictions for *new users* and *new items*. To address the data sparsity, in [183], the authors propose exact product classification according to the taxonomic information based on profiles creation through inference of super-topic score and topic diversification. To tackle *cold start issue*, Shein et al. proposed the *aspect model latent variable* method that combines both *CF* and content information in model fitting [184]. Kim and Li proposed a probabilistic model to address the *cold start issue* by classifying items into groups and predictions are produced for users, considering the Gaussian distribution of user ratings [185].

Model-based *CF* approaches, such as *TAN-ELR* (tree augmented naive Bayes optimized by extended logistic regression [107, 108]), provide more accurate predictions for sparse data. The recently proposed model-based *CF* approaches use *association retrieval technique*, which includes retrieval framework and related algorithms to explore transitive associations among users considering their rating history [186]. Some of the above mentioned approaches include: *Maximum margin matrix factorization* (*MMMF*), a convex, infinite dimensional alternative to low-rank approximations and standard factor models [153, 154], ensembles of (*MMMF*) [187], multiple imputation-based *CF* approaches [188], and imputation-boosted *CF* algorithms [97].

All afore mentioned *data sparsity* related issues are present in service-oriented systems, underlying the Consumer Computing environment, in which the reliability of consumer applications needs to be predicted. In terms of web services reliability prediction, where each value in the user-item matrix represents the reliability experienced by the user u on the service (item) i, the issues related to the data sparsity are present. In real-service oriented systems, having millions of users and services, new users and services emerge in real-time. Also, a certain user visits only a very limited subset of services, according to her/his preferences and interest. Thus, the user-item matrix is extremely sparse and there is a significant amount of missing values that needs to be predicted. The *new user* and *new item* issues closely related to the *cold start* problem are also present and even more emphasized. For instance, in case a very new service appears in the system, the internal complexity and the implementation details of the service are usually not provided. Additionally, some environment related parameters such as service provider load are commonly not available for the very new services and providers. The same remarks can be identified when a very new user enters the system. A lot of user-specific parameters highly influence the perceived service reliability such as: user's physical device, geographic location, network bandwidth or personal usage profile etc.

4.6.2 Scalability

In general, the collaborative filtering approaches suffer from potentially serious scalability issues when the numbers of existing users and items substantially grow. More specifically, these approaches require storing ratings for each user-item pair in a matrix. In a real-world recommendation system, given millions of users and items, these approaches demand a great amount of computational resources which causes potential scalability issues. In addition, a real recommendation system needs to react in real-time while producing predictions for all users in the system regardless of their rating history, which requires high scalability of a *CF* system [124].

The techniques of dimensionality reduction such as *SVD* can deal with the scalability related issues and produce predictions and recommendations fast, but they have to perform expensive matrix factorization steps. An incremental *SVD CF* algorithm [189] performs decomposition using the existing users. Once a new set of ratings are added to the system, the algorithm uses the *folding-in* projection technique [181, 190] to build an incremental system without re-computing the low-dimensional model from scratch. Hence, this technique makes the recommender more scalable.

Memory-based *CF* algorithms, such as the item-based *Pearson correlation* algorithm (*IPCC*) can achieve respectable scalability in one of its variations. Instead of computing similarities between all pairs of items, this variation of *IPCC* calculates the similarity only between the pair of co-rated items by the active user [15, 124]. A simple Bayesian *CF* algorithm addresses the scalability issue by producing predictions based on observed ratings [105]. Model-based *CF* approaches, such as clustering *CF* algorithms, improve scalability by reducing the data set size and considering users for recommendation within the smaller and similar clusters instead the entire data set [95, 117, 118, 122]. However, these approaches present a trade-off between scalability and prediction accuracy.

In real service-oriented systems, having tremendously large number of users and services, the collaborative filtering tasks for prediction of nonfunctional service properties suffer from serious scalability issues. In the Consumer Computing environment, consumers need to select services (underlying consumer applications) with appropriate reliability properties in real-time. Hence, the prediction and recommendation approaches should provide fast and accurate predictions in real-time and demonstrate higher scalability than the existing *CF* approaches.

4.6.3 Dynamic Environments

Another important challenge that is very specific to collaborative filtering based approaches is related to the low prediction accuracy issue in dynamic environments. Regarding the prediction accuracy, collaborative filtering provides accurate recommendations in static environments where the collected data records are relatively stable. This means that the records remain up to date for a reasonably long period of time (e.g. *movie ratings*, *product recommendations*).

However, service-oriented systems, which are infrastructure underneath the consumer computing, are deployed on the Internet which is a very dynamic environment in which service providers register significant load variations during the day [65–67]. In such a dynamic environment, user perceived service reliability may considerably change depending on the invocation context parameters (already described in Section 3.3.3).

4.6.4 Synonymy

The challenge of *Synonymy* is related to the fact that the same or very similar items may have different names. The recommendation systems usually do not have the ability to detect the latent connection between such items, and commonly threat them as different items. For instance, items labeled "*movie*" and "*film*" have different names but have the same meaning and actually present the same item. However, the collaborative filtering systems do not discover the equivalence between same items and threat them differently, which results in decreasing of prediction and recommendation performance.

The first attempts to address the synonymy challenge used intellectual or automatic terms expansion, or the construction of a thesaurus. The disadvantage of these automated approaches is that some new added terms might get associated with some other meanings different from those originally intended, which again results in degrading of prediction and recommendation performance [191].

The *SVD* techniques, especially *Latent Semantic Indexing* (*LSI*), demonstrate promising results in dealing with synonymy challenge. *SVD* uses a large matrix of term-document association data and constructs a *semantic* space where terms and documents that are closely related are placed closely to each other. The performance of *LSI* in coping with the synonymy challenge is significant at higher *recall* levels where precision is usually low, while the performance at lower *recall* levels is rather poor than impressive *deerwester1990indexing*.

Also, the *LSI* approach provides only a partial solution to the *polysemy* challenge, which is

related to the fact that respectable amount of words and terms commonly have more than one distinct meaning [181]

In service-oriented systems, underlying consumer computing, collaborative filtering is used to predict the values of nonfunctional properties (such as reliability, availability etc.) of services. In such an environment, the synonymy challenge is not so relevant because each service is a unique resource on the Internet, accessible via public interfaces and located at a unique *URL*.

4.6.5 Gray Sheep

The *Gray sheep* issue is related to the fact that there are users whose opinions do not consistently match with any group of people which makes the recommendation for those people more difficult [171]. On the other hand, the *Black sheep* are the group of people whose individual tastes make high quality recommendations almost impossible. Although, the recommendation systems fails in this case, *black sheep* users are considered as an acceptable failure [192].

In [171], the authors propose a hybrid approach using content-based and *CF* recommendations and producing the prediction as a weighted average of content-based and *CF* prediction, in which the weights of the content-based and *CF* predictions are determined on a per-user basis, allowing the system to balance the optimal mix of content-based and *CF* recommendation for each user, and thus, improve the performance for the *gray sheep* users.

The *gray sheep* phenomena is present in case the collaborative filtering technique is applied for predicting the reliability in service-oriented computing. In fact, the prediction process is more difficult for those entities which do not belong to any of distinct entity groups.

4.6.6 Shilling Attacks

In the systems where everyone can provide ratings, people may give a lot of positive ratings for their own content and negative ratings for their competition. Successful recommendation *CF* systems should provide mechanisms to disable this kind of activities [193].

The *shilling attacks models* for collaborative filtering have been studied recently and their effectiveness has been analyzed. In [194], the authors discovered that item-based collaborative filtering recommendation systems are more resilient to shilling attacks that user-based approaches, and they suggest that new approaches should be used to mitigate shilling attacks on recommendations systems. Mobasher et al. over-viewed the shilling models for item-based

approaches, and the authors claim that alternative *CF* algorithms such as model-based and hybrid *CF* approaches, provide the partial solutions to the bias ratings injection problem [195]. In [196], the authors addressed the shilling attacks by analyzing recommender's immunity to potentially malicious manipulations in user-item matrix.

Bell and Koren [197] propose a comprehensive approach for solving shilling attacks by removing global effects in the data normalization stage of neighbor-based *CF*, and thus, working with residual of global effects to select neighbors.

When services reliability prediction is considered, the collaborative filtering output is used to select the most eligible items according to their reliability values, and compose them to support more advanced functionalities as services compositions. It is extremely important to predict the reliability of each component as accurately as possible, since the errors in prediction propagate on the output as even greater errors when composite reliability is computed. Thus, any possible malicious or biased manipulations with the collected reliability data would compromise the entire process of composite services development, which marks the shilling attacks as an important challenge in this context.

4.6.7 Other Challenges

People usually do not want to share their personal preferences and interests, and make them publicly accessible, which raises personal privacy as an important challenge in collaborative filtering. Some of the proposed approaches, such as [125, 198], try to address the privacy issues in *CF* algorithms.

Another important challenge is noise which gets increased as the population of users grows and becomes more diverse. Some proposed techniques, such as ensembles of *MMMF*s [187], instance selection [199] find a way to reduce the problem of increased noise in *CF*. When knowledge representation and classification tasks are considered, *Dempster-Shafer* (*DS*) theory [200, 201] and imputation techniques [202] are found useful for processing imperfect or noisy data, which suggests that those techniques could be employed to address the noise issue in *CF*.

Explainability is a very important feature of the recommendation systems. The idea is to provide the users an explanation why certain items are recommended, such as "you might like this item A because you previously liked items B and C". Although such an simple explanation might not be completely accurate, most of the users will find it useful and beneficial [203].

The privacy challenge is also very important aspect in consumer computing, since the av-

erage consumers prefer to keep their personal preferences private but still get quality recommendations and predictions. Regarding explainability, it is crucial to provide an explanation to the consumers reasoning why a particular component is recommended. For instance, while creating his composite application, the consumer might be in doubt which of the recommended components to choose. The recommender's explanation like "the component A is very likely to be more reliable for you than the component B" would be very appealing and helpful during the selection process.

Chapter 5

LUCS - Model for Prediction of Application's Reliability

In this chapter, *LUCS*, a model for service reliability prediction is presented [24]. Section 5.1 provides a brief overview of the model, introduces the model parameters and describes the reliability prediction process. Section 5.2 provides a detailed mathematical description of the model.

5.1 *LUCS* Overview

This section overviews *LUCS* (service **L**oad, **U**ser location, service **C**lass, **S**ervice location), prediction model that utilizes contextualized information about the user and service to obtain a more accurate prediction of service reliability. For example, different users may be in different geographic locations with different network capabilities thus resulting in varying service reliabilities. Section 5.1.1 introduces different *LUCS* parameters while Section 5.1.2 illustrates the overall process for predicting the service reliability.

5.1.1 Model Parameters

In the *LUCS* model the following parameters are used to predict the service availability: the users' and services' geographic locations, the load of the service provider at the actual time of the invocation, and the computational requirements of the invoked service.

Service users are grouped into a finite set of user groups by their geographic locations using their IP address and this set is named U *(User locations)*. Generally, these groups may be

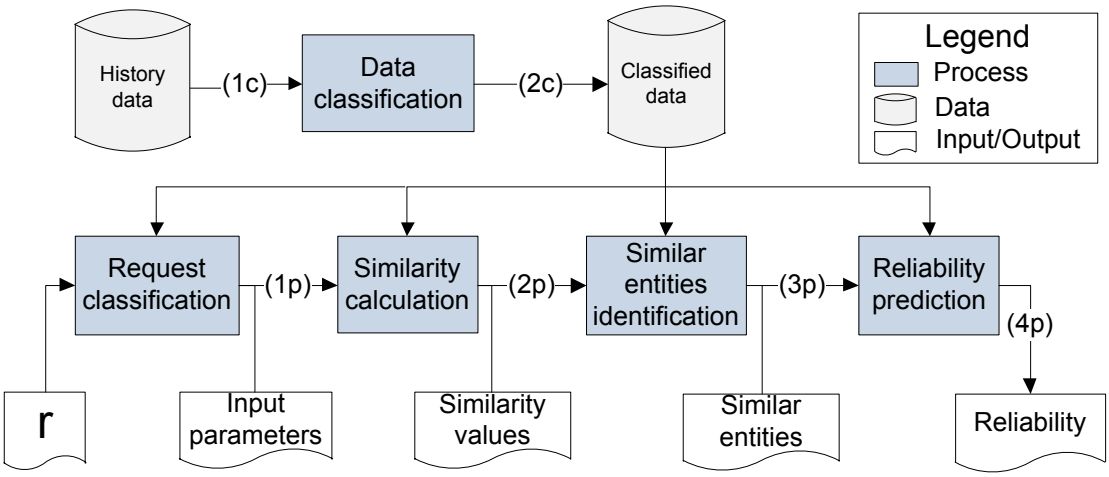


Figure 5.1: The process of reliability prediction using *LUCS*.

determined using the appropriate granularity (e.g., continent, region, city or suburb) depending on the requirements and the scale of the designed system.

Additionally, services are grouped into a finite set of service groups by their geographic locations and this set is named S *(Service locations)*. Similarly like user groups, the appropriate granularity should be selected with respect to the system requirements and its scale.

Furthermore, collected records about previous service invocations are grouped into a finite set of service provider loads by the actual time of the service invocation and this set is named L *(Loads)*. The service load is defined as the number of requests received per second. In general, the load may be determined according to the load curve obtained from the service provider. For example, the service provider may provide a curve that defines time windows related to the times of the day, where each window has an assigned average load.

Finally, services are grouped into service classes with respect to their computational requirements and this set is named C *(Service classes)*. For evaluation purposes, the amount of memory required for the service execution is used to define different service classes. Note, however, that model is general and different grouping criteria (e.g., CPU demand) may be used.

5.1.2 Reliability Prediction Process

Figure 5.1 depicts the proposed service reliability prediction process, which starts with the classification of the collected past invocation data into the sets related to the model parameters. The process of data classification is performed only once as a step preceding the prediction (1c, 2c). Once the collected data is classified, the service reliability prediction is performed through

four steps executed in a pipeline (1p–4p in the flowchart).

The input of the prediction process is a specific invocation r for which the reliability of the invoked service needs to be predicted. Each prediction step in the pipeline uses the classified data. In the first step (1p), an ongoing invocation is classified in the four-dimensional parameter space. In the second step (2p), similarity relations among entities are calculated. In the third step (3p), the entities most similar to the current invocation are chosen based on the similarity relations obtained in the second step. In the final step (4p), the missing reliability values are estimated using the known reliability values of the most similar entities.

5.2 Formal Definition of *LUCS*

This section formally defines *data classification* process and four prediction steps of the *LUCS* reliability prediction process from Figure 5.1.

5.2.1 Data Classification

The service invocation r is formally defined with its properties as:

$$r(s, u, t). \tag{5.1}$$

In the above Equation 5.1, s is the service to be invoked, u is the user performing the service invocation and t is the actual time of the service invocation. These properties of the service invocation are used in *LUCS* model to determine the input parameters for the reliability prediction (e.g., information about the user u contains the location information).

A special function f that determines the user location group of a user associated to a given service invocation r is defined. The grouping is done according to the IP of the user:

$$f : r(s, u, t) \rightarrow u_r, u_r \in U. \tag{5.2}$$

Similarly, a special function g determines the service location group of the service associated to a given service invocation r using the IP of the service:

$$g : r(s, u, t) \rightarrow s_r, s_r \in S. \tag{5.3}$$

In addition, a special function h that determines the service provider load using the load curve of the provider and the actual time of the invocation for a given service invocation r is defined:

$$h : r(s, u, t) \rightarrow l_r, l_r \in L \cup \{null\}. \tag{5.4}$$

Function h leverages the statistics collected from the past invocation data and the load curve obtained by the provider to estimate the actual load on the service provider. When the provider load is unavailable, h simply returns a $null$ value.

Finally, a special function j determines the computational service class for a given service invocation r using the data about internal service complexity published by the provider:

$$j : r(s, u, t) \rightarrow c_r, c_r \in C \cup \{null\}. \tag{5.5}$$

In cases when the service class cannot be determined (e.g., the invoked service is new), j returns a $null$ value.

In order to perform the prediction for future service invocations the collected past invocations data is classified in a four dimensional space $D[u \times s \times l \times c]$ using the functions defined in the Equations 5.2 – 5.5. The dimensions u, s, l, c are mapped to the sets defined in the *LUCS* model: **U**ser location, **S**ervice location, service provider **L**oad, service **C**lass computational requirement.

5.2.2 Request Classification

Consider a new service invocation $r(s, u, t)$ whose availability p needs to be predicted. First the invocation r is classified using the functions defined in the Equations 5.2 – 5.5 and the groups u_r, s_r, l_r and c_r are determined for the invocation. These groups are then used as input parameters for predicting the reliability. Below, formally specification describing the process of reliability calculation is presented. Separate definitions are provided for the cases when some of the parameters are not available and the cases when all of the parameters are available.

For the cases in which either one of the parameters l_r or c_r is unavailable (i.e., has an

assigned value $null$), the prediction is calculated as follows:

$$p = \begin{cases} \overline{D[u_r, s_r]}, & l_r = null \wedge c_r = null \\ \overline{D[u_r, s_r, c_r]}, & l_r = null \wedge c_r \neq null \\ \overline{D[u_r, s_r, l_r]}, & c_r = null \wedge l_r \neq null \end{cases} \tag{5.6}$$

In the Equation 5.6, the $\overline{D[u_r, s_r]}$ presents the average availability value for the user location u_r and service location s_r; $\overline{D[u_r, s_r, c_r]}$ is the average availability value for the user location u_r, service location s_r and service class c_r; and $\overline{D[u_r, s_r, l_r]}$ is the average availability value for the user location u_r, service location s_r and load l_r.

For the cases in which both l_r and c_r are available (i.e., not $null$), two cases depending of whether the particular user u has previously invoked the service s are distinguished. If such an invocation previously occurred and there exists sufficiently extensive history data for that particular user-service invocation in the four-dimensional space $D[u_r, s_r, l_r, c_r]$, the value produced with the Equation 5.7 is the reliability.

$$p = D[u_r, s_r, l_r, c_r] \tag{5.7}$$

In the case when the history data is not available for that particular invocation in the four-dimensional space $D[u_r, s_r, l_r, c_r]$, the reliability is predicted via the steps described in the following sections.

5.2.3 Calculating Similarity Relations

To predict reliability of a new service invocation with no prior invocation data, the similarity of that invocation with other previously observed invocations is utilized.The similarity relations between the different entities are defined using the *Pearson Correlation Coefficient (PCC)*, which is a correlation metric widely used in different recommendation systems [17, 20–23]. Intuitively, PCC measures the statistical similarity between two different entities represented with

generic variables e_1 and e_2.

$$Sim_e(e_1, e_2) = \frac{\sum_{h \in H}(D[e_1, h] - \overline{D[e_1]}) \times (D[e_2, h] - \overline{D[e_2]})}{\sqrt{\sum_{h \in H}(D[e_1, h] - \overline{D[e_1]})^2} \sqrt{\sum_{h \in H}(D[e_2, h] - \overline{D[e_2]})^2}} \tag{5.8}$$

The following sections discuss how the *PCC* is used to calculate the similarity of user locations, service locations, service loads, and service classes. Similarity in this case denotes the statistical correlation of the experienced service reliabilities.

User location similarity

To calculate the similarity $Sim_u(u_1, u_2)$ of user locations u_1 and u_2, the substitution $e_1 = u_1$ and $e_2 = u_2$ is performed in the similarity relation presented with the Equation 5.8:

$$Sim_u(u_1, u_2) = \frac{\sum_{h \in H}(D[u_1, h] - \overline{D[u_1]}) \times (D[u_2, h] - \overline{D[u_2]})}{\sqrt{\sum_{h \in H}(D[u_1, h] - \overline{D[u_1]})^2} \sqrt{\sum_{h \in H}(D[u_2, h] - \overline{D[u_2]})^2}} \tag{5.9}$$

where H is defined as:

$$H = \{(s, l, c) \in [s \times l \times c] | \exists D[u_1, s, l, c] \wedge \exists D[u_2, s, l, c]\}. \tag{5.10}$$

In the Equation 5.9, $D[u_1, s, l, c]$ and $D[u_2, s, l, c]$ are the reliabilities perceived by users at locations u_1 and u_2 while accessing services of a class c at the service location s during the provider load l. $\overline{D[u_1]}$ and $\overline{D[u_2]}$ are the average reliabilities perceived by users at locations u_1 and u_2. Equation 5.10 defines a set of triples H where both values $D[u_1, s, l, c]$ and $D[u_2, s, l, c]$ need to be available in D for them to contribute to the similarity measure. The calculated similarity value $Sim_u(u_1, u_2)$ is in the interval $[-1, 1]$, where higher value indicates a higher degree of similarity between the entities.

Service location similarity

The substitution of $e_1 = s_1$ and $e_2 = s_2$ in the similarity Equation 5.8 provides $Sim_s(s_1, s_2)$, the similarity relation for two different service locations s_1 and s_2

$$Sim_s(s_1, s_2) = \frac{\sum_{h \in H}(D[s_1, h] - \overline{D[s_1]}) \times (D[s_2, h] - \overline{D[s_2]})}{\sqrt{\sum_{h \in H}(D[s_1, h] - \overline{D[s_1]})^2} \sqrt{\sum_{h \in H}(D[s_2, h] - \overline{D[s_2]})^2}} \tag{5.11}$$

where H is defined as:

$$H = \{(u, l, c) \in [u \times l \times c] | \exists D[u, s_1, l, c] \wedge \exists D[u, s_2, l, c]\}. \tag{5.12}$$

In the Equation 5.11, $D[u, s_1, l, c]$ and $D[u, s_2, l, c]$ are the reliabilities for services at locations s_1 and s_2, both of service class c, perceived by users at the location u, during the provider load l. $\overline{D[s_1]}$ and $\overline{D[s_2]}$ are the average reliabilities perceived by users at service locations s_1 and s_2. The Equation 5.12 defines a set of triples H where both values $D[u, s_1, l, c]$ and $D[u, s_2, l, c]$ need to be available in D to contribute to the similarity calculation.

Service load similarity

To calculate the similarity $Sim_l(l_1, l_2)$ of the loads l_1 and l_2 for two different service loads, the substitution $e_1 = l_1$ and $e_2 = l_2$ is performed in the similarity Equation 5.8:

$$Sim_l(l_1, l_2) = \frac{\sum_{h \in H}(D[l_1, h] - \overline{D[l_1]}) \times (D[l_2, h] - \overline{D[l_2]})}{\sqrt{\sum_{h \in H}(D[l_1, h] - \overline{D[l_1]})^2} \sqrt{\sum_{h \in H}(D[l_2, h] - \overline{D[l_2]})^2}} \tag{5.13}$$

where H is defined as:

$$H = \{(u, s, c) \in [u \times s \times c] | \exists D[u, s, l_1, c] \wedge \exists D[u, s, l_2, c]\}. \tag{5.14}$$

In the Equation 5.13, $D[u, s, l_1, c]$ and $D[u, s, l_2, c]$ are the reliabilities during the service provider loads l_1 and l_2, perceived by users at the location u while accessing services of class c at the location s. $\overline{D[l_1]}$ and $\overline{D[l_2]}$ are the average reliabilities perceived by users during the service provider loads l_1 and l_2. The relation (5.14) defines a set of triples H where both

values $D[u, s, l_1, c]$ and $D[u, s, l_2, c]$ need to be available in the D to contribute to the similarity calculation.

Service class similarity

Finally, the substitution of $e_1 = c_1$ and $e_2 = c_2$ in the similarity Equation 5.8 provides $Sim_c(c_1, c_2)$, the similarity relation for two different service classes c_1 and c_2:

$$Sim_c(c_1, c_2) = \frac{\sum_{h \in H}(D[c_1, h] - \overline{D[c_1]}) \times (D[c_2, h] - \overline{D[c_2]})}{\sqrt{\sum_{h \in H}(D[c_1, h] - \overline{D[c_1]})^2} \sqrt{\sum_{h \in H}(D[c_2, h] - \overline{D[c_2]})^2}} \tag{5.15}$$

where H is defined as:

$$H = \{(u, s, l) \in [u \times s \times l] | \exists D[u, s, l, c_1] \wedge \exists D[u, s, l, c_2]\}. \tag{5.16}$$

In the Equation 5.15, $D[u, s, l, c_1]$ and $D[u, s, l, c_2]$ are the reliabilities perceived by users at the location u while accessing services of classes c_1 and c_2 at the locations s during the provider load l. $\overline{D[c_1]}$ and $\overline{D[c_2]}$ are the average reliabilities perceived by users while accessing services of classes c_1 and c_2. The relation (5.16) defines a set of triples H where both values $D[u, s, l, c_1]$ and $D[u, s, l, c_2]$ need to be available in the D to contribute to the similarity calculation.

5.2.4 Determining Similar Sets of Entities

In Section 5.2.3 the relations that measure similarity between entities considered in the proposed model are defined. In this section, the subsequent step that determines the most similar sets of entities is discussed. Similar like in the previous sections, the generic equation is presented to describe sets of the most similar entities for a given service invocation r. Using the similarity relation captured in the Equations 5.9, 5.11, 5.13 and 5.15, the similarity of the current entity e_r with all the others corresponding entities is calculated. According to the given PCC values, a ranking of the entities according to their similarity values is performed. The dissimilar entities have negative PCC values and they are not taken into consideration. The top

k similar entities are selected into the set of the most similar entities $S_e(e_r)$.

$$S_e(e_r) = \{e \in E | \exists D[e, h_r] \wedge (Sim_e(e, e_r) \geq Sim_e(e_k, e_r)) \wedge (Sim_e(e, e_r) > 0) \wedge (e \neq e_r)\} \tag{5.17}$$

The Equation 5.17 defines the set of similar entities $S_e(e_r)$, where E represents the set of entities, e_r stands for the current entity, e_k is the entity with k^{th} largest PCC value and the condition $Sim_e(e, e_r) > 0$ stands for the exclusion of dissimilar entities.

By substituting the actual variables u, s, l, c in place of the generic variable e and the actual sets U, S, L, C in place of the generic set E used in Equation 5.17, the following sets of the most similar entities for a given service invocation r are determined:

- Set of most similar user locations:

$$\begin{aligned} S_u(u_r) = &\{u \in U | \exists D[u, h_r] \wedge (Sim_u(u, u_r) \geq Sim_u(u_k, u_r)) \\ &\wedge (Sim_u(u, u_r) > 0) \wedge (u \neq u_r)\}, \end{aligned} \tag{5.18}$$

- Set of most similar service locations:

$$\begin{aligned} S_s(s_r) = &\{s \in S | \exists D[s, h_r] \wedge (Sim_s(s, s_r) \geq Sim_s(s_k, s_r)) \\ &\wedge (Sim_s(s, s_r) > 0) \wedge (s \neq s_r)\}, \end{aligned} \tag{5.19}$$

- Set of most similar service provider loads:

$$\begin{aligned} S_l(l_r) = &\{l \in L | \exists D[l, h_r] \wedge (Sim_l(l, l_r) \geq Sim_l(l_k, l_r)) \\ &\wedge (Sim_l(l, l_r) > 0) \wedge (l \neq l_r)\}, \end{aligned} \tag{5.20}$$

- Set of most similar service classes:

$$\begin{aligned} S_c(c_r) = &\{c \in C | \exists D[c, h_r] \wedge (Sim_c(c, c_r) \geq Sim_c(c_k, c_r)) \\ &\wedge (Sim_c(c, c_r) > 0) \wedge (c \neq c_r)\}. \end{aligned} \tag{5.21}$$

The generic variable h_r in the Equations 5.18 – 5.21 presents the set of triples defined in the Equations 5.10, 5.12, 5.14 and 5.16 for each set respectively.

5.2.5 Calculating the Expected Reliability

In the final step of the *LUCS* prediction model (recall Figure 5.1), the missing reliability value for a given service invocation r is calculated. The reliability is calculated considering impacts of the four *LUCS* parameters. The generic equation that can be adopted to calculate the impact of different *LUCS* parameters is provided. This equation uses the results obtained from the Equations 5.9, 5.11, 5.13 and 5.15 which capture similarity computation, and the Equations 5.18 – 5.21 which compute the most similar entities. According to the generic equation, the expected reliability p_{e_r} of the current invocation is calculated considering similar entities data as follows:

$$p_{e_r} = \begin{cases} \sum\limits_{a \in S_e(e_r)} \omega_a \times D[a, h_r] \times \dfrac{\overline{D[e_r]}}{\overline{D[a]}}, & S_e(e_r) \neq \emptyset \\ \overline{D[e_r]}, & S_e(e_r) = \emptyset \end{cases} \tag{5.22}$$

where

$$\omega_a = \frac{Sim_e(e_r, a)}{\sum\limits_{b \in S_e(e_r)} Sim_e(e_r, b)} \tag{5.23}$$

In the Equation 5.22, $\overline{D[e_r]}$ and $\overline{D[a]}$ are the average reliabilities for the entities e_r and a collected on the whole set of triples H, while ω_a is the weight factor that enables more similar entities with higher $Sim_e(e_r, a)$ values to contribute more to the prediction. The generic variable h_r presents the set of triples defined in Equations 5.10, 5.12, 5.14 and 5.16 for each *LUCS* parameter respectively. In case the similar set of entities is empty, the average reliability $\overline{D[e_r]}$ is used.

By substituting the actual variables u, s, l, c instead of generic variable e into Equations 5.22 and 5.23, four different contributions to the reliability prediction according to the respected *LUCS* parameters are obtained:

- prediction based on similar user locations:

$$p_{u_r} = \begin{cases} \sum\limits_{a \in S_u(u_r)} \omega_a \times D[a, h_r] \times \dfrac{\overline{D[u_r]}}{\overline{D[a]}}, & S_u(u_r) \neq \emptyset \\ \overline{D[u_r]}, & S_u(u_r) = \emptyset \end{cases} \tag{5.24}$$

where

$$\omega_a = \frac{Sim_u(u_r, a)}{\sum_{b \in S_u(u_r)} Sim_u(u_r, b)}, \tag{5.25}$$

- prediction based on similar service locations:

$$p_{s_r} = \begin{cases} \sum_{a \in S_s(s_r)} \omega_a \times D[a, h_r] \times \frac{\overline{D[s_r]}}{\overline{D[a]}}, & S_s(s_r) \neq \emptyset \\ \overline{D[s_r]}, & S_s(s_r) = \emptyset \end{cases} \tag{5.26}$$

where

$$\omega_a = \frac{Sim_s(s_r, a)}{\sum_{b \in S_s(s_r)} Sim_s(s_r, b)}, \tag{5.27}$$

- prediction based on similar service loads:

$$p_{l_r} = \begin{cases} \sum_{a \in S_l(l_r)} \omega_a \times D[a, h_r] \times \frac{\overline{D[l_r]}}{\overline{D[a]}}, & S_l(l_r) \neq \emptyset \\ \overline{D[l_r]}, & S_l(l_r) = \emptyset \end{cases} \tag{5.28}$$

where

$$\omega_a = \frac{Sim_l(l_r, a)}{\sum_{b \in S_l(l_r)} Sim_l(l_r, b)}, \tag{5.29}$$

- prediction based on similar service classes:

$$p_{c_r} = \begin{cases} \sum_{a \in S_c(c_r)} \omega_a \times D[a, h_r] \times \frac{\overline{D[c_r]}}{\overline{D[a]}}, & S_c(c_r) \neq \emptyset \\ \overline{D[c_r]}, & S_c(c_r) = \emptyset \end{cases} \tag{5.30}$$

where

$$\omega_a = \frac{Sim_c(c_r, a)}{\sum_{b \in S_c(c_r)} Sim_c(c_r, b)} \tag{5.31}$$

In order to make the prediction more accurate, all four prediction impacts defined in the Equations 5.24, 5.26, 5.28 and 5.30 are incorporated in the final prediction equation. Hence, to combine the individual reliability prediction impacts, their linear combination is used as

proposed:

$$p = \alpha \times p_{u_r} + \beta \times p_{s_r} + \gamma \times p_{l_r} + (1 - (\alpha + \beta + \gamma) \times p_{c_r}) \tag{5.32}$$

where:

$$\alpha, \beta, \gamma \in [0, 1] \wedge \alpha + \beta + \gamma \leq 1. \tag{5.33}$$

The above Equation 5.32 assigns weights α, β, and γ for different prediction impacts that correspond to the respected *LUCS* dimensions. These are model parameters which may have different values for specific user and service locations, service loads and classes. In the evaluation chapter (see Chapter 7), the heuristics for adjustment of the parameter's values to a specific environment is presented.

Chapter 6

CLUS - Model for Prediction of Application's Reliability Based on K-means Clustering

In this chapter, *CLUS* [25], a model for service reliability prediction is presented. Section 6.1 provides a brief overview of the model, introduces the model parameters and describes the reliability prediction process. Section 6.2 provides a detailed mathematical description of the model.

6.1 *CLUS* Overview

In this section, the overview of the *CLUS*, model for prediction of atomic web services reliability is provided. With the aim to improve the prediction accuracy and scalability, user–, service– and environment specific parameters are defined to determine service invocation context in greater detail then the related prediction models (described in Chapter 4). The collected history invocation data is grouped across three different dimensions associated with the defined model parameters. The rest of the section is organized as follows. Section 6.1.1 describes the parameters that determine service invocation context, while Section 6.1.2 presents the reliability prediction process required in *CLUS*.

6.1.1 Invocation Context Parameters in *CLUS*

In CLUS model three different groups of parameters that impact the reliability performance of the service are distinguished: user–, service– and environment– specific parameters.

User-specific parameters

User-specific parameters are associated with user-introduced fluctuations in the service reliability performance. Those parameters include a variety of factors that might impact the reliability of s service such as user's location, network and device capabilities, usage profiles. To incorporate user-specific parameters in the process of prediction, users are grouped into clusters according to their reliability performance gained from the past invocation sample using K-means clustering algorithm.

Service-specific parameters

Service-specific parameters are related with the impact of service characteristics on the reliability performance. Numerous factors influence service-specific parameters such as service's location, computational complexity and system resources (e.g. CPU, RAM, disk and I/O operations). The service-specific parameters are included in the prediction process by grouping services into clusters according to their reliability performance obtained from the past invocation sample using K-means clustering algorithm.

Environment-specific parameters

Environment-specific parameters relate to the current conditions in the environment such as service provider load or network performance at the time of the invocation. For the purposes of evaluation, only service load is considered as the environment parameter. Service load is defined as the number of requests received per second. The nonfunctional qualities of a service, such as availability and reliability are significantly influenced by fluctuations in the service load. Since web servers register considerable load variations during the day [67], the day is divided into an arbitrary number of *time windows*. In order to improve the prediction accuracy the past invocation data is dispersed among different time windows. Finally, time windows are grouped into clusters according to the reliability performance computed from the past invocation sample using K-means clustering algorithm.

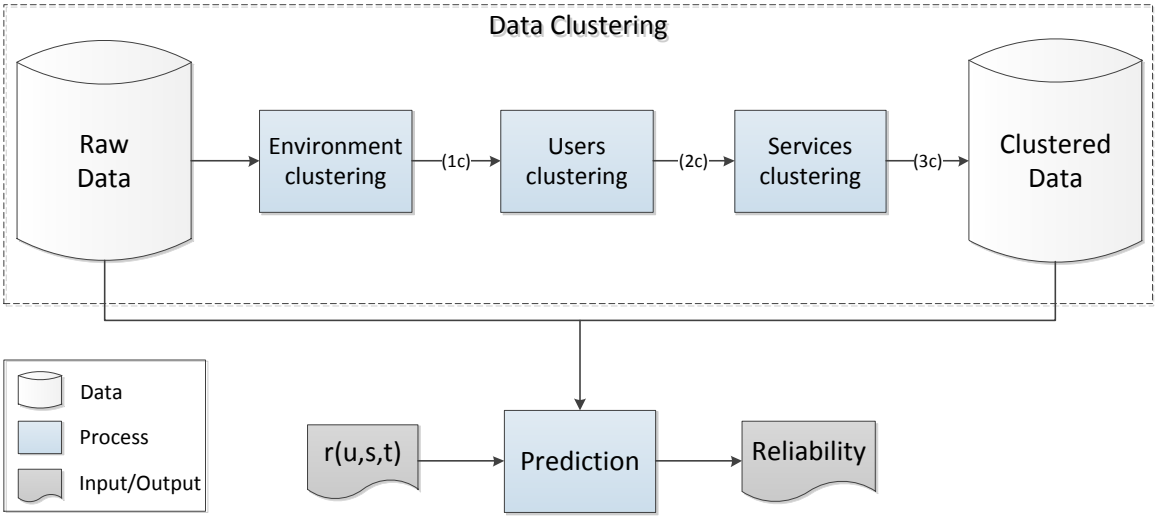


Figure 6.1: CLUS reliability prediction overview

6.1.2 Reliability Prediction Process

The high level overview of *CLUS*, model for prediction of atomic web services is depicted in Figure 6.1. Prior to reliability prediction, the clustering of the history invocation data is performed. First, time windows associated with the environment conditions are clustered according to the reliability performance fetched from the past invocation sample. Then, users (2c) and services (3c) are clustered considering their reliability performance for each time window cluster. Once the data is clustered, the prediction of service reliability can be performed.

6.2 Formal Definition of *CLUS*

In this section, *CLUS*, the model for web services reliability prediction is presented in formal description. Each step of data clustering process that is crucial for the reliability prediction is separately described.

Let the service invocation be formally defined as:

$$r(u, s, t). \tag{6.1}$$

In the presented Equation 6.1, u is the user executing the invocation, s is the service to be invoked and t is the actual time of the service invocation.

The history invocation sample contains data addressed as in the Equation 6.1. In order to

make scalable and accurate reliability predictions for future service invocations, the data needs to be transformed into a more structured and compact form. The idea is to store the data into a three-dimensional space $D[u, s, e]$ according to the defined groups of parameters. Each dimension u, s and e in space D is associated with one group of parameters respectively. The following sections describe how particular records are clustered and associated with environment–, user– and service–specific parameters. Finally, the creation of space D is described, i.e. how each entry in D is calculated and how the reliability is predicted for an ongoing service invocation.

6.2.1 Environment-specific Data Clustering

First, the set of different environment conditions E is defined as follows:

$$E = \{e_1, e_2, ..., e_i, ..., e_n\}, \tag{6.2}$$

where e_i refers to a specific environment condition determined by service provider load and n is an arbitrary number of distinct environment conditions.

The aim is to correlate each available history invocation record with the service provider load at the time it was performed. As already stated in Sections 3.4.1 and 4.6.3, the analyses of the collected data from different service providers can pinpoint the regularities in the load distribution for certain time periods [65–67]. Thus, the day is divided into an arbitrary number of time windows, where each time window w_i is determined with its start time t_i and end time t_{i+1}. This is performed under the assumption that the environment-specific parameters are stable during the same time window. Once the time windows are determined, the average reliability value $\overline{p_{w_i}}$ for each time window w_i is calculated as follows:

$$\overline{p_{w_i}} = \frac{1}{|W_i|} \sum_{r \in W_i} p_r \tag{6.3}$$

where W_i is the set of records within the time window w_i, r is the record from the past invocation sample and p_r is user perceived reliability for that invocation.

The average reliability value $\overline{p_{w_i}}$ is assigned to each respected time window w_i. Each particular time window is clustered using K-means clustering algorithm into an appropriate environment condition e_i according to its average reliability value. Now each particular record from the

history invocation sample can be correlated with the environment conditions at the time it was performed. By applying a transitive relation, if the record about previous invocation $r(u, s, t)$ belongs to the time window w_i and the time window w_i is clustered into environment condition e_i, than the invocation $r(u, s, t)$ is performed during the environment condition e_i.

6.2.2 User-specific Data Clustering

Next, the set of different user groups is defined U as follows:

$$U = \{u_1, u_2, ..., u_i, ..., u_m\}, \tag{6.4}$$

where each user group u_i contains users that achieve similar reliability performance, while m is the number of different user groups.

For each user u in the past invocation sample, the n-dimensional reliability vector $\boldsymbol{p_u}$ is calculated as follows:

$$\boldsymbol{p_u} = \{\overline{p_{e_1}}, \overline{p_{e_2}}, ..., \overline{p_{e_i}}, ..., \overline{p_{e_n}}\}, \tag{6.5}$$

where each vector dimension $\overline{p_{e_i}}$ represents the average reliability value perceived by the given user u during the environment conditions e_i. Once the the average reliability n-dimensional vector is calculated and assigned to each user, K-means clustering is performed to group users into different user groups according to their reliability vector's $\boldsymbol{p_u}$ values. Now each available previous invocation record $r(u, s, t)$ can be easily correlated with an appropriate user group u_i.

6.2.3 Service-specific Data Clustering

Finally, the set of distinct service groups S is defined as follows:

$$S = \{s_1, s_2, ..., s_i, ..., s_l\}, \tag{6.6}$$

where each service group s contains services that achieve similar reliability performance, while l is an arbitrary number of different service groups.

For each service s in the past invocation sample, the n-dimensional reliability vector $\boldsymbol{p_s}$ is calculated as follows:

$$\boldsymbol{p_s} = \{\overline{p_{e_1}}, \overline{p_{e_2}}, ..., \overline{p_{e_i}}, ..., \overline{p_{e_n}}\}, \tag{6.7}$$

where each vector dimension $\overline{p_{e_i}}$ represents the achieved average reliability for invoking service s during the environment conditions e_i. Once the the average reliability n-dimensional vector is calculated for each service, K-means clustering is performed to group services into different service groups according to their reliability vector's $\boldsymbol{p_s}$ values. Now each available previous invocation record $r(u, s, t)$ can be easily associated with appropriate service group s_i.

6.2.4 Creation of Space D and Prediction

After the steps which are described in previous sections are completed, each available history invocation record $r(u, s, t)$ is associated with the respected data clusters e_i, u_k and s_j. The next step in the reliability prediction process is to create the space D and calculate each value in D. Each entry of D is calculated as follows:

$$D[u_k, s_j, e_i] = \frac{1}{|R|} \sum_{r \in R} p_r. \tag{6.8}$$

where p_r is user perceived reliability for invocation r and:

$$R = \{r(u, s, t) | r \in u_k \wedge r \in s_j \wedge r \in e_i\} \tag{6.9}$$

Now, let there is an ongoing service invocation $r_c(u_c, s_c, t_c)$ whose reliability p_c needs to be predicted. First, the past invocation sample is checked for the set H containing records with the same invocation context parameters as r_c:

$$H = \{r_h | u_h = u_c \wedge s_h = s_c \wedge t_h, t_c \in w_i\}. \tag{6.10}$$

If the set H is not empty, than the reliability p_c is calculated by using the existing reliability values in the set H:

$$p_c = \frac{1}{|H|} \sum_{r \in H} p_r. \tag{6.11}$$

Otherwise, if the set H is empty, the reliability p_c is calculated using the data stored in the space D as follows:

$$p_c = D[u_k, s_j, e_i], \tag{6.12}$$

where current user u_c belongs to the user group u_k, current service s_c belongs to the service group s_j and the actual time t_c is associated with the environment conditions e_i.

Chapter 7

Evaluation

In this chapter, the proposed models *CLUS* and *LUCS*, are evaluated and compared to the existing collaborative filtering based approaches primary regarding prediction accuracy and performance. Also, several quality aspects of the *CLUS* and *LUCS* models are studied.

In the evaluation, the proposed models are compared to three state-of-the-art approaches: *user-based* approach *(UPCC)* [14], *item-based* approach *(IPCC)* [15] and the *Hybrid* approach [16], [17], which is a linear combination of *IPCC* and *UPCC*. User-based approach employs the impact of similar users, while item-based approach employs exclusively the impact of similar services. By contrast, the *Hybrid* approach utilizes both the impact of similar users and services for reliability prediction. Presented evaluation results suggest that proposed approaches are significantly more accurate than the existing collaborative filtering based techniques. Regarding prediction performance, evaluation results confirm that proposed approaches obtain better scalability and have greater potential to produce real time reliability predictions.

To compare the prediction accuracy of competing approaches, commonly used measures *Mean Absolute Error (MAE)* and *Root Mean Square Error (RMSE)* are applied. The *MAE* calculates the average magnitude of the errors in a prediction set. It is defined as follows:

$$MAE = \frac{\sum_{j=1}^{N} |p_j - \hat{p_j}|}{N} \tag{7.1}$$

In the Equation 7.1, p_j is the measured reliability value calculated from the data collected in experiment, $\hat{p_j}$ is the predicted reliability value, while N is the number of predicted values. The *RMSE* is a quadratic scoring rule which measures the average magnitude of the error and is

defined as:

$$RMSE = \sqrt{\frac{\sum_{j=1}^{N}(p_j - \hat{p_j})^2}{N}} \tag{7.2}$$

Both *MAE* and *RMSE* can range from 0 to ∞. They are negatively-oriented scores, which means that lower values are better.

To compare the prediction performance of competing approaches, *aggregated time* that takes to produce the predictions for the entire prediction set is measured. In addition, the analytical proof containing the analysis of complexity for each considered approach is provided to support claims in favor of better *CLUS*'s and *LUCS*'s scalability.

The rest of the chapter is organized as follows. Section 7.1 describes the experiment setup. Section 7.2 compares the predicted and measured reliability values with a special focus on the areas where the accuracy of proposed approaches notably differs from the others. Section 7.3 analyzes the impact of the collected data density on the prediction accuracy and performance. Section 7.4 analyzes the impact of different loads and service classes on the accuracy of prediction for each model. Section 7.5 analyzes the influence of the explicit availability of *LUCS*'s input parameters on prediction accuracy. Section 7.6 assesses the sensitivity of the *LUCS* model on the homogeneity inside the parameter classes. Section 7.7 provides a heuristic for fine-tuning the values of *LUCS*'s model parameters (α, β and γ in the Equation 5.32). Section 7.8 reveals how the impact of clusters number in *CLUS* model affects its prediction accuracy and performance. Section 7.9 analyzes the theoretical worst case complexity as well as the expected practical complexity for different prediction approaches. Section 7.10 provides a brief overview of the entire evaluation chapter summarizing all evaluation results together.

7.1 Experiment Setup

In order to perform experiments that test different aspects of proposed models, a controlled environment comprising a set of web services was constructed. The implemented web services create two random matrices and execute a matrix multiplication operation. By doing so, the external noise was limited, while full control on the services computations, locations, and loads was preserved.

The computational complexity of a web service can depend on the variety of specific parameters such as CPU demands, the amount of memory, and the amount of disk or I/O operations.

Service class	1	2	3	4	5	6	7
Matrix rank	350	310	280	250	210	180	150

Table 7.1: Matrix ranks in different service classes

Load level	1	2	3	4	5	6	7
Time interval / sec	3	4	5	6	7	8	9

Table 7.2: Time intervals in different load levels

For technical reasons, in this evaluation the amount of memory to group services into classes was chosen. Note, however, that this does not reduce the generality of obtained results as other factors that impact reliability may be considered as well. Seven different computational classes of the matrix multiplication web services were created. Each class requires a different amount of memory for its execution as depicted in Table 7.1. All of the service classes have the same computational complexity $O(n^3)$.

To test the location-related parameters of the proposed approaches, the *Amazon Web Services* [204] technology as the platform for deployment of web services was used. The amount of 49 web services were deployed in seven available *Amazon EC2 (Elastic Compute Cloud)* geographic regions: *Ireland*, *Virginia*, *Oregon*, *California*, *Singapore*, *Japan* and *Brazil*, having each service class in each region. Each service was deployed on an Amazon machine image running *Microsoft Windows Server 2008 R2 SP1 Datacenter edition*, 64-bit architecture, *Microsoft SQLServer*, *Internet Information Services 7* and *ASP.NET 3.5*.

In a similar manner, users coming from different locations were simulated by placing instances of *loadUI* [205], an open source tool intended for stress and performance testing, on different cloud locations. Two different types of environments regarding network connection capabilities were simulated during the experiment. The first simulated environment is a typical high bandwidth Internet connection where users obtain very similar network capabilities. The second simulated environment is a more realistic environment where users network capabilities differ due to provider's network infrastructure or physical constraints of the access device. This environment was achieved in the experiment by using special tools for limiting the performance of the network adapter.

The *loadUI* tool enables creation of various test cases including different load generators. In the experiment, the services were burdened with seven different load levels defined by the *time interval* between sending each request as depicted in Table 7.2. The maximum load level,

Load 1, uses the time interval of *3 sec*, while each higher load level increases the time interval by *1 sec*, and thus the minimum load level, *Load 7*, uses the time interval of *9 sec*. For each load level, a special test case was defined. Each defined test case was delivered as a task to all distributed agents running in the cloud. During the test case each agent sent 150 requests to each deployed web service. When the test case was done, the data from the agents was collected and the machines hosting services were restarted to fully recover for the next text case. Each test case was done for both considered environment for the purposes of proposed approaches evaluation. Overall, as part of experiments, the information about $5.145,000$ distinct web service invocations was collected.

The final reliability prediction in *LUCS* model is calculated using the Equation 5.32. It should be noted that an equal value of 0.25 for each model parameter α, β and γ was used during the model evaluation in experiments.

7.2 Overall Prediction Accuracy

Model	LUCS	CLUS	Hybrid	IPCC	UPCC
RMSE	0.01697	0.04632	0.09372	0.09082	0.10199
MAE	0.00810	0.02615	0.07287	0.07040	0.07974

Table 7.3: MAE and RMSE values for each approach for the density of 25% in the environment where users have similar network capabilities.

To evaluate the overall quality of the *CLUS* and *LUCS* prediction models in comparison with the other state-of-art models, *MAE* and *RMSE* values for their predicted reliabilities are analyzed. This calculation is done for each model over the complete set of data collected and measured during the experiments. The *MAE* and *RMSE* value depend of the amount of the available history data in D, and both of the distribution of the available data in D. The impact of the amount of the available data (see details in Section 7.3) is separately analyzed. For the purpose of overall prediction accuracy analysis, the amount of available history data is assumed to be 25% of all the data in D. In order to minimize the impact of the potentially biased distribution of available data, the calculation is repeated $1k$ times having different randomly generated distributions of available data each time. Finally, average *MAE* and *RMSE* values for each particular approach were calculated.

Table 7.3 shows obtained average *MAE* and *RMSE* values for every considered approach

for the environment where users have similar network capabilities. The collected results clearly indicate that both *LUCS* and *CLUS* outperform the other analyzed approaches. The *LUCS* approach achieves best prediction accuracy among considered approaches with at least 81% lower *RMSE* value (for the *Hybrid* model) and at least 88% lower *MAE* value, while *CLUS* model achieves second best prediction accuracy among analyzed approaches with at least 48% lower *RMSE* value (for the *Hybrid* model) and at least 62% lower *MAE* value. In terms of the absolute values, the *LUCS* predictions are typically off by only a couple of percentage points. Such inaccuracy is significantly more acceptable during application selection than several times higher errors of other, less scalable, approaches. In particular, application selection involves comparing the quality of candidate application of similar functionalities. Hence, each decrease in the prediction accuracy may be increased by a factor of two during application selection.

Table 7.4 shows obtained average *MAE* and *RMSE* values for every considered approach for the environment where users have different network capabilities. The evaluation results show that *CLUS* approach provides best prediction performance regarding prediction accuracy. The *CLUS* approach achieves better prediction accuracy in this environment than *LUCS* approach due to implicit consideration of user specific parameters by performing clustering of users according to their performance. The *LUCS* approach considers only user's location as the user specific parameter which is obviously not sufficient to provide best prediction accuracy. Both *LUCS* and *CLUS* approach significantly outperform the existing state-of-the-art approaches. The *CLUS* approach achieves at least 44% lower *RMSE* value and 57% lower *MAE* value compared to the *Hybrid* approach while *LUCS* approach achieves at least 37% lower *RMSE* value and 45% lower *MAE* value than the *Hybrid* approach. The importance of user specific parameters in this environment is manifested in fact that *UPCC* approach improves its prediction accuracy when compared to the prediction performance in the environment where users have similar network capabilities.

Model	LUCS	CLUS	Hybrid	IPCC	UPCC
RMSE	0.05830	0.05185	0.09405	0.09194	0.10129
MAE	0.03935	0.03116	0.07280	0.07117	0.07862

Table 7.4: MAE and RMSE values for each approach for the density of 25% in the environment where users have different network capabilities.

To explore the primary causes of variance between the different prediction models, the four-dimensional space D is analyzed. This analysis confirms the underlying motivation for the

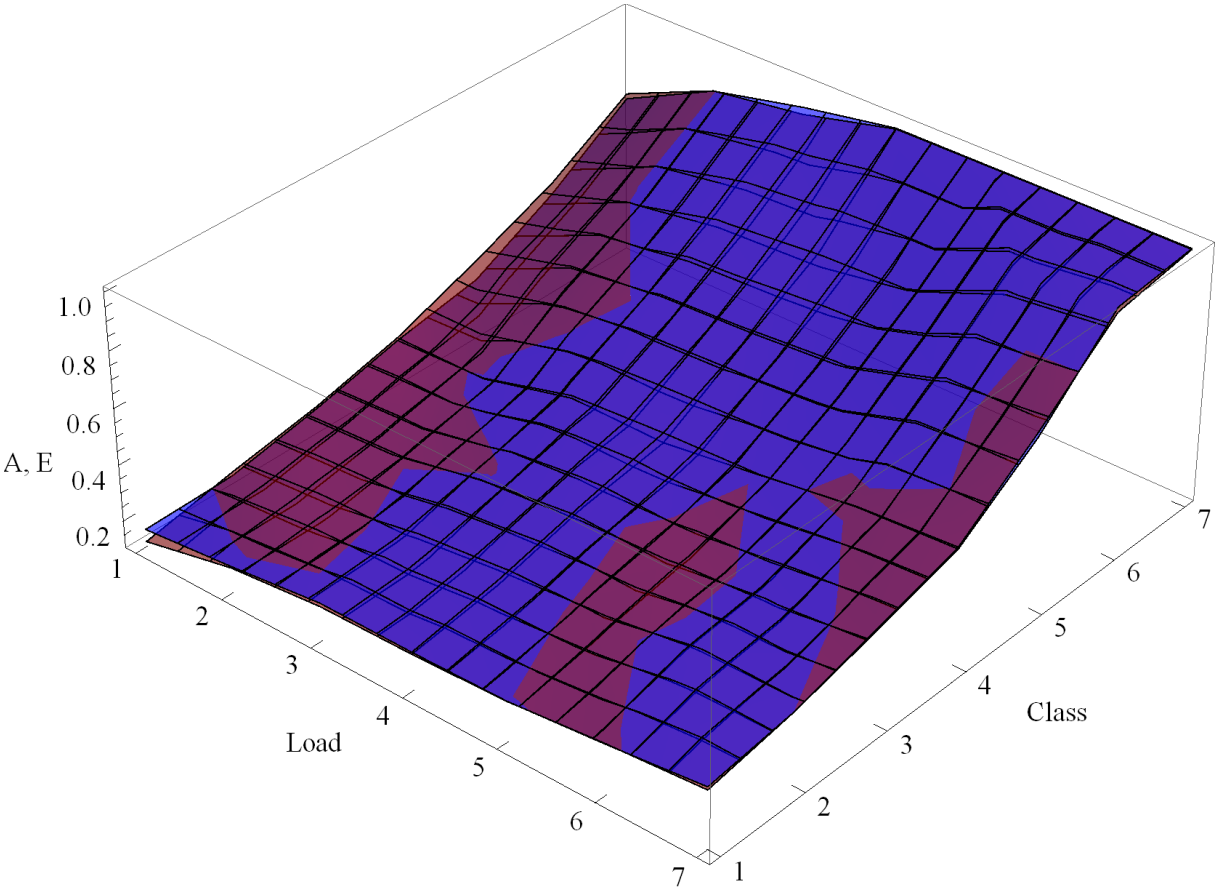


Figure 7.1: LUCS, predicted and measured reliability, users location: Ireland, services location: Brazil

additional *LUCS* and *CLUS* dimensions and parameters. These findings are illustrated using the four graphs depicted in Figures 7.1–7.5 that show the predicted reliability values (A) alongside the measured reliability values (E) for each of the four evaluated approaches for the environment with users that have similar network properties having available data density 25% of all measured data. The area shown in these figures includes all points from D where the user location is *Ireland* and the service location is *Brazil*, while the values for service load and service class are on the axes. Each figure shows two transparent plots where the blue plot presents the measured values and the other plot presents values predicted by model. Note that the plots obtained for other combinations of user locations and service locations are qualitatively similar.

The Figures 7.1–7.5 reinforce the overall results that the *LUCS* predictions provide the best accuracy among the competing approaches. Furthermore, the Figures 7.3–7.5 discover the primary source of the inaccuracies of the *Hybrid*, *UPCC* and *IPCC* prediction models — these models do not consider environment conditions such as service load as a parameter. As a consequence, the reliability prediction highly differs from the expected values in those areas where

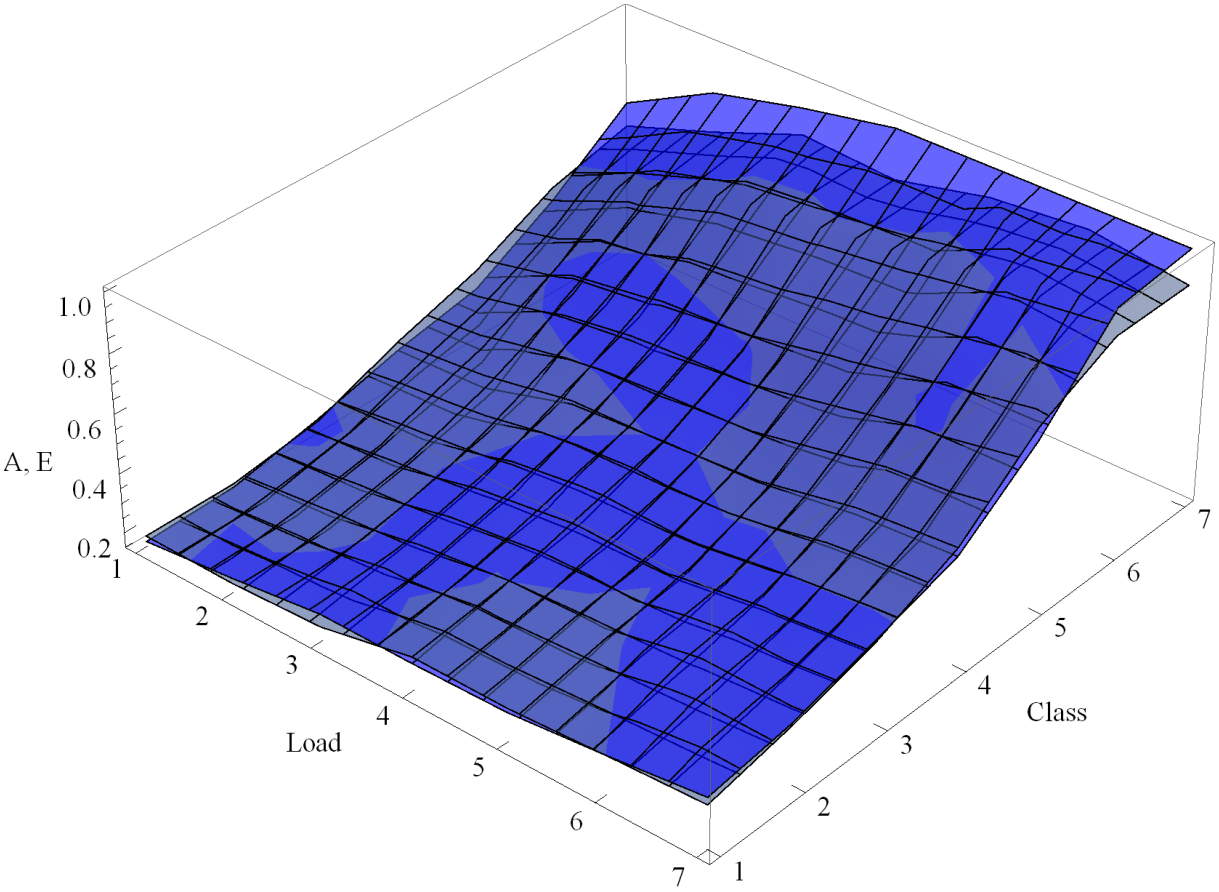


Figure 7.2: CLUS, predicted and measured reliability, users location: Ireland, services location: Brazil

the service load differs from the average values. By contrast, in the areas where the service load is average, the *Hybrid*, *UPCC* and *IPCC* prediction models provide predictions that are similar to the *LUCS* and *CLUS* predictions and notably closer to the measured reliability values.

7.3 Impact of Data Density

The overall accuracy results presented in the previous section suggest *LUCS* and *CLUS* models as a better solutions than the alternative options. However, these conclusions were made for a specific experiment instantiation with densely populated reliability data matrix D. To assess how lower amount of available data affects the different prediction models, various data densities between 5% and 50% with a step value of 5% are simulated. Section 7.3.1 analyses the impact of data density on prediction accuracy while Section 7.3.2 studies how data density effects computational performance of the prediction for every considered approach.

In this evaluation, three distinct cases related to the variations of environment conditions are

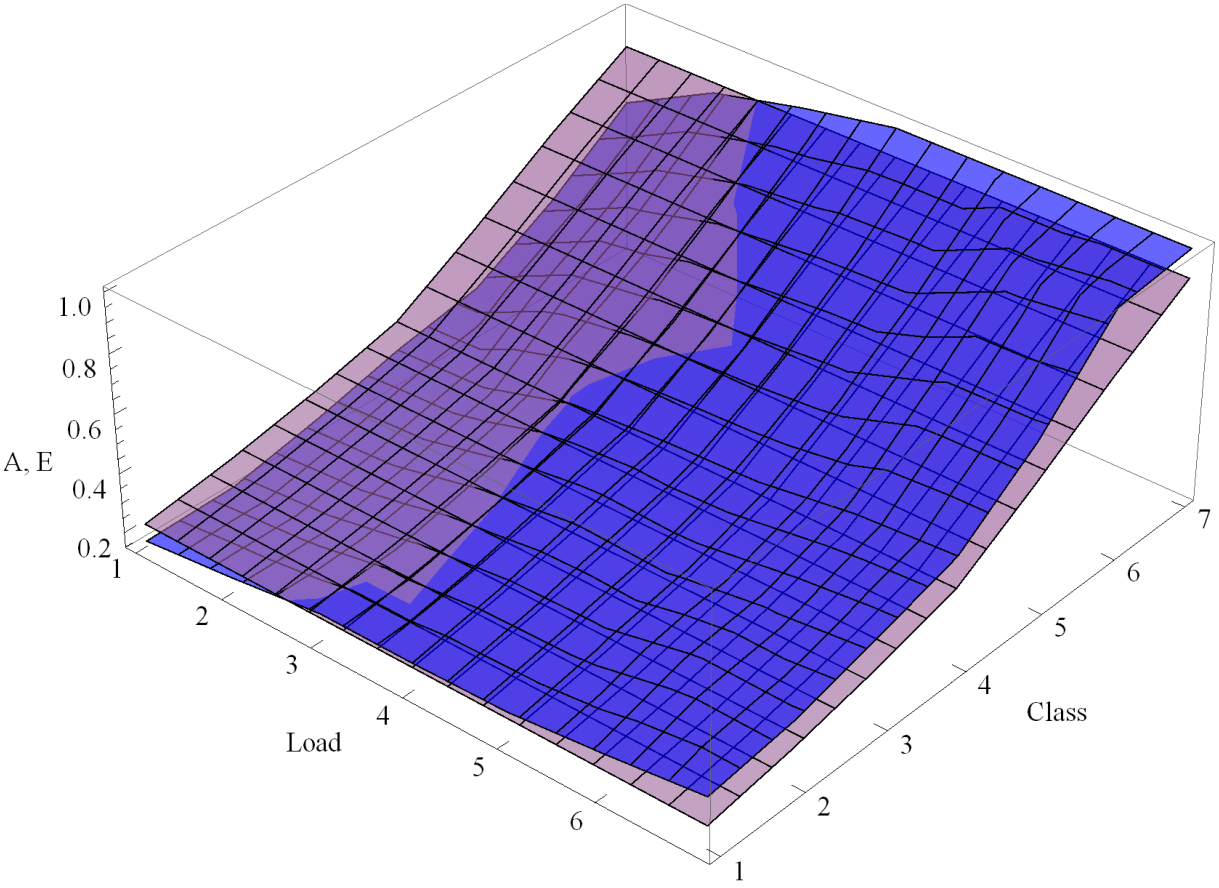


Figure 7.3: Hybrid, predicted and measured reliability, users location: Ireland, services location: Brazil

considered. In the first case, variable loads with request frequencies between *req/3 sec*, *req/4 sec*, *req/5 sec*, *req/6 sec*, *req/7 sec*, *req/8 sec*, and *req/9 sec* and users with very similar network capabilities are assumed. In the second case, variable loads with request frequencies like in the first case are considered, but users are distinguished having different network capabilities. The third case considers users having similar network capabilities in the environment with constant service load with request frequency of *req/3 sec*.

To evaluate the accuracy and performance of the prediction, for all cases, the testing sets containing all measured reliability records are determined. Once the testing set is determined, the amount of 5% of the collected data is included and put it into the set of data that is used to calculate the reliability prediction. Then, reliability prediction for each approach for the testing set using the available data is calculated. Since the actual reliability values have been measured during the experiments, the *MAE* and *RMSE* values are calculated for each approach. In addition, the time that takes to compute the predictions for all approaches is measured. In the next step, data density is increased by adding another 5% of the collected data into the

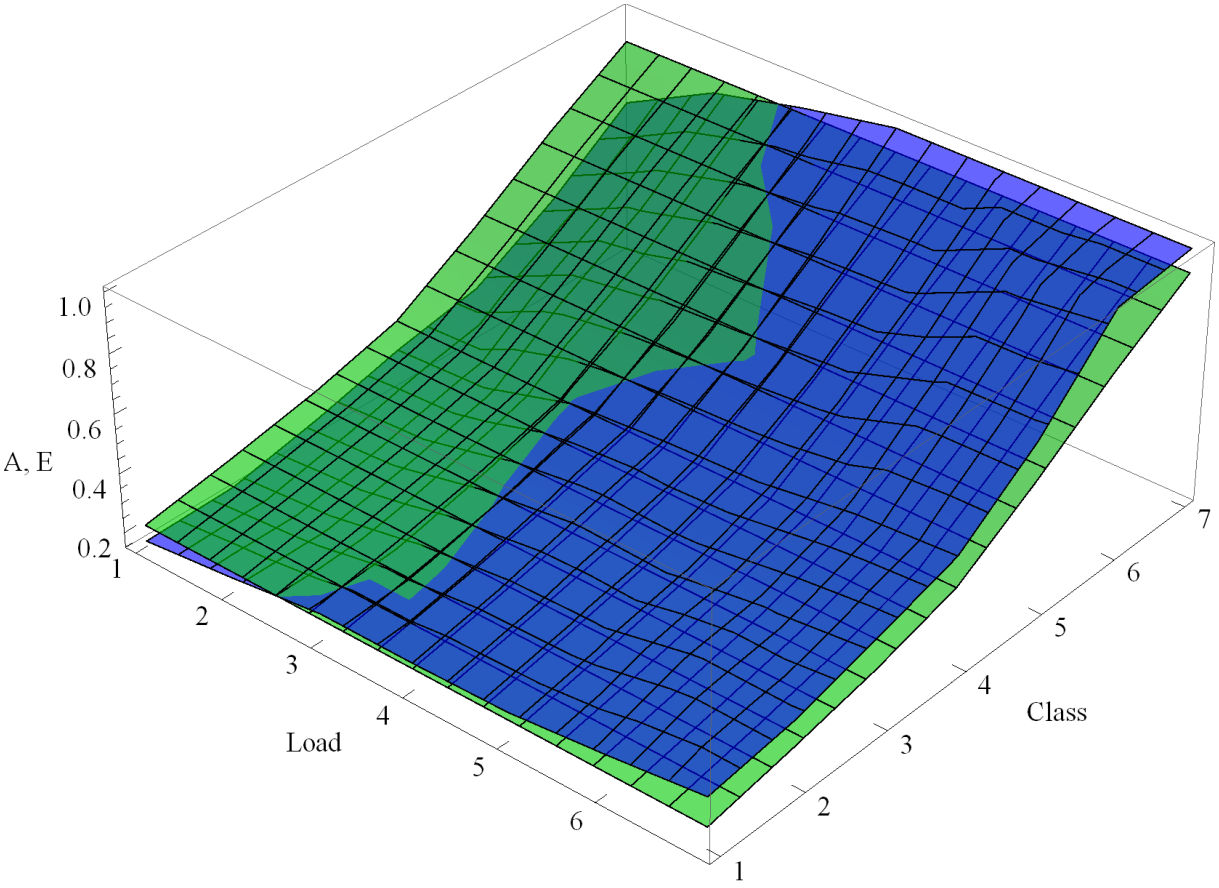


Figure 7.4: IPCC, predicted and measured reliability, users location: Ireland, services location: Brazil

set of available data and the reliability predictions and predictions performance measures are recalculated. This process is repeated until the data density reaches 50%.

7.3.1 Prediction Accuracy

The Figures 7.6a and 7.6b capture the relation between the data density and the *MAE* and *RMSE* values for the case with varying service loads and having users with similar network capabilities. These results show that the data density highly impacts the prediction accuracy for all of the analyzed approaches (e.g. for the *IPCC* approach the *RMSE* value varies between 0.274 and 0.090). Furthermore, the figures confirm that *LUCS* has a significantly better prediction accuracy with near-constant margin compared to the *CLUS* and *Hybrid* model. As expected, the *LUCS* approach improves prediction accuracy as the data density increases, lowering the *RMSE* and *MAE* values from 0.038 to 0.012 and 0.024 to 0.004 respectively. It is also important to note that the prediction accuracy of *LUCS* even for data density of 5%, with the *RMSE* value of 0.038, is significantly better or similar to (in the case of *CLUS*) than the accuracy reached by

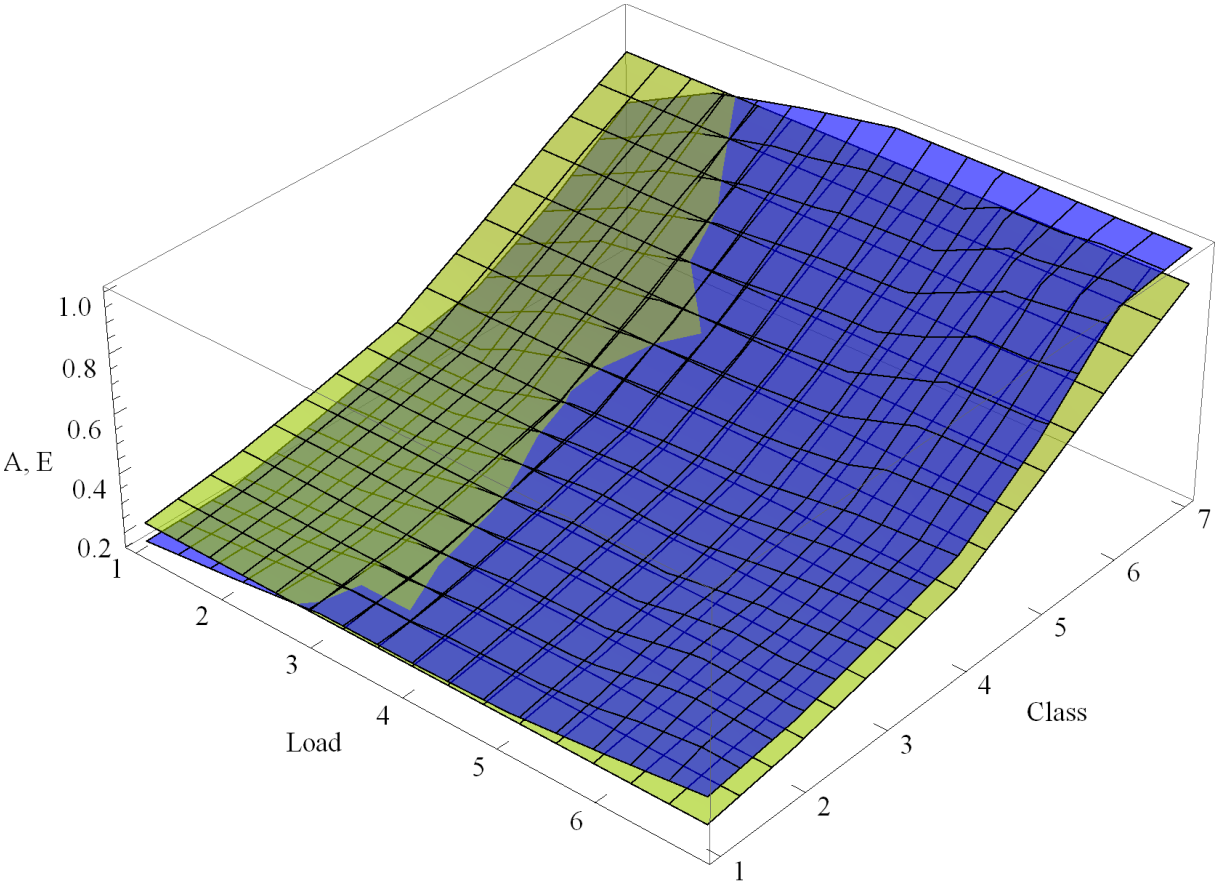
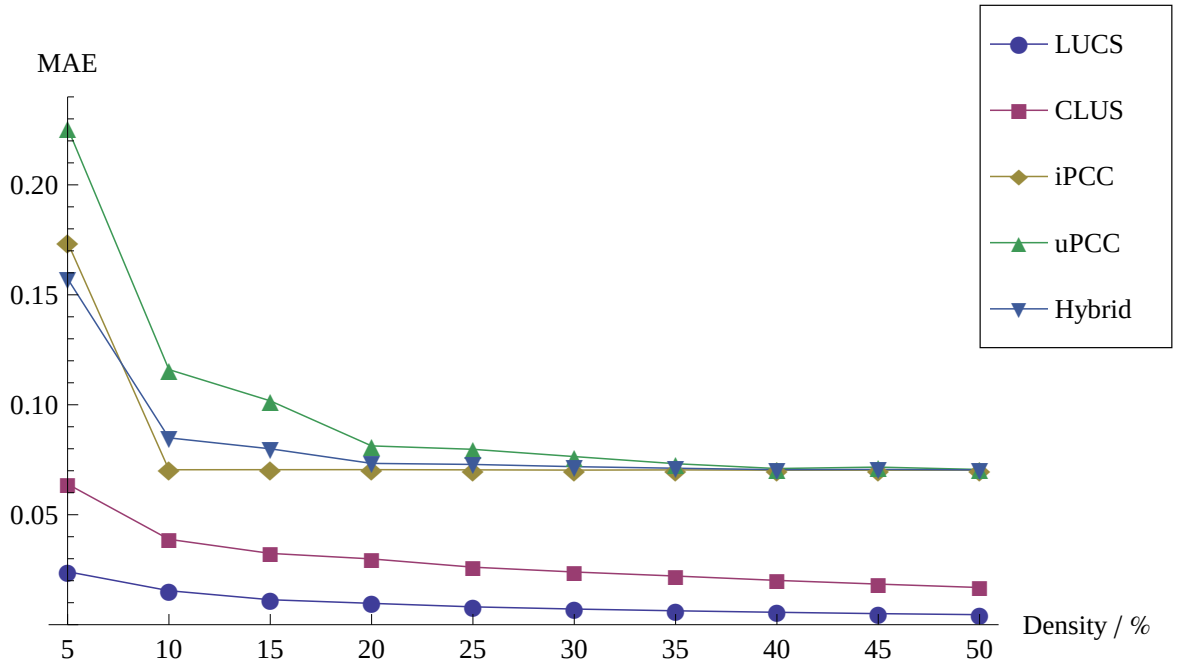


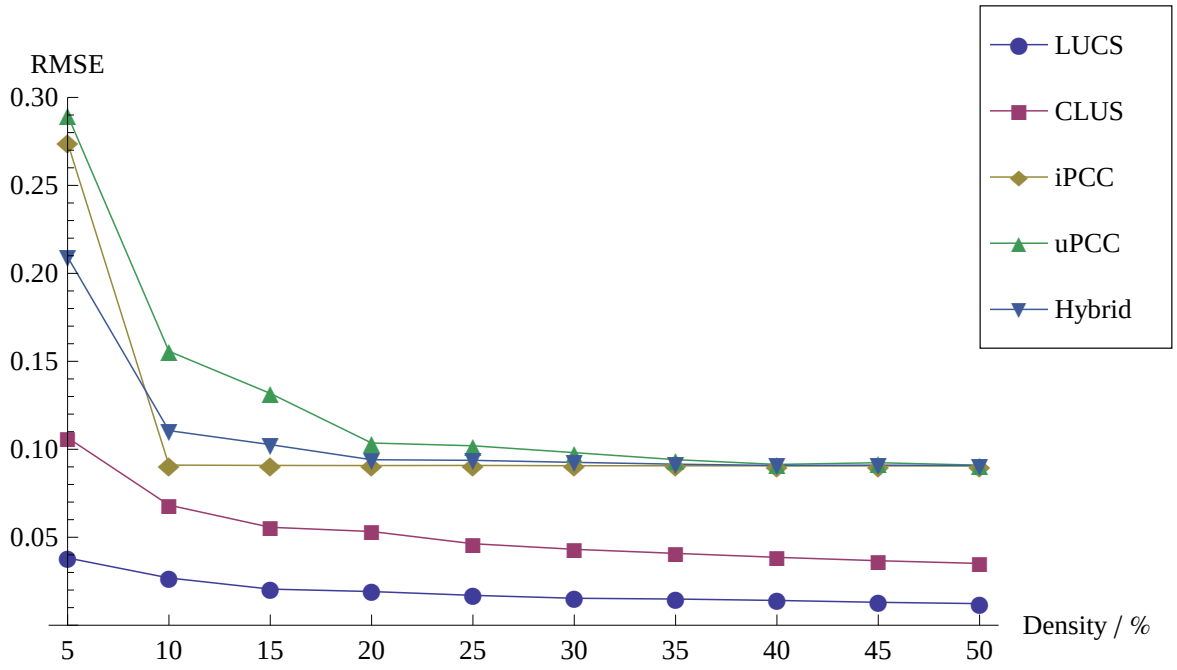
Figure 7.5: UPCC, predicted and measured reliability, users location: Ireland, services location: Brazil

the other approaches at 50% of data density. The second proposed approach *CLUS*, provides better prediction accuracy than the existing collaborative filtering based approaches (the *MAE* value from 0.063 to 0.016 and the *RMSE* from 0.106 to 0.035).

The differences in the accuracy of the *IPCC*, *UPCC* and *Hybrid* models stem from the structure of experimental conditions. In particular, the evaluation was performed using *50* distributed agents and *49* different applications. The higher number of services implies less information per individual application, which affects *IPCC* because its predictions are based on service-specific parameters. In the environment where users have similar network capabilities, the service-specific parameters make the difference in the perceived reliability. Hence, having different service classes like in the experiments, the *IPCC* approach is expected to provide better predictions than *UPCC*. For the lower data densities, the *Hybrid* expectedly outperforms both *IPCC* and *UPCC*, but with relatively small margin. It should be noted that all collaborative filtering based approaches achieve similar prediction accuracy that cannot be considerably improved with the increase of the collected data due to negligence of environment's dynamic nature and



(a) MAE



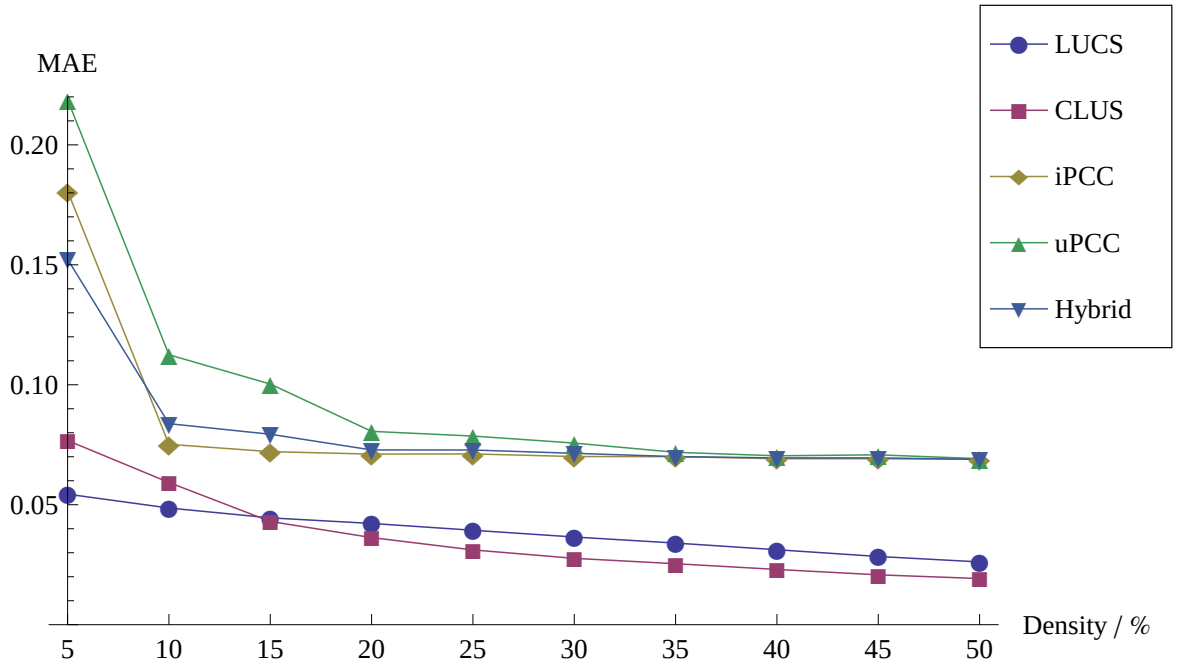
(b) RMSE

Figure 7.6: The impact of data density in the environment with load intensity having users with similar network capabilities

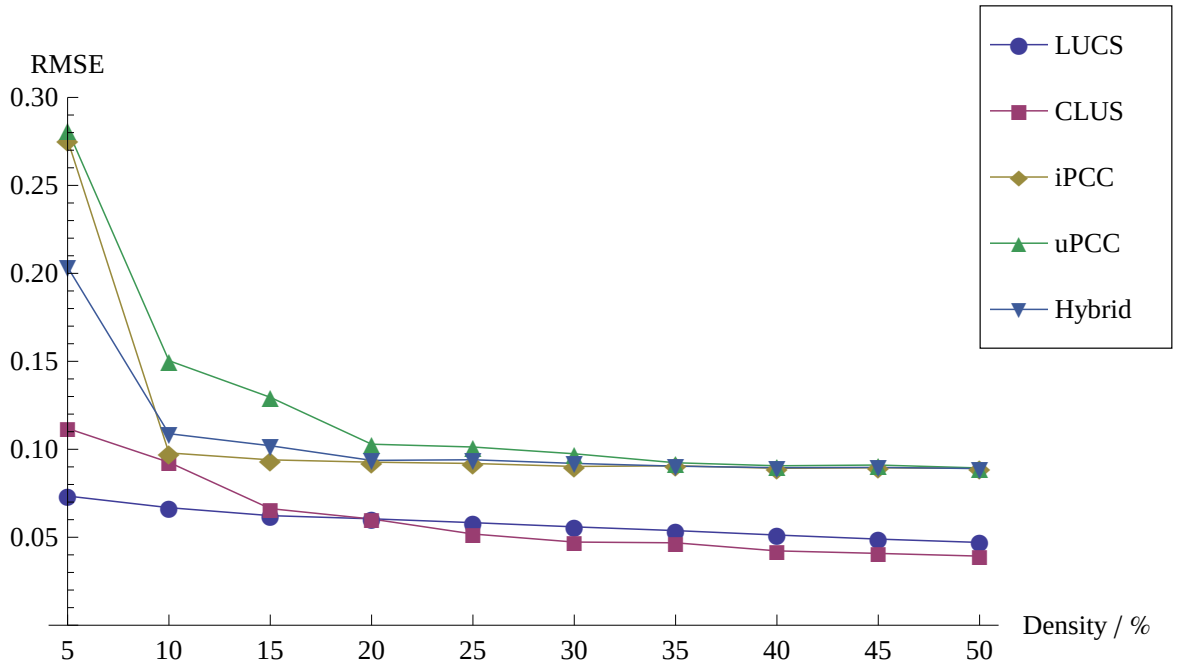
inefficient usage of the collected data.

The evaluation results for the environment with different service loads and having users with different network capabilities are captured in Figures 7.7a and 7.7b. It is obvious from the presented *MAE* and *RMSE* values that proposed approaches provide significantly better prediction accuracy than the existing collaborative filtering based approaches. For the lower data densities *LUCS* approach provides best prediction accuracy (the *MAE* value from 0.054 to 0.026 and the *RMSE* value from 0.073 to 0.047), while *CLUS* approach provides best prediction accuracy for densely populated data (the *MAE* value from 0.076 to 0.019 and the *RMSE* value from 0.111 to 0.039). Such results are expected in the environment where users obtain different network capabilities. The *CLUS* approach achieves best prediction accuracy thanks to consideration of user-specific parameters and users clustering according to their reliability performance. It is interesting that *LUCS* approach still achieves better prediction accuracy for the lower data density. This results is related to the poor clustering performance in *CLUS* approach due to insufficient available data. The collaborative filtering based approaches achieve similar prediction performance like in the first case and the same remarks can be applied in this case.

The Figures 7.8a and 7.8b depict relationship between the data densities obtained for constant service load and the resulting *MAE* and *RMSE* errors. These figures demonstrate that, even in this seldom expected case, the *LUCS* model either outperforms or provides accuracy that is very similar to the accuracy of the other analyzed approaches. In particular, *LUCS* predictions are more accurate for the sparser data density (the *RMSE* value between 0.072 and 0.012 as the density increases), while the *IPCC*, *UPCC*, and *Hybrid* models provide marginally better prediction accuracy for higher data densities (the *RMSE* value between 0.188 and 0.013 for the *IPCC* model). This result is unsurprising since *LUCS* model aggregates the data by grouping the services into service classes. The *CLUS* approach provides better prediction accuracy for the sparser data density (the *RMSE* value between 0.105 and 0.043) when compared to the collaborative filtering based approaches. Note that as the data density increases, prediction accuracy of the *CLUS* approach is remarkably worse which is expected due to *CLUS* approach inability to achieve full precision of services clustering unlike *LUCS*. However, the *CLUS* and specially *LUCS* approach should be considered for prediction even is such case of an environment with constant service load because of their better scalability (further discussed in Section 7.3.2 and Section 7.9).

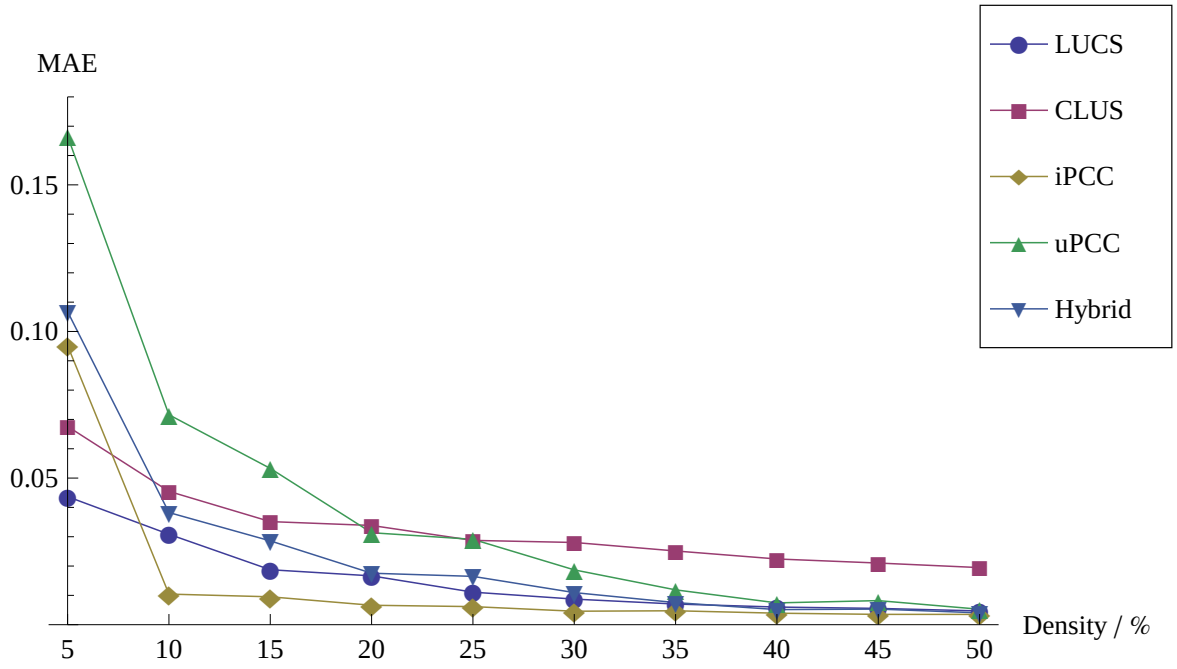


(a) MAE

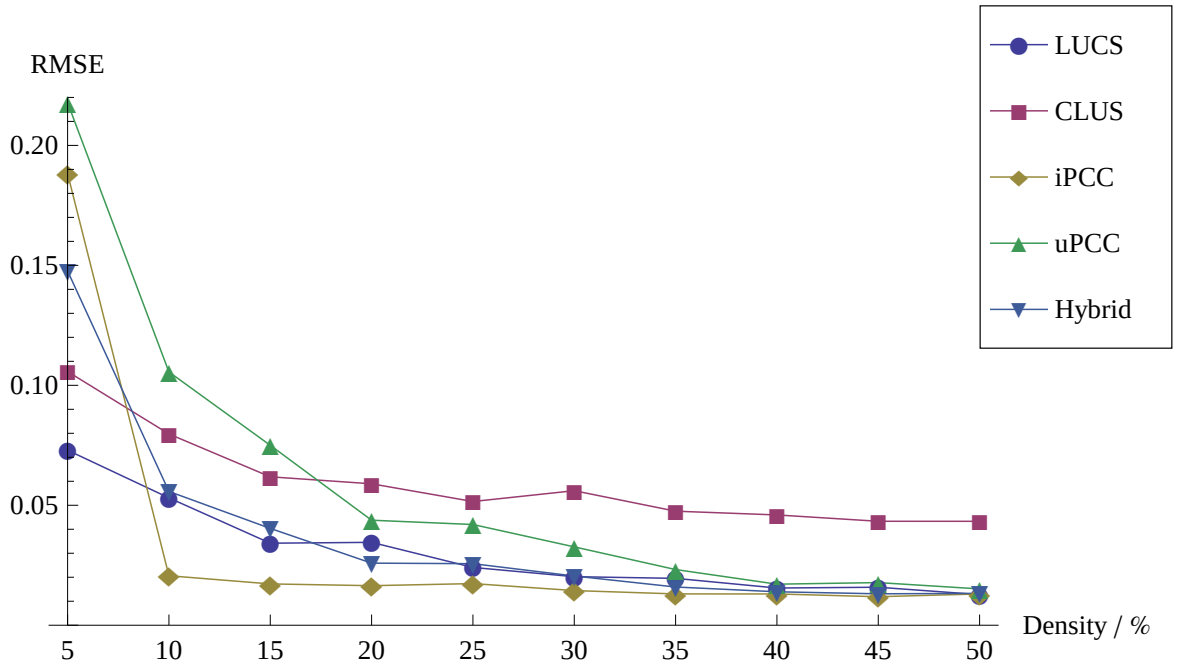


(b) RMSE

Figure 7.7: The impact of data density in the environment with load intensity having users with different network capabilities



(a) MAE



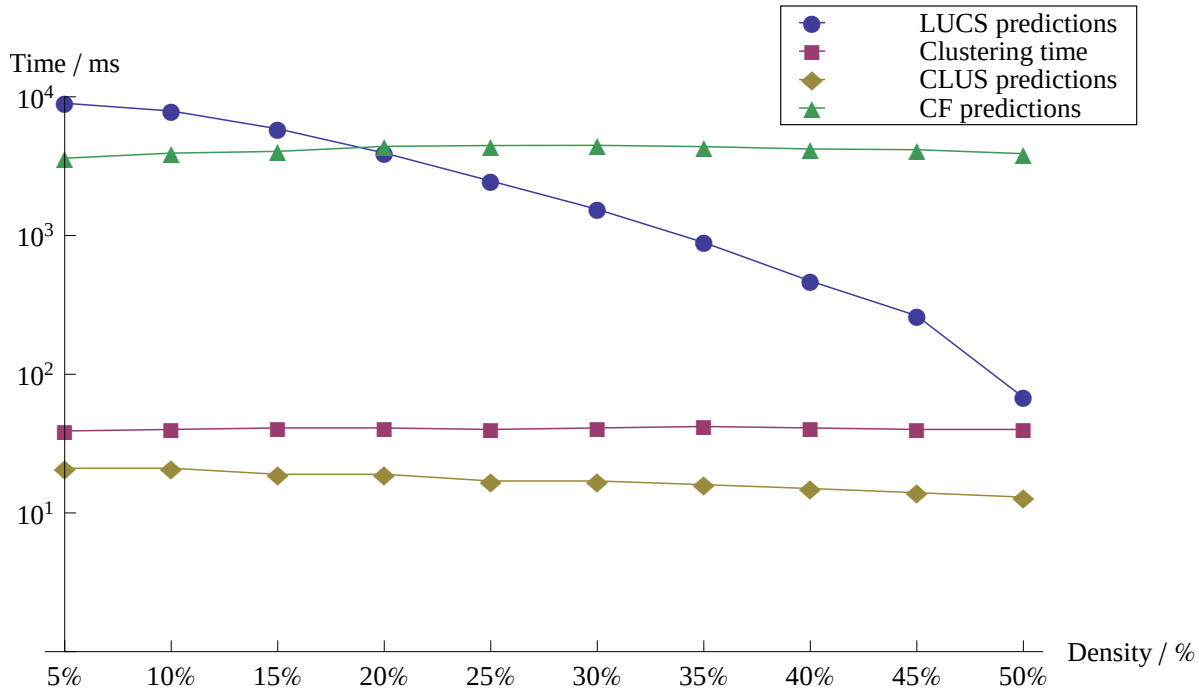
(b) RMSE

Figure 7.8: The impact of data density in the environment without load intensity having users with similar network capabilities

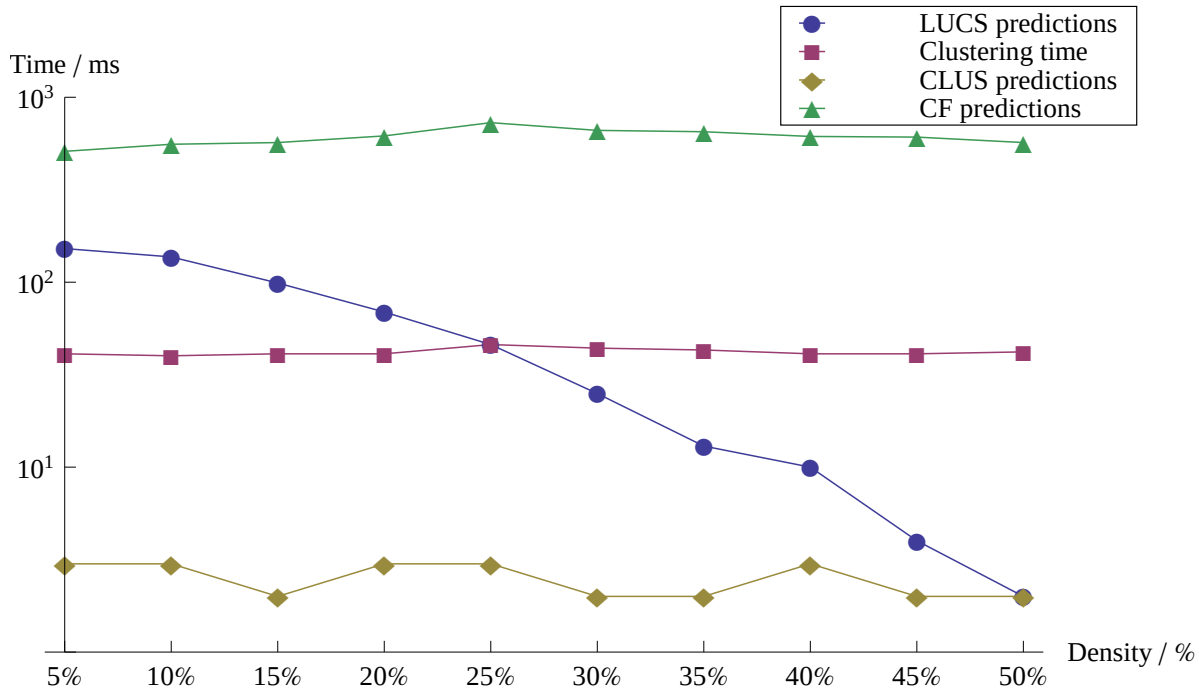
7.3.2 Computational Performance

The evaluation results for computational performance are captured in Figures 7.9a and 7.9b. The execution time that takes to compute the predictions is chosen as the measure for computational performance. The figures depict aggregated *prediction time* for the whole testing set in *milliseconds* in relation to the data density for the *LUCS*, *CLUS* and *collaborative filtering* based approaches in the logarithmic scale. Since all the collaborative filtering approaches have similar analytical complexity (see Section 7.9), the *Hybrid* approach is selected as the representative for the computational performance evaluation. The *CLUS* prediction process is done in two phases *data clustering* and *prediction calculation* as depicted in Figure 6.1. Thus, both *clustering time* and *prediction time* are presented for the *CLUS* approach. Note that the data clustering phase is done only once prior to the prediction phase.

Figure 7.9a depicts computational performance evaluation for the competing approaches in the case of a dynamic environment with different service loads. It is obvious from the presented graphs that the *CLUS* approach provides better performance for the two orders of magnitude (e.g. prediction time of 1543 *ms* for *LUCS* approach and 4437 *ms* for collaborative filtering approach against *CLUS* clustering time of 40 *ms* and prediction time of 17 *ms* for the data density of 30%). Note that both *CLUS* clustering and prediction times are relatively stable as the data density changes. On the other hand, both *LUCS* and collaborative filtering approaches prediction time depends on the data density. In the case of collaborative filtering, for sparser data densities, the computational performance is better (prediction time of 3579 *ms* for the data density of 5%) since low amounts of collected data require less computation. With the increase of the collected data, the computation time increases as can be expected (e.g. prediction time of 4437 *ms* for the density of 30%). As the amount of the collected data continues to increase, the number of records with available reliability values grows. Thus, the reliability value needs to be predicted for fewer records which in turn results in decrease of the prediction time (e.g. prediction time of 3892 *ms* for the density of 50%). In the case of *LUCS* approach, the prediction time highly depends of the data density. For the low data density *LUCS* achieves the longest prediction time among competing approaches (e.g. prediction time of 8893 *ms* for the density of 5%) on collected data set size. Note, that real web application systems contain substantially large number of users and applications which results in a larger data set size. Hence, in a such environment *LUCS* approach performance is expected to be considerably better when compared to collaborative filtering performance. With the increase of data density, the number of records



(a) Prediction time, with load intensity



(b) Prediction time, without load intensity

Figure 7.9: The impact of data density on prediction performance

with available reliability values grows which reduces the computation time (e.g. prediction time of 68 *ms* for the density of 50%).

The computational performance of the prediction in the case of a static environment with constant load is presented in Figure 7.9b. Like in the case of a dynamic environment, the performance is presented in the logarithmic scale. The evaluation results show that *CLUS* approach provides better performance for the order of magnitude whern compared to the collaborative filtering (e.g. prediction time of 662 *ms* for collaborative filtering approach against *CLUS* clustering time of 44 *ms* and prediction time of 3 *ms* for the data density of 30%). The *CLUS* approach provides almost constant clustering and prediction time as the data density changes, while collaborative filtering approaches manifest similar behavior like in the case of a dynamic environment. The *LUCS* approach achieves better prediction performance on the data set size reduced to the constant application load when compared to collaborative filtering (e.g. prediction time of 152 *ms* for the density of 5% and prediction time of 2 *ms* for the density of 50%). Note that *LUCS* approach as the data density increases outperforms the *clustering phase* of the *CLUS* approach. However, these measures are not suitable to be directly compared since the *clustering phase*of the *CLUS* approach is done only once prior to *prediction phase*. The prediction phase of the *CLUS* approach still provides best performance as can be seen in the presented graphs.

7.4 The Significance of Service Load and Class Parameters

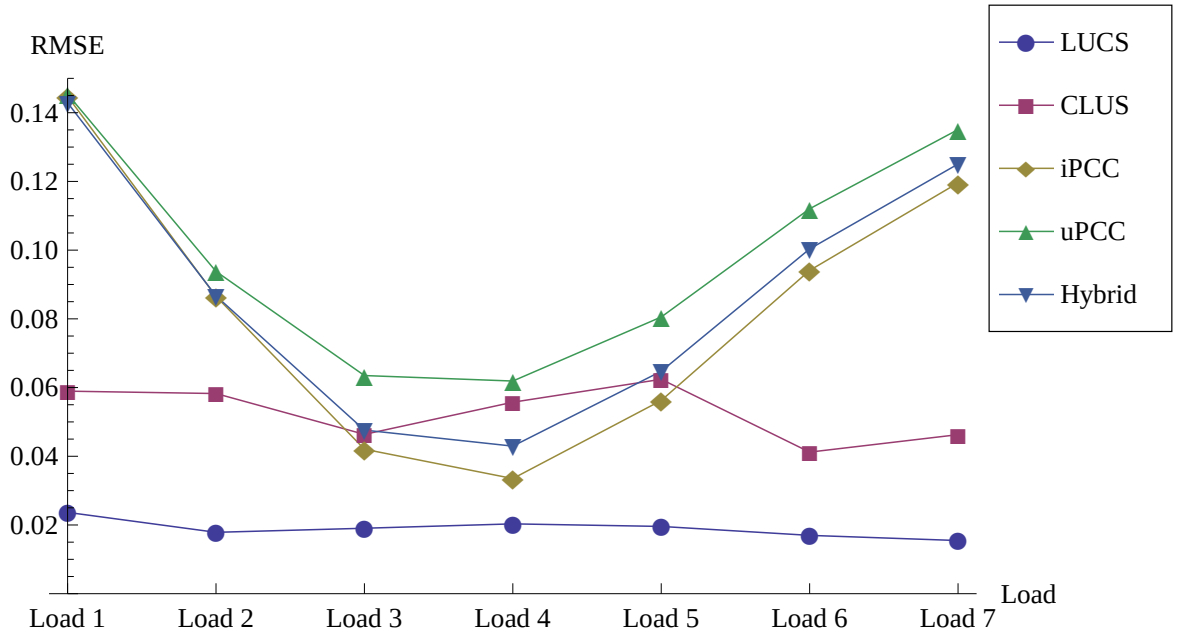
This section studies how two specific parameters, which are considered in the proposed approaches, service load and service class affect the accuracy of reliability predictions.

7.4.1 Significance of Load Parameter

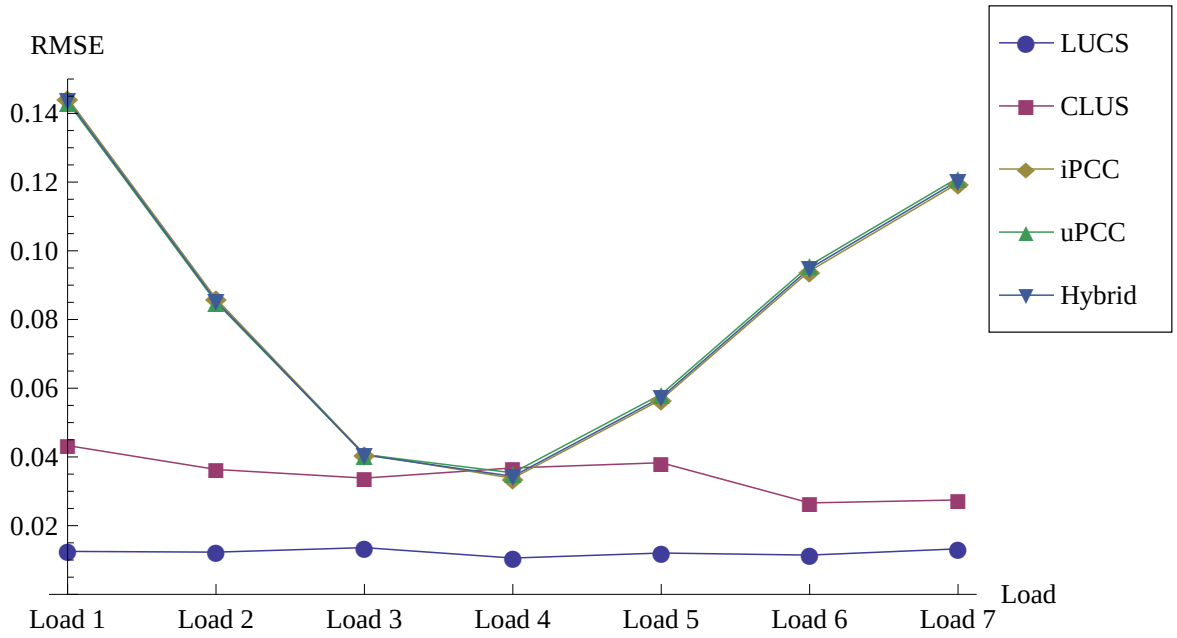
To study the impact of the service load parameter utilized by *LUCS* and *CLUS*, the accuracy of the five prediction models is analyzed when reliability is being predicted for services of different loads. The reader should note that the same analysis is conducted for data densities of 20% and 50% and that two different environments are considered:

1. Environment where users have similar network capabilities, and

2. Environment where users have different network capabilities.

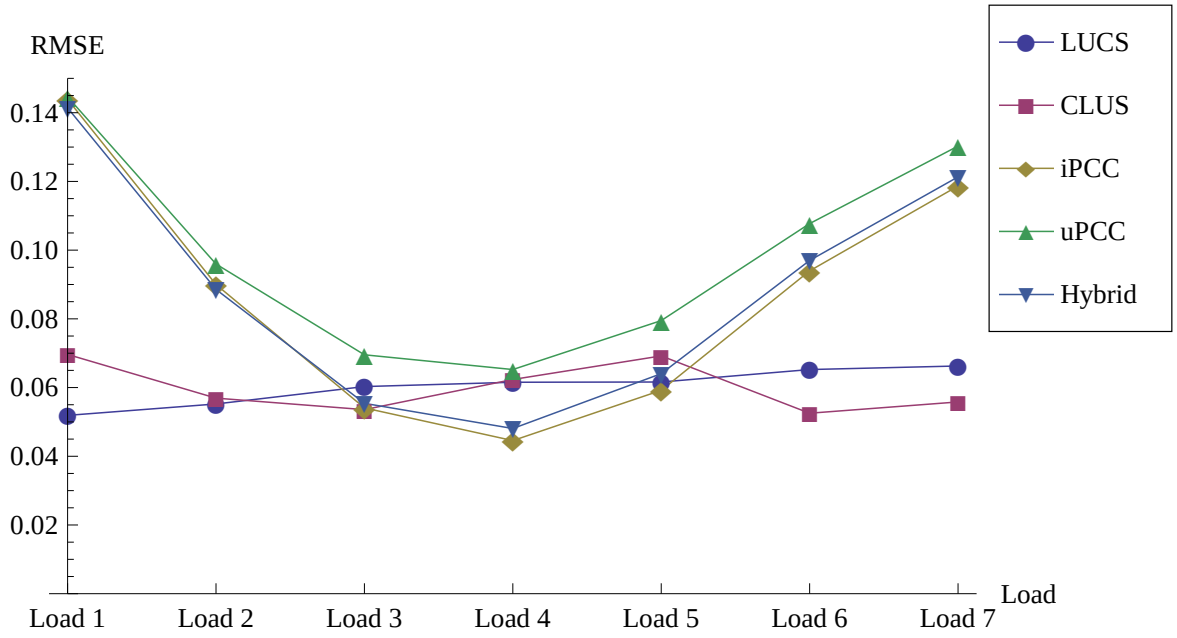


(a) data density 20%

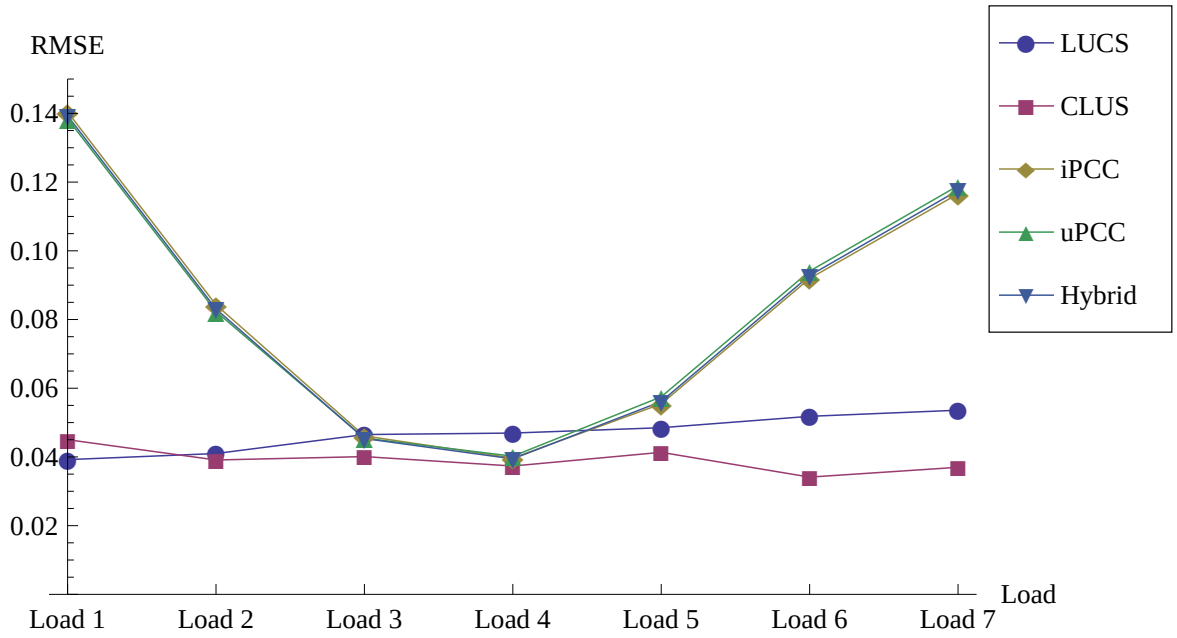


(b) data density 50%

Figure 7.10: The impact of different service loads on prediction accuracy in the environment with users having similar network capabilities



(a) data density 20%



(b) data density 50%

Figure 7.11: The impact of different service loads on prediction accuracy in the environment with users having different network capabilities

The Figures 7.10a and 7.10b depict the relation between the load for the service whose reliability is being predicted and the resulting *RMSE* values for the environment in which users have similar network capabilities. These results support the prior results in that the *LUCS* model provides the best prediction accuracy for each individual load level. Moreover, the prediction errors of the *LUCS* model for different service load levels are fairly similar (the *RMSE* value between 0.010 and 0.013 for different service loads for the density of 50%). The second proposed approach, *CLUS*, provides less accurate predictions compared to predictions of *LUCS*, which is expected in this environment knowing the results from the previous Section 7.3.1. Note, however, that *CLUS*'s predictions are far more accurate and stable when compared to collaborative filtering approaches (the *RMSE* value between 0.026 and 0.043 for different service loads for the density of 50%). By contrast, the errors of the *IPCC*, *UPCC* and *Hybrid* models are highly dependent on the service load of the input service (e.g. for the *Hybrid* model the *RMSE* value varies between 0.023 and 0.126 for different loads for the density of 50%). This is expected since these models do not consider the provider's service load in their prediction calculations. Hence, these models achieve their best prediction accuracy at average loads.

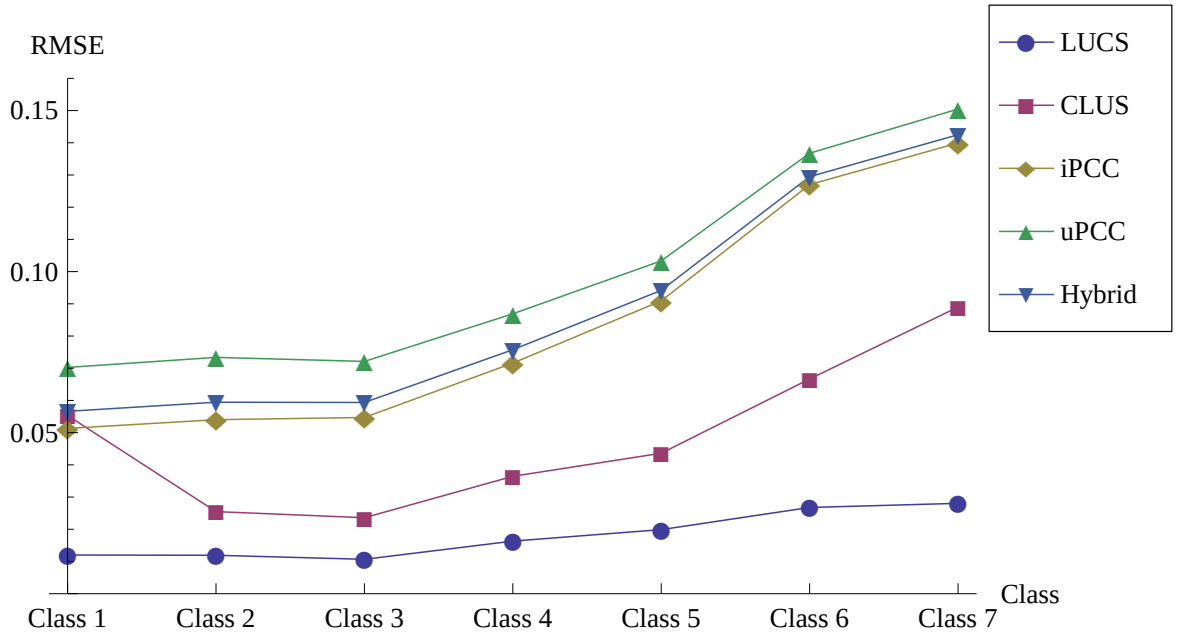
The Figures 7.11a and 7.11b depict the relation between the load for the application whose reliability is being predicted and the resulting *RMSE* values for the environment in which users have different network capabilities. The results draw similar conclusions as the one for the environment considered in the first case, except *CLUS* approach provides better prediction accuracy than *LUCS* approach expectedly (see explanation in Section 7.3.1). Note that in this case, collaborative filtering based approaches provide similar or even better prediction accuracy for average loads (especially for the lower data density) than *CLUS* and *LUCS* approaches.

7.4.2 Significance of Class Parameter

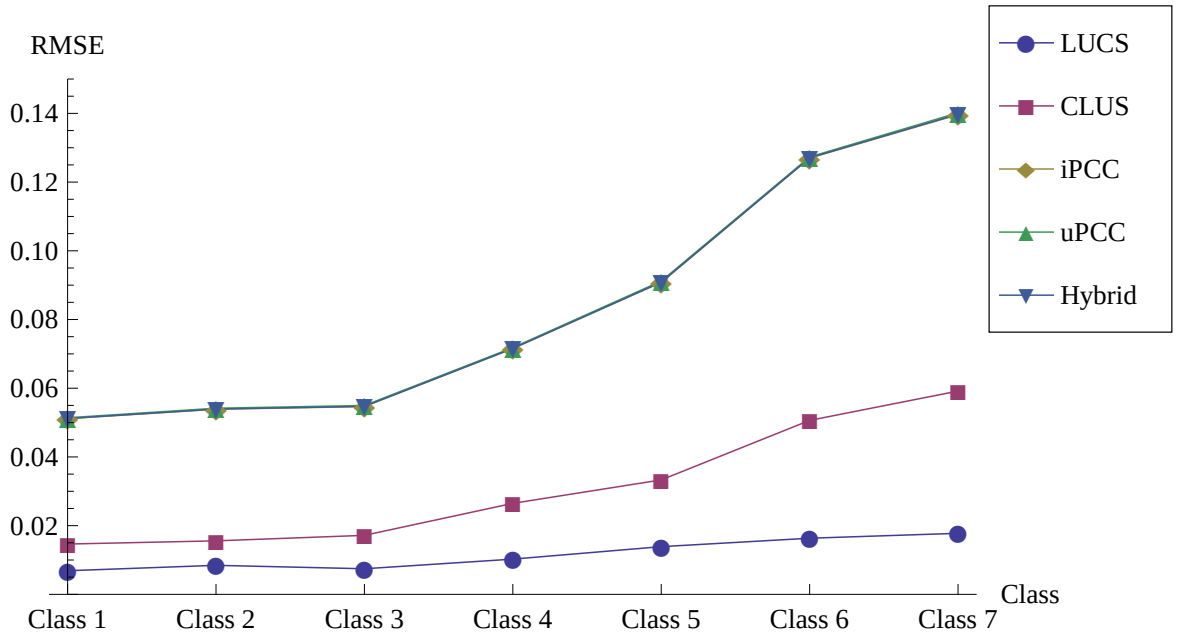
Similarly to the case of service loads, the accuracy of the five prediction models is analyzed when reliability is being predicted for services of different computational classes. The reader should note that the same analysis is conducted for data densities of 20% and 50% and that two different cases are considered:

1. Environment where users have similar network capabilities, and

2. Environment where users have different network capabilities.

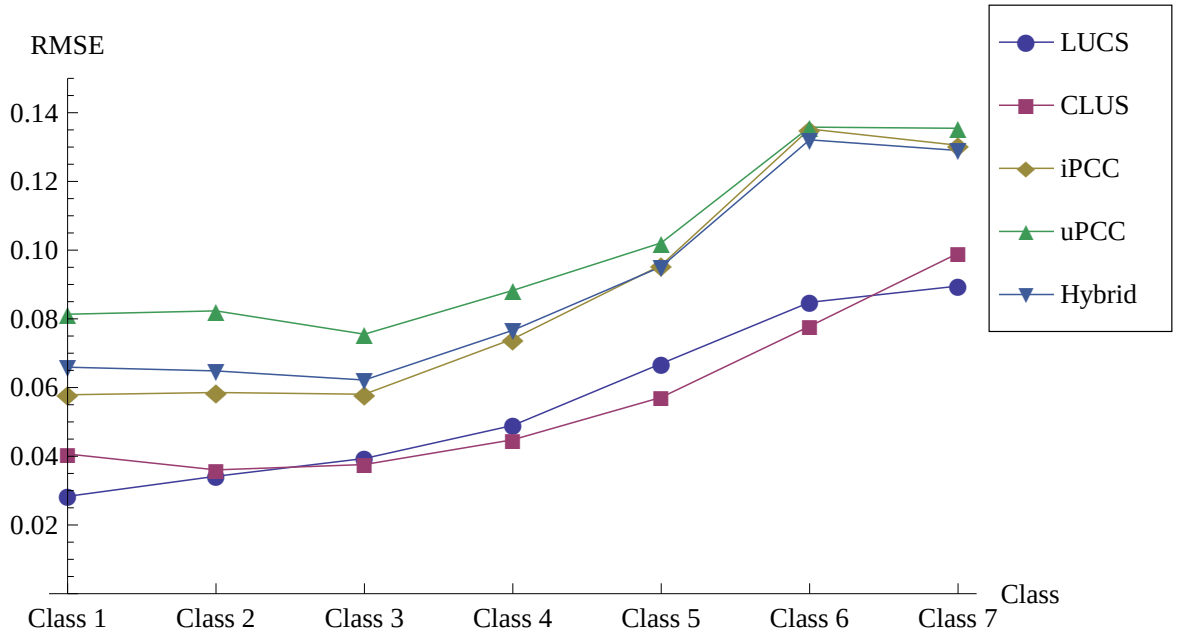


(a) data density 20%

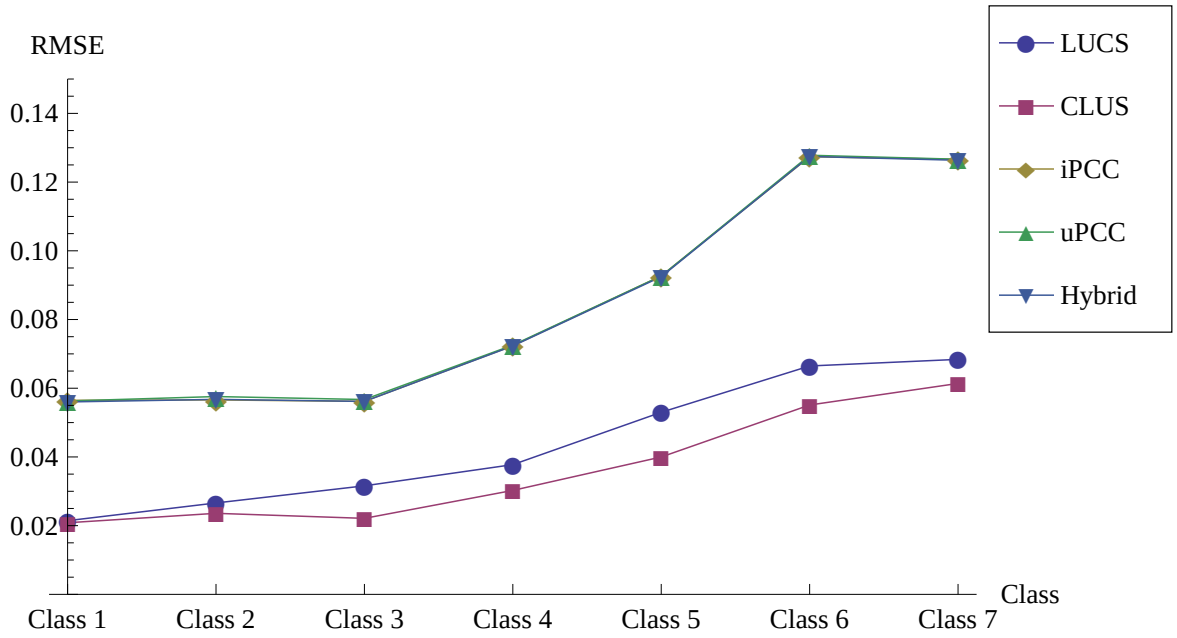


(b) data density 50%

Figure 7.12: The impact of different service classes on prediction accuracy in the environment with users having similar network capabilities



(a) data density 20%



(b) data density 50%

Figure 7.13: The impact of different service classes on prediction accuracy in the environment with users having different network capabilities

The Figures 7.12a and 7.12b capture the distribution of prediction errors for different service classes in the environment in which users obtain similar network capabilities. The results indicate that each respective model provides better prediction accuracy for predicting reliability of computationally heavier service classes. This result is unsurprising given that the lighter service classes (*class 6* and *class 7*) are less similar to other classes (referred in literature as *gray sheep* [18]). Thus, the sets of similar entities for those classes are more limited compared to other classes, which increases the prediction error. Note, however, that this error increase is significantly less prominent for *LUCS* (the *RMSE* value between 0.006 and 0.017 for different classes for the density of 50%) and *CLUS* (the *RMSE* value between 0.014 and 0.059 for different classes for the density of 50%) approaches than the collaborative filtering based models (i.e the *RMSE* value for the *Hybrid* model between 0.051 and 0.139 for different classes for the density of 50%), which makes proposed approaches, specially *LUCS*, more reliable for real-world systems with rich types of services.

The Figures 7.13a and 7.13b show the prediction errors distribution on different service classes in the environment in which users achieve different network capabilities. The results draw similar conclusions as in the first case, except *CLUS* approach outperforms *LUCS* approach which is quite expected knowing the discussion in Section 7.3.1.

It should be noted that similar results and conclusions regarding *LUCS*'s and *CLUS*'s specific parameters are obtained if *MAE* values are studied.

7.5 The Importance of Each Individual *LUCS*'s Input Parameter

Section 5.1.2 states that *LUCS* provides the reliability prediction even in the case when some of the input parameters are missing. The input parameters u (user location) and s (service location) can be easily determined by checking the IP address of the user and the service. However, determining parameters l (service load) and c (service class) can be challenging in case the service provider is unwilling to immediately share them. Furthermore, when a new service provider appears in the system, the provider's load can only be approximated since the load curve is not available yet. Similarly, the service class may be difficult to determine for a new service with certain notable exceptions (e.g., when deploying a service on a third-party cloud infrastructure, the computational demands are often specified in advance). In order to deter-

mine how the access to the input parameters impacts the performance of *LUCS*, two tests were conducted. The first test studies the impact of each individual parameter available on *LUCS* approach prediction accuracy. The second test assesses how lack of each individual parameter influence the prediction accuracy of *LUCS*. These tests were performed on the testing sets comprising all collected records during the experiments. For the purpose of input parameters importance evaluation, two different environments are considered:

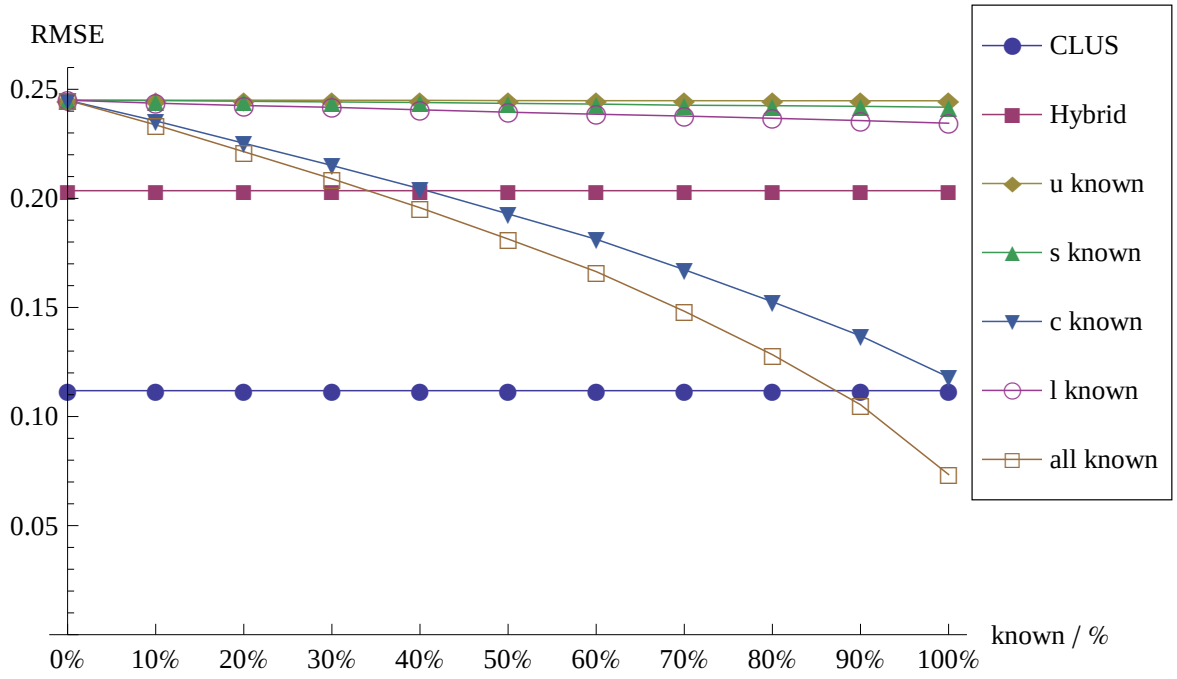
1. Environment where users have different network capabilities, and
2. Environment where users have similar network capabilities.

7.5.1 The Impact of Individual Input Parameter Available

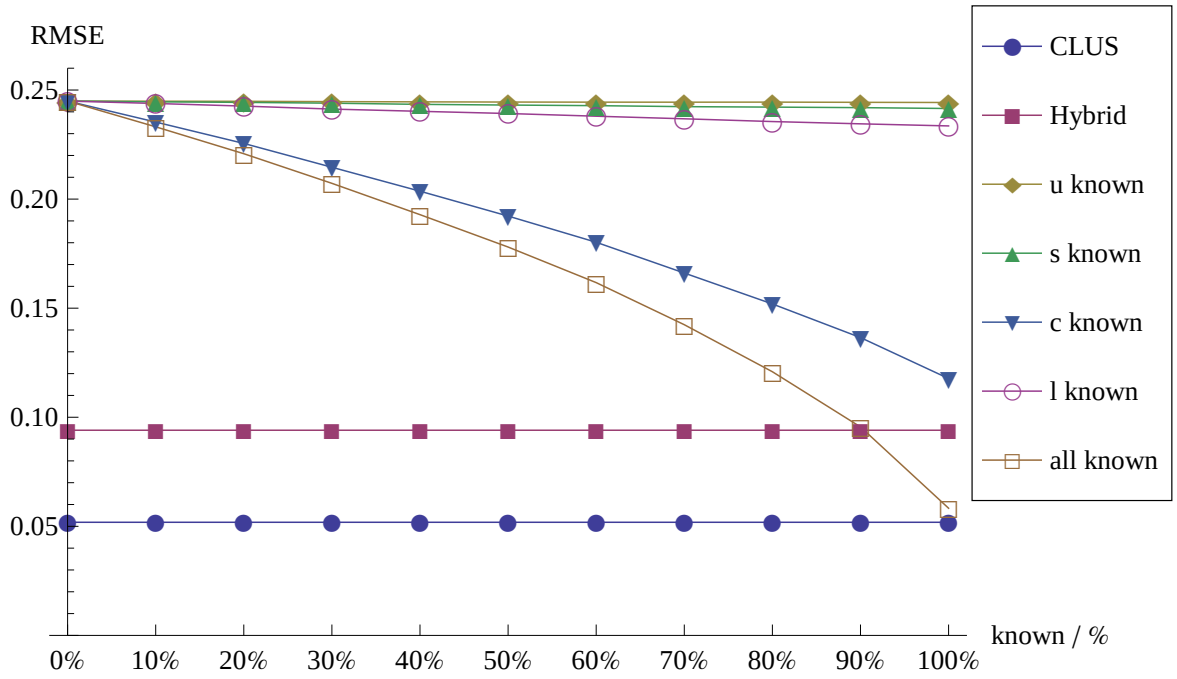
In the first test, the impact of availability of each particular *LUCS* parameter is considered. The *RMSE* values are calculated for the testing set while considering only a single input parameter with varying rates of points having considered parameter available from 0% to 100%.

The Figures 7.14a and 7.14b show the *RMSE* values obtained in the test for two different data densities for the environment where users have different network capabilities. Additionally, the figures depict the *RMSE* for *CLUS* and *Hybrid* approaches which are not affected by the availability of *LUCS*' s input parameters. As expected, the availability of the input parameters highly impacts the accuracy of the *LUCS* model. In particular, for the lower data density (Figure 7.14a), the service class is a parameter that can be used in isolation to predict service reliability more accurately than the *Hybrid* model (in the case the service class parameter is known for at least 40% of all invocations in the testing set), but still it can not be used to achieve the accuracy of the *CLUS* approach. By contrast, the service load l, user location u, and service location s parameters used in isolation can not accurately predict the reliability of a current invocation. For high data density (Figure 7.14b), no single parameter can be used in isolation to achieve better accuracy than the *Hybrid* or *CLUS* approach.

The Figures 7.15a and 7.15b show the *RMSE* values obtained in the test for two different data densities for the environment where users have similar network capabilities. In this environment, knowing the results from previous discussions (see Section 7.3.1), *LUCS* approach achieves even better prediction accuracy. Thus, for lower data densities, (Figure 7.15a), the service class is a parameter that can be used in isolation to predict service reliability more accurately than the *Hybrid* model (in the case the service class parameter is known for at least

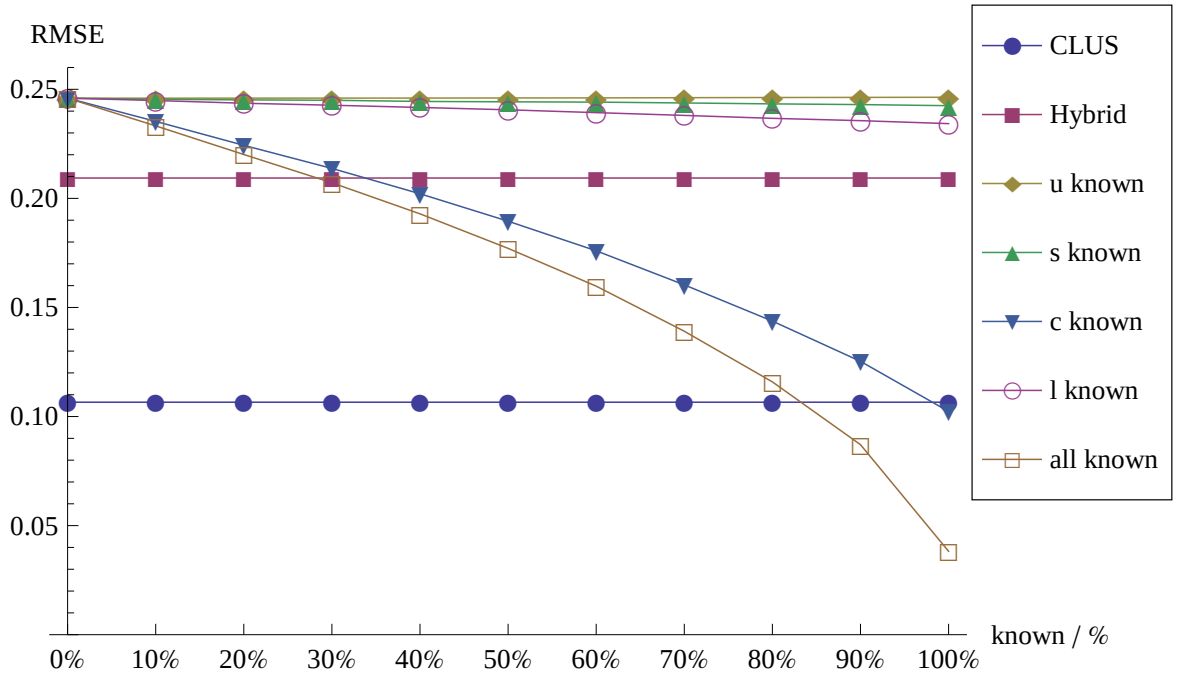


(a) data density 5%

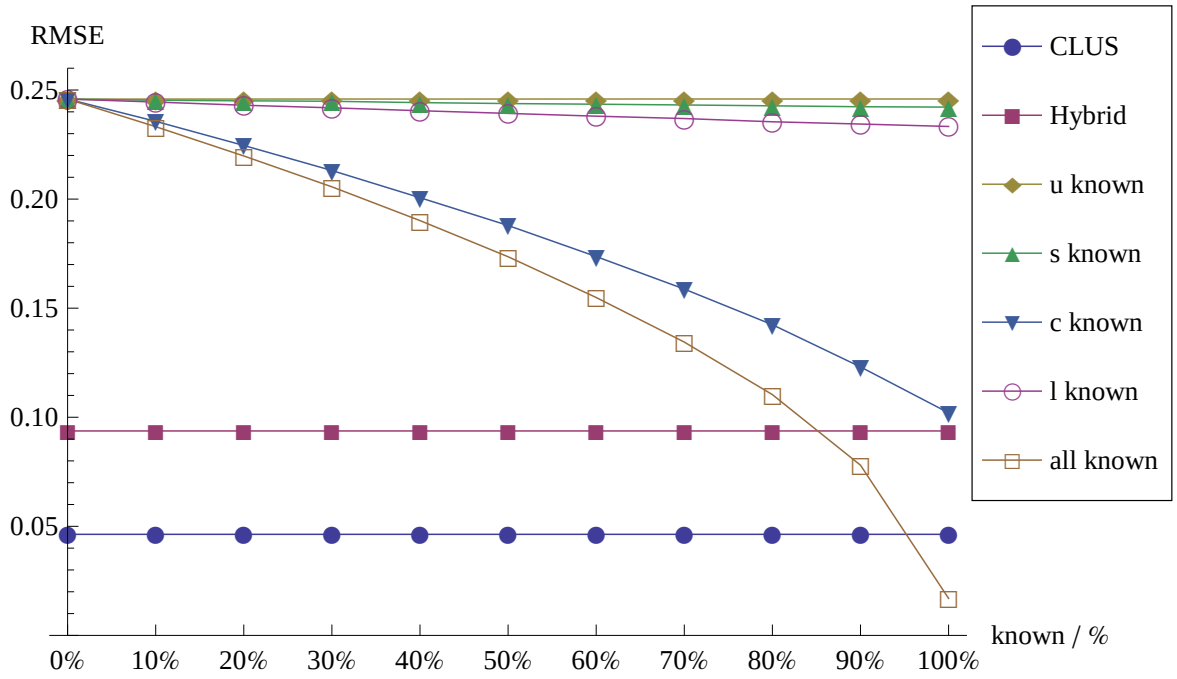


(b) data density 25%

Figure 7.14: The impact of individual input parameters on the *LUCS* prediction accuracy in the environment in which users have different network capabilities



(a) data density 5%



(b) data density 25%

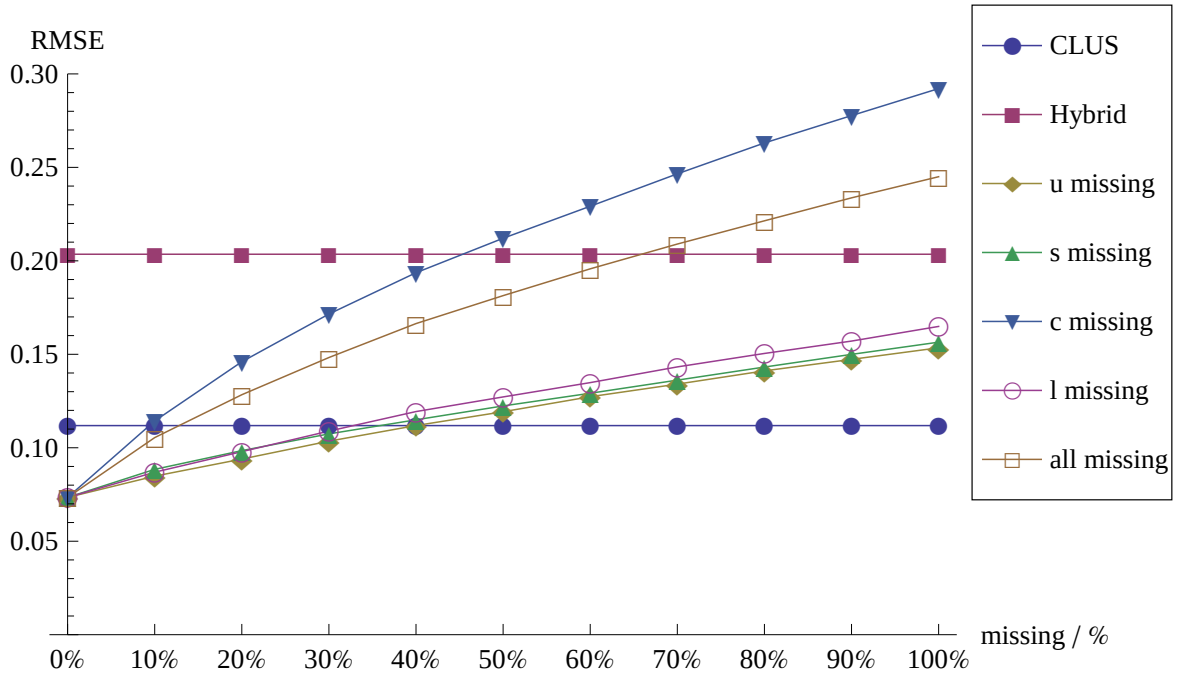
Figure 7.15: The impact of individual input parameters on the *LUCS* prediction accuracy in the environment in which users have similar network capabilities

35% of all invocations in the testing set). However, it can not be used to achieve the accuracy of the *CLUS* approach unless the parameter is available for all the invocations in the testing set. Other parameters can not be used in isolation to achieve the prediction accuracy of the *CLUS* approach. For high data density (Figure 7.15b), no single parameter can be used to predict the service reliability more accurately than the *Hybrid* or *CLUS* approach.

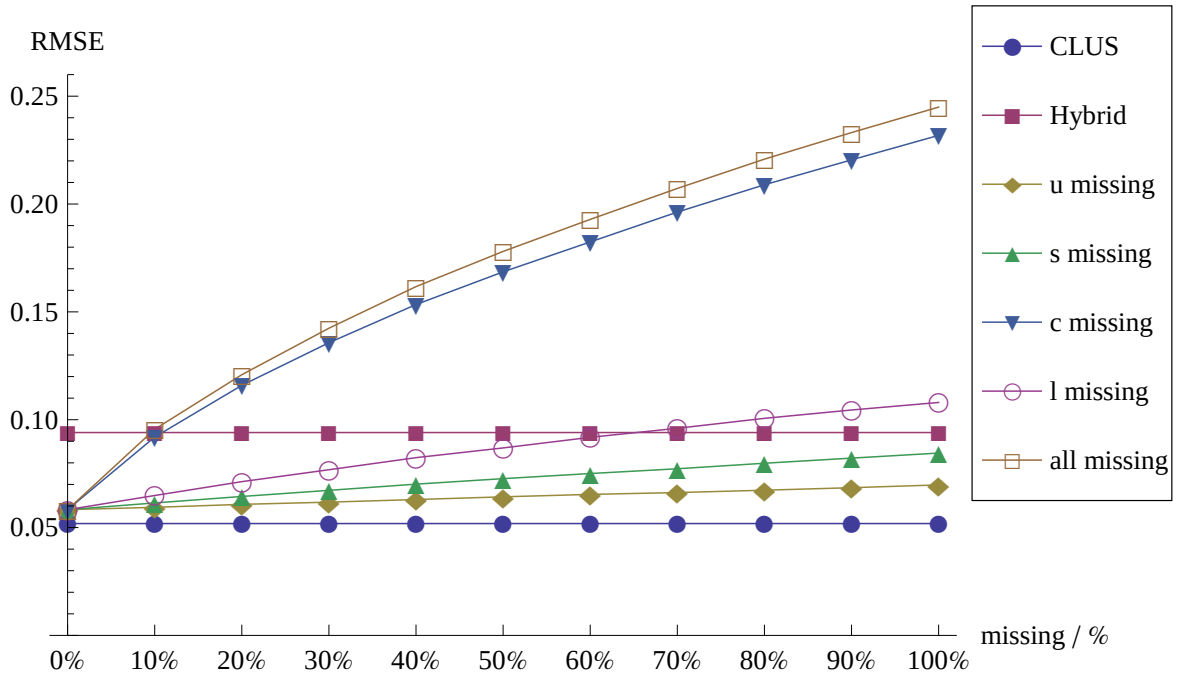
7.5.2 The Impact of Individual Input Parameter Missing

In the second test, the impact of lack of each particular *LUCS* parameter on prediction accuracy is considered. The *RMSE* values are calculated for the testing set while having three input parameters readily available and the fourth one missing with varying rates of points with the missing parameter from 0% to 100%.

The Figures 7.16a and 7.16b show the *RMSE* values obtained in the test for two different data densities for the environment where users have different network capabilities. Note that the "all missing" group in the figures varies the availability of all of the parameters together (e.g., 80% missing for this group means that none of the input parameters is available in four out of five cases). The collected results indicate that the lack of service class parameter c highly decreases the prediction accuracy of the model. For lower data density, it is important to note that, according to Figure 7.16a, *LUCS* exhibits better accuracy with all of its input parameters missing than when only parameter c is missing. This behavior is a consequence of the fact that, with all of the parameters missing, *LUCS* predicts the average values calculated on the whole space D. By contrast, when only c parameter is missing, *LUCS* calculates the average values on fewer points that are determined according to the other known parameters. Since the reliability varies widely for two of seven service classes (recall Section 7.4.2), the errors become higher as some of the groupings in the sparse 3-dimensional space will have records for those service classes, while others will not. In addition, the graphs show that the lack of load parameter l degrades *LUCS* prediction accuracy but not as much as the lack of class parameter c. The lack of parameters u and s degrades the prediction accuracy of *LUCS*. However, in this case the model still achieves better prediction accuracy than the *Hybrid* approach. In particular, for lower data density, as depicted in Figure 7.16a, the *LUCS* model can tolerate up to 10% of individual parameters missing and still achieve better performance than *CLUS* model. Also, the model can tolerate up to 45% of individual parameters missing and still achieve better performance than the *Hybrid* approach. For high data density, as depicted in Figure 7.16b, the *LUCS* model

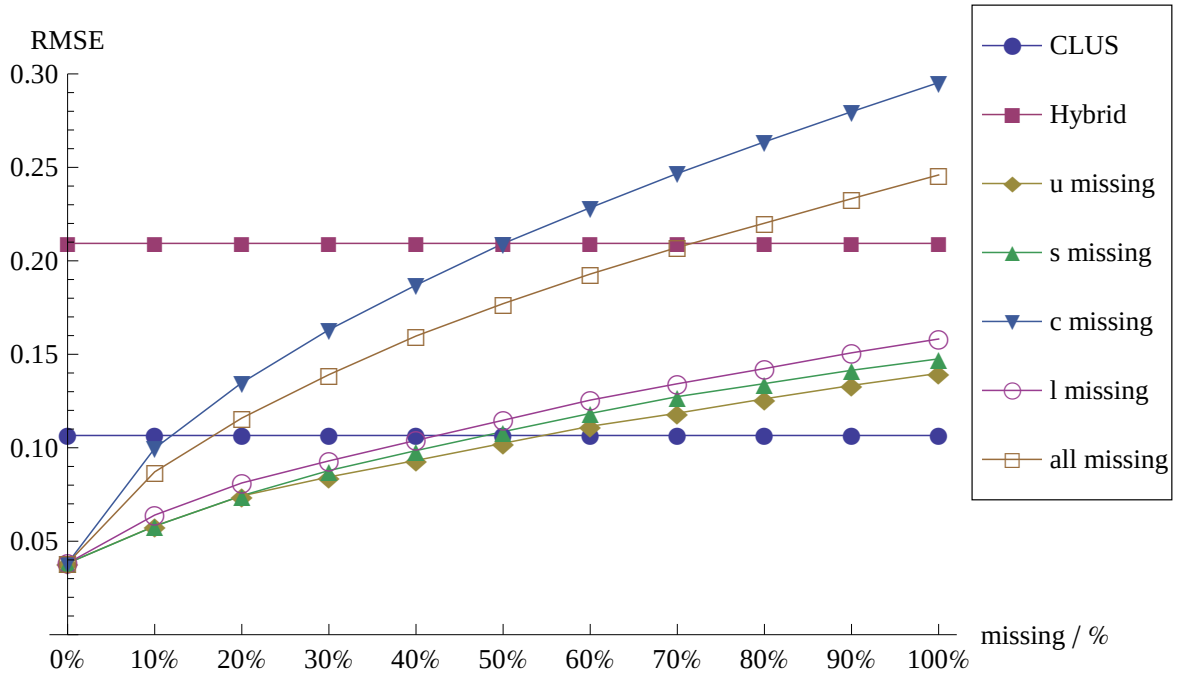


(a) data density 5%

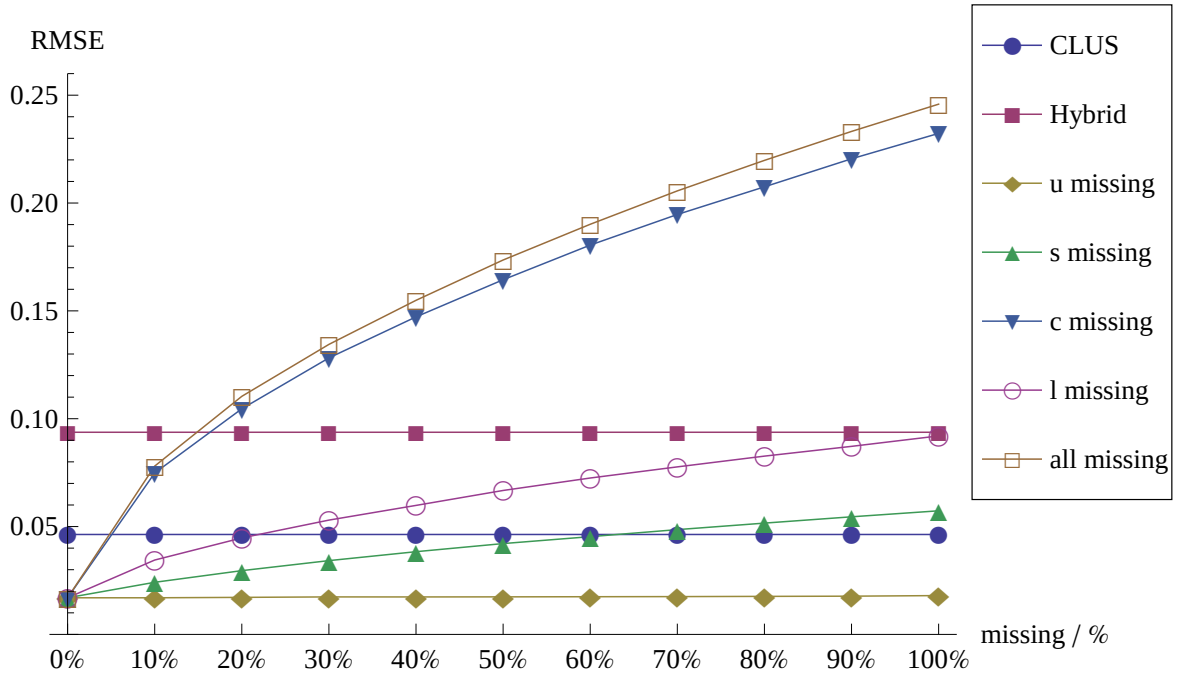


(b) data density 25%

Figure 7.16: The impact of lack of individual input parameters on the *LUCS* prediction performance in the environment in which users have different network capabilities



(a) data density 5%



(b) data density 25%

Figure 7.17: The impact of lack of individual input parameters on the *LUCS* prediction performance in the environment in which users have similar network capabilities

can tolerate up to 10% of individual parameters missing and outperform the state-of-the-art the *Hybrid* model.

The Figures 7.17a and 7.17b capture the *RMSE* values obtained in the test for two different data densities for the environment where users have similar network capabilities. Quite similar conclusions can be drawn as in the case of the environment where users have different network capabilities. The parameter class c proves to be the most important for the *LUCS* approach prediction accuracy, while lack of parameter load l significantly degrades the prediction accuracy. However, *LUCS* approach manifests considerable tolerance to lack of u and s parameters. Specifically, for the lower data density, as depicted in Figure 7.17a, the *LUCS* model can tolerate up to 10% of individual parameters missing and still achieve better performance than *CLUS* model. Also, the model can tolerate up to 50% of individual parameters missing and still achieve better performance than the *Hybrid* approach. For high data density, as depicted in Figure 7.17b, the *LUCS* model can tolerate up to 5% of individual parameters missing and outperform the *CLUS* model. Also, the model can tolerate up to 15% of individual parameters missing and still achieve better performance than the state-of-the-art *Hybrid* approach.

In general, our results suggest that the *LUCS* model is limited to the environments where the input parameters are highly available.

7.6 The Sensitivity of *LUCS* Groupings

The evaluations presented in the previous sections confirm the importance of the two *LUCS*'s specific input parameters—service load l and service class c. However, a potential obstacle to considering these two input parameters is that grouping the services into similar computational classes or determining the load at the time of the current request (recall Section 5.2.2) may be a difficult and error-prone tasks. For example, a lack of sufficient information about a service's computational demands may result in an incorrect classification of that service as a computationally less intensive application. Similarly, a recently developed services may have limited accompanying information about its average load. Hence, the magnitude of the risk that the prediction accuracy of *LUCS* is highly dependent on fully correct groupings is analyzed.

In order to evaluate the sensitivity of *LUCS* on the correctness of the parameter groupings, *fault injection* [49, 206] technique is utilized. Specifically, a testing set containing all points from the four-dimensional space D is used, and a currently invoked service is allowed to be

incorrectly classified into the neighboring groupings. For example, this fault injection process can group a service that actually belongs to the computational *class 3* into the computational *class 2* or computational *class 4* instead. Similarly, when the current load of a servuce is *req/5 sec*, the current service load can be erroneously classified as *req/4 sec* or *req/6 sec*. The effect of these errors is analyzed by calculating the *RMSE* for different grouping error rates ranging between 0% and 100%. The reader should note that the same analysis is conducted for data densities of 5% and 25% and that two different cases are considered:

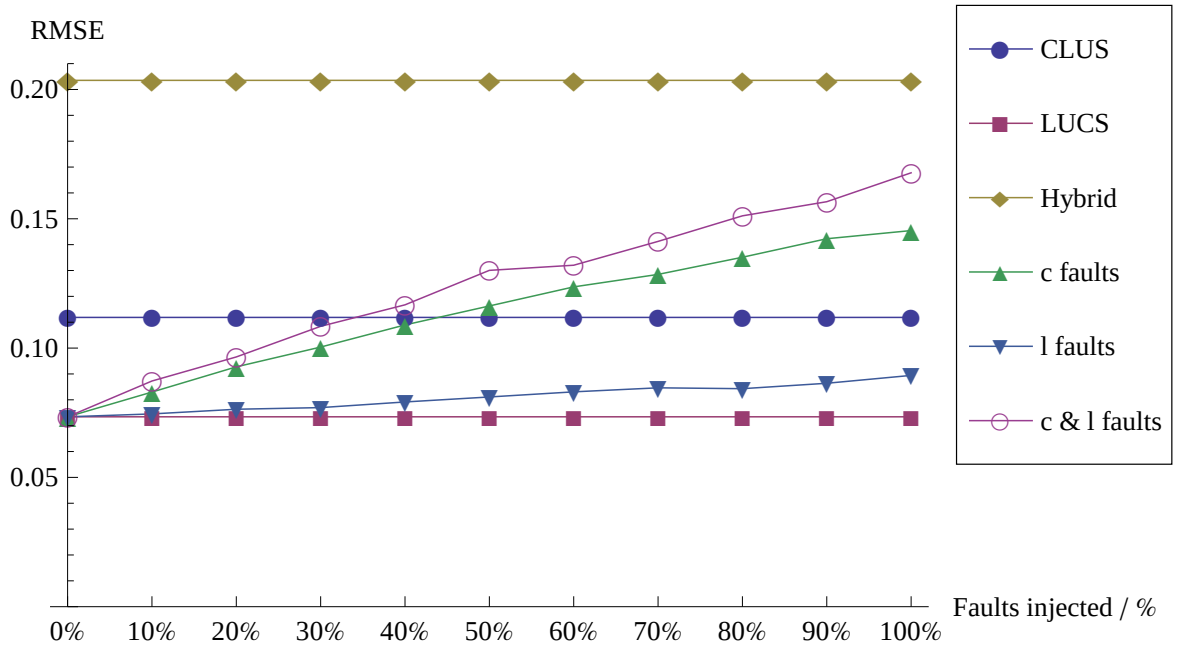
1. Environment where users have different network capabilities, and

2. Environment where users have similar network capabilities.

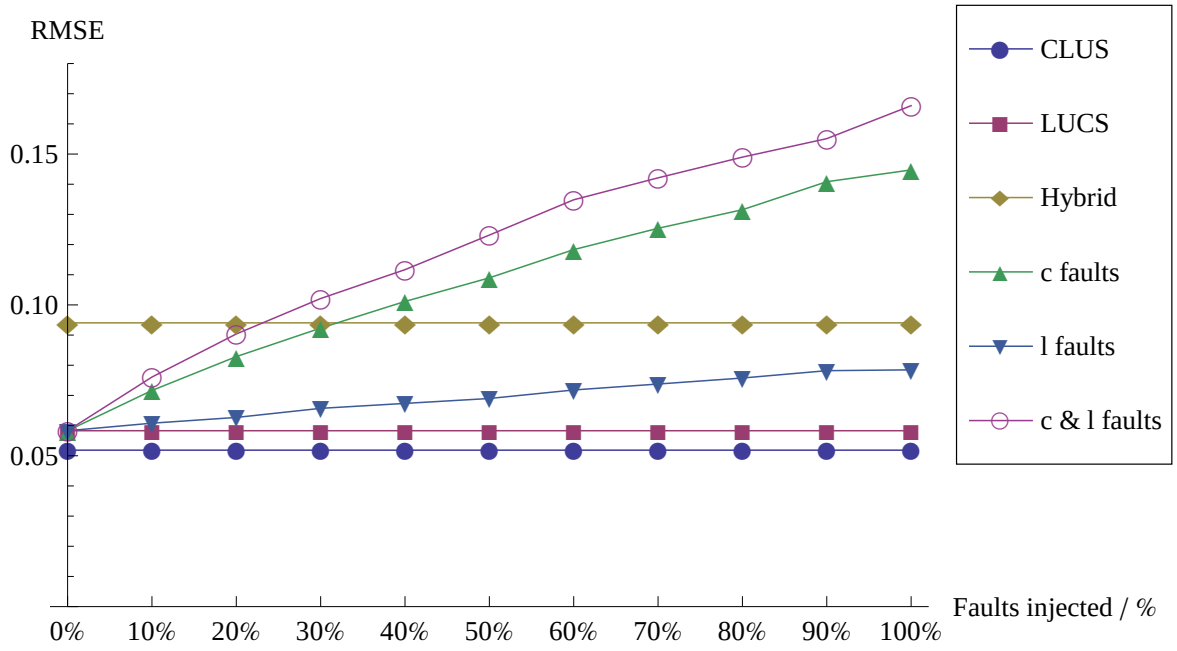
The Figures 7.18a and 7.18b depict the results for two different data densities for the environment where users have different network capabilities. The figures include *RMSE* values of error-free *LUCS*, *CLUS* and *Hybrid* models as well as the *LUCS* model with faults injected into load and class groupings in isolation, and load and class groupings at the same time. As expected, the obtained results confirm that incorrect groupings impact the prediction accuracy. However, this impact is generally acceptable since, for low data densities, *CLUS* outperforms *LUCS* only once more than 40% of service class groupings are erroneous, and *Hybrid* does not outperform *LUCS* even in the extreme case when every analyzed request is classified in a wrong way. For high data densities, *Hybrid* outperforms *LUCS* once more than 30% of service class groupings are erroneous. Our results also indicate that *LUCS* is resilient to errors in the service load groupings as these incorrect groupings decrease the accuracy by only up to a couple percentage points. *LUCS* is more sensitive to incorrect classification of the computational service classes, but the accuracy is still better than that of the *Hybrid* model even for high error rates.

The Figures 7.19a and 7.19b depict the results for two different data densities for the environment where users have similar network capabilities. The figures include *RMSE* values of error-free *LUCS*, *CLUS* and *Hybrid* models as well as the *LUCS* model with faults injected into load and class groupings in isolation, and load and class groupings at the same time. The results presented in figures suggest similar conclusions like for the environment where network capabilities of users differ, except *LUCS* approach manifests even better performance as can be expected (see Section 7.3.1).

It should be noted that similar or better results in favor of *LUCS* are obtained if *MAE* values are analyzed.

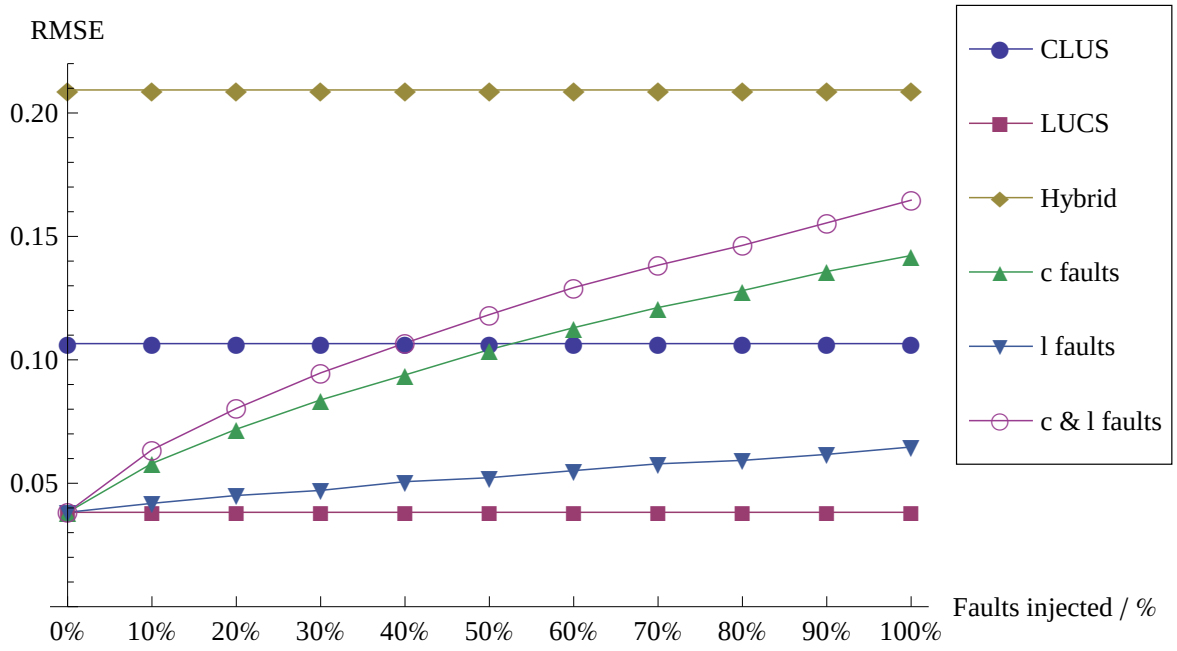


(a) data density 5%

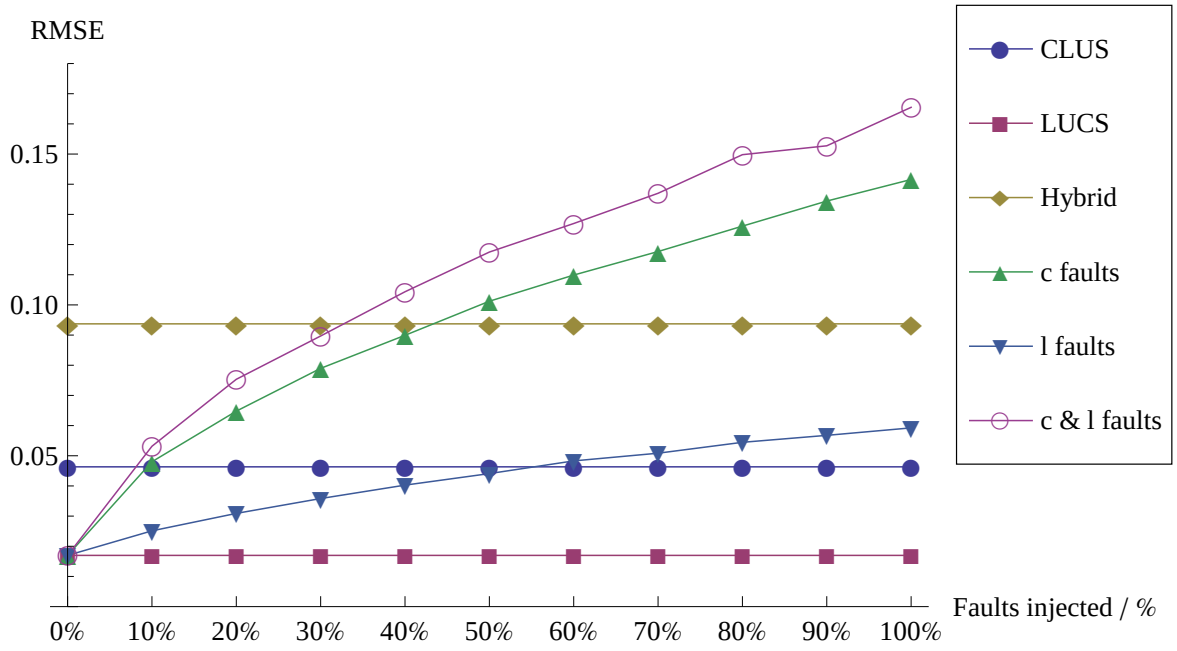


(b) data density 25%

Figure 7.18: The sensitivity of LUCS groupings in the environment where users have different network capabilities



(a) data density 5%



(b) data density 25%

Figure 7.19: The sensitivity of LUCS groupings in the environment where users have different network capabilities

7.7 The Heuristics for *LUCS* Model's Parameters α, β and γ

The *LUCS* model utilizes Equation 5.32 to predict reliability based on the different parameters. The evaluations so far presented utilized the equal value of 0.25 for each coefficient α, β and γ. While these values were sufficient to demonstrate the different aspects and the advantages of *LUCS* in terms of its accuracy, these values are not optimal for the given environment. To achieve better performance, fine tunning of the coefficients α, β and γ must be performed for each quadruple $D[u, l, s, c]$. To calculate better coefficient values and improve the prediction accuracy, the prediction accuracy should be analyzed for each *LUCS* parameter p_u, p_s, p_l, p_c in the Equation 5.32. Below, a heuristic for calculating better coefficient values is proposed.

Let us assume a quadruple $D[u, l, s, c]$ and $\alpha[u, l, s, c]$, $\beta[u, l, s, c]$ and $\gamma[u, l, s, c]$ need to be calculated. First, each parameter's mean *RMSE* for the past invocations is calculated as follows:

$$\overline{R_U[u, s, l, c]} = RMSE(rmse_{U_u}, rmse_{U_s}, rmse_{U_l}, rmse_{U_c}) \tag{7.3}$$

$$\overline{R_S[u, s, l, c]} = RMSE(rmse_{S_u}, rmse_{S_s}, rmse_{S_l}, rmse_{S_c}) \tag{7.4}$$

$$\overline{R_L[u, s, l, c]} = RMSE(rmse_{L_u}, rmse_{L_s}, rmse_{L_l}, rmse_{L_c}) \tag{7.5}$$

$$\overline{R_C[u, s, l, c]} = RMSE(rmse_{C_u}, rmse_{C_s}, rmse_{S_l}, rmse_{C_c}) \tag{7.6}$$

where each $rmse_{i_j}$ represents *RMSE* of the impact $i \in \{U, S, L, C\}$ at the specific $j \in \{u, s, l, c\}$. The goal is to assign the parameters with smaller aggregated *RMSE* a higher degree of influence when calculating the final prediction. Thus, the coefficients of the linear combination are defined as follows:

$$\alpha[u, l, s, c] = \frac{\overline{R_U[u, s, l, c]}^{-1}}{\sum_{I \in \{U,S,L,C\}} \overline{R_I[u, s, l, c]}^{-1}} \tag{7.7}$$

$$\beta[u, l, s, c] = \frac{\overline{R_S[u, s, l, c]}^{-1}}{\sum_{I \in \{U,S,L,C\}} \overline{R_I[u, s, l, c]}^{-1}} \tag{7.8}$$

$$\gamma[u,l,s,c] = \frac{\overline{R_L[u,s,l,c]}^{-1}}{\sum_{I \in \{U,S,L,C\}} \overline{R_I[u,s,l,c]}^{-1}} \tag{7.9}$$

To test the proposed heuristic, the prediction algorithm is run and compared to the prediction accuracy for the raw *LUCS* model with fixed parameters ($\alpha = \beta = \gamma = 0.25$) and the tuned *LUCS* model with parameter values calculated using the proposed heuristic.

The Figures 7.20a and 7.20b depict the *MAE* and *RMSE* values of the basic and the heuristics tuned *LUCS* models with predictions made based on the whole space D varying the data density from 2% to 20% with a step value of 2. The results show that the fine tunning of the model parameters and the adjustment of parameters to the specific environment further improved the model's performance, specially for the sparser data density.

7.8 The Impact of *CLUS*'s Number of Clusters

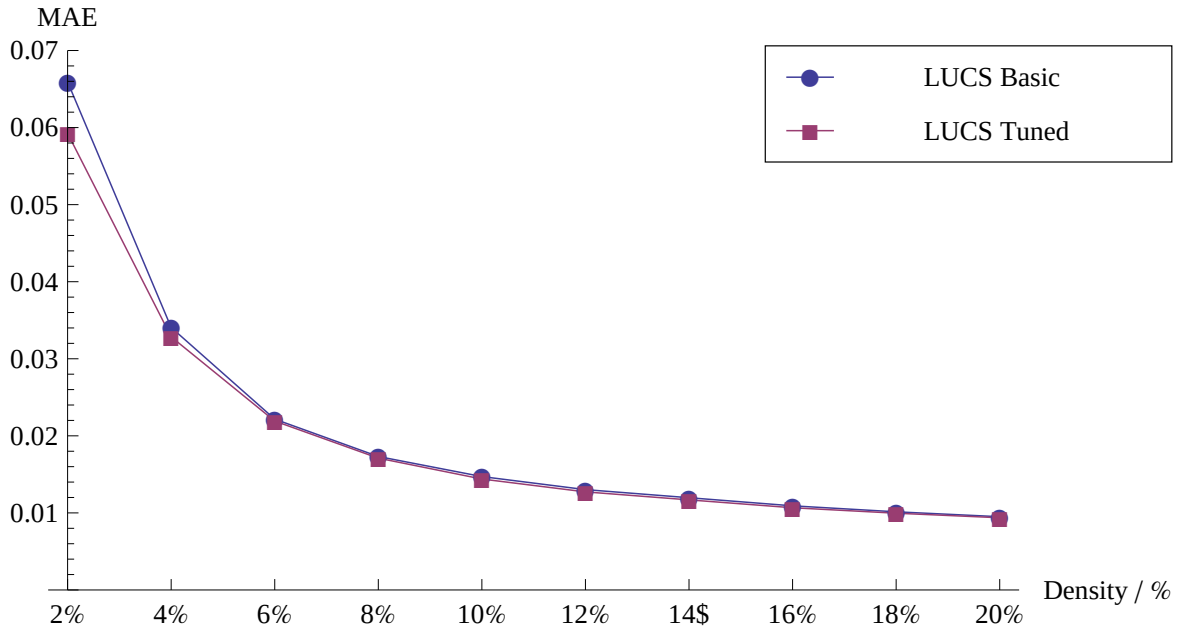
As described in Section 6.2, *CLUS* supports an arbitrary number of user, service and environment conditions clusters. The *number of clusters* is a model parameter which can be adjusted to a specific environment. This section studies how this number of clusters impacts different aspects of the prediction performance. The reader should note that the same analysis is conducted for data densities of 20% and 50% and that two different cases are considered:

1. Environment where users have different network capabilities, and
2. Environment where users have similar network capabilities.

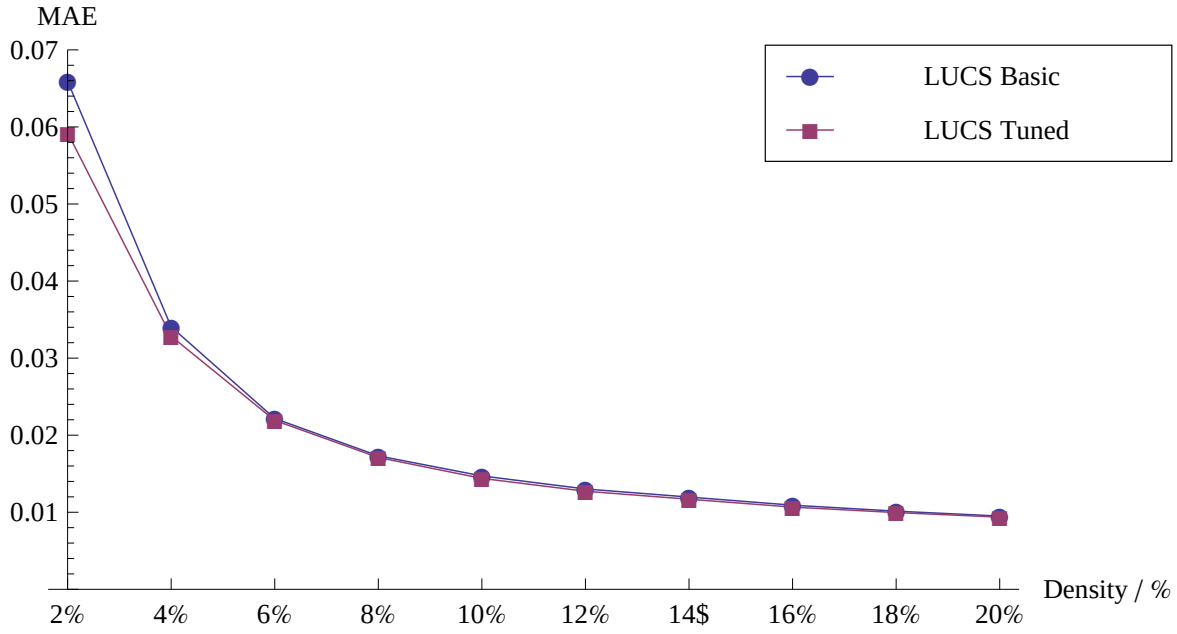
Similarly like with the density impact evaluation, all the collected data is included in the testing set for evaluation. In the evaluation process, the number of user and service clusters is varied keeping the number of environment conditions clusters constant at value of a 7. The initial value of 2 is chosen for the number of user and service clusters. Then, the reliability predictions and prediction performance measures are calculated. In the next step, the number of clusters is increased for the step value of 1 and the predictions and measures are recalculated. The procedure is repeated until the number of clusters reaches the value of 9.

7.8.1 Prediction Accuracy

The evaluation results capturing the impact of cluster number on prediction accuracy in the environment where users obtain different network capabilities are shown in Figures 7.21 and

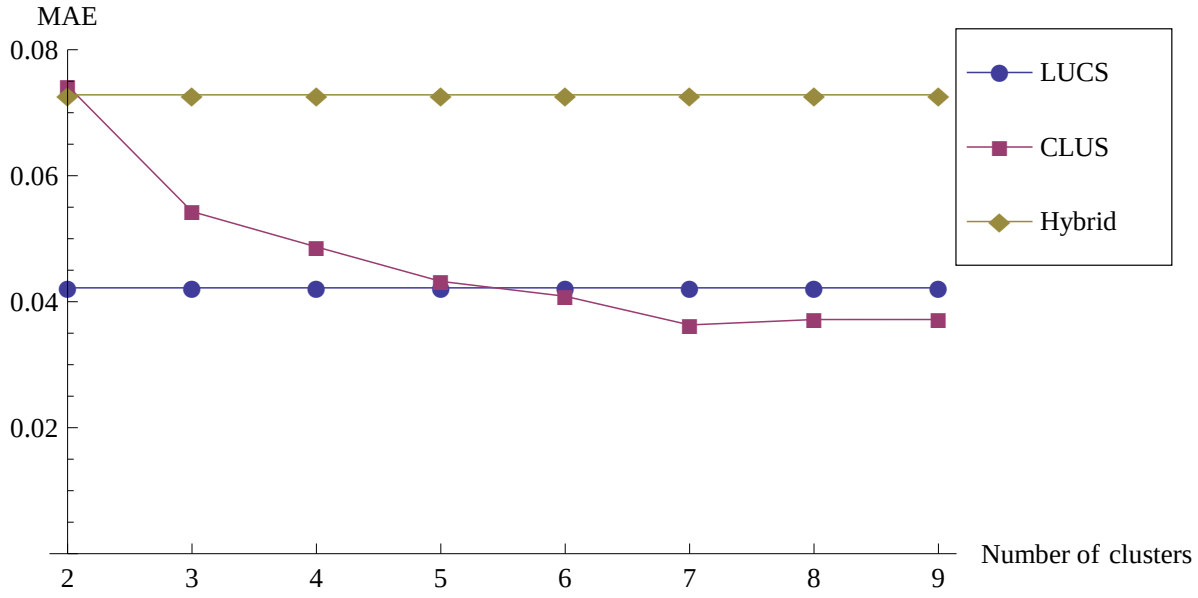


(a) MAE

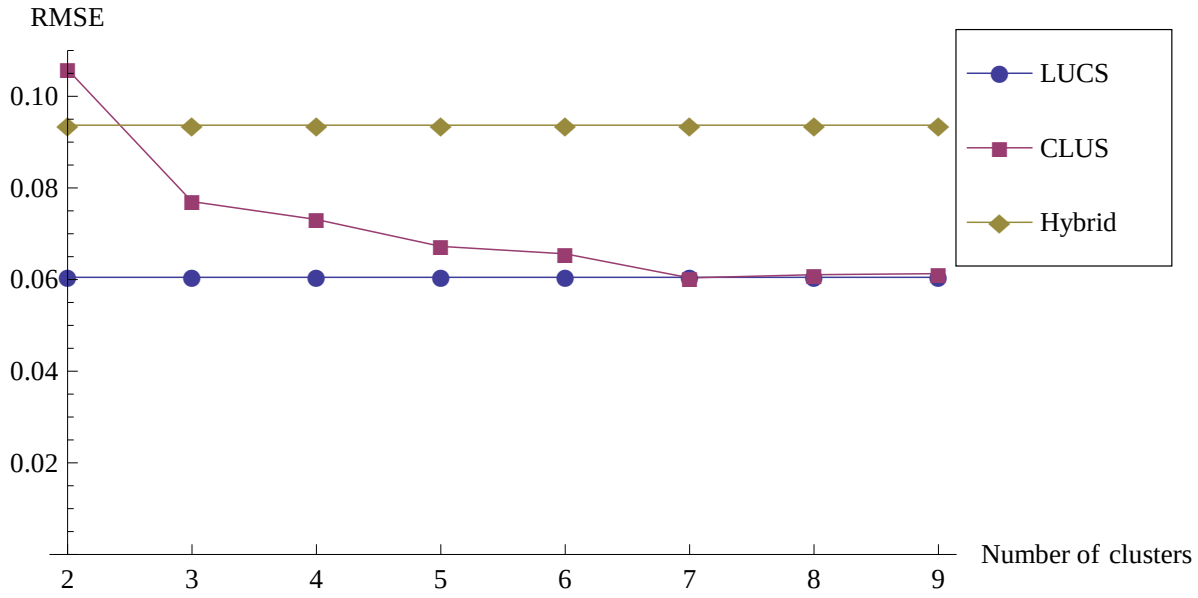


(b) RMSE

Figure 7.20: Comparison of the basic and heuristics tuned LUCS

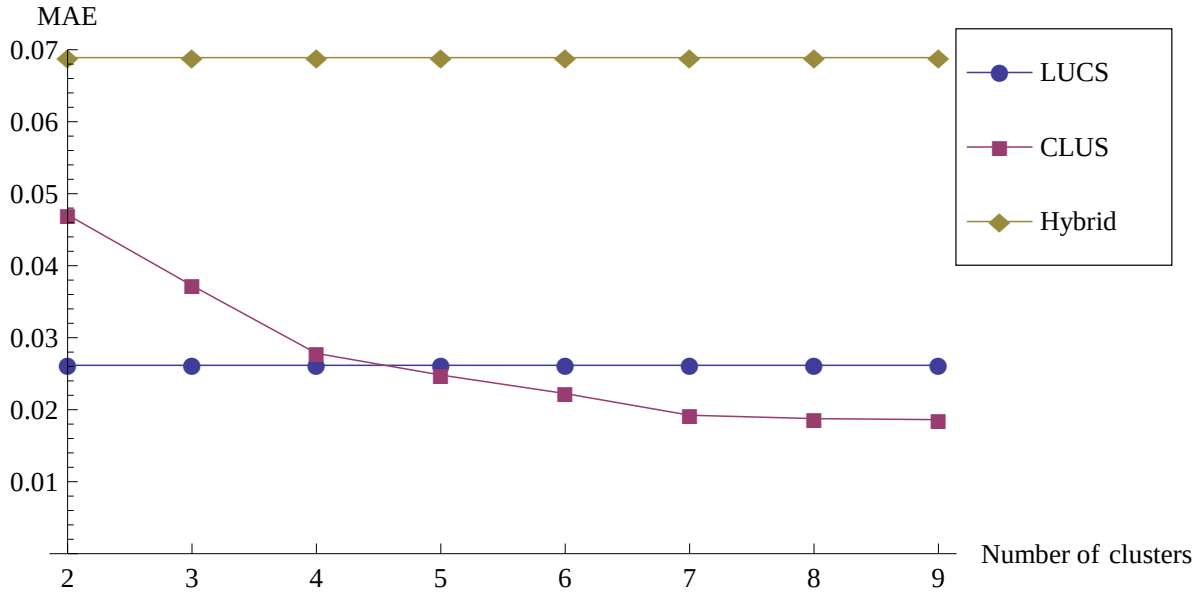


(a) MAE

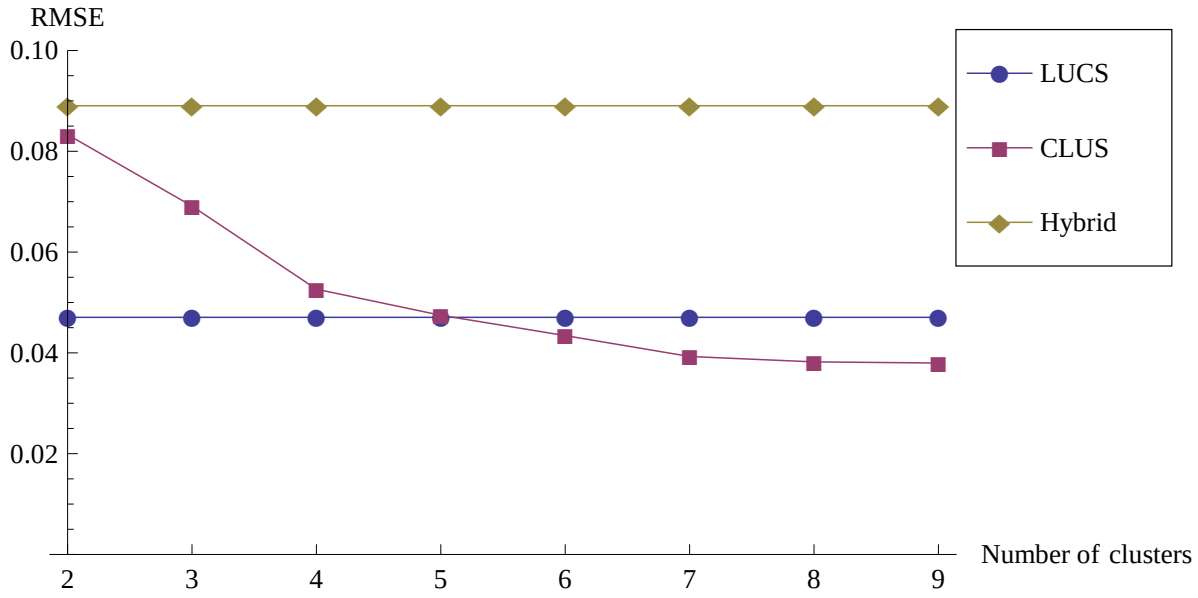


(b) RMSE

Figure 7.21: The impact of number of clusters on prediction accuracy with users having different network capabilities for the data density of 20%



(a) MAE



(b) RMSE

Figure 7.22: The impact of number of clusters on prediction accuracy with users having different network capabilities for the data density of 50%

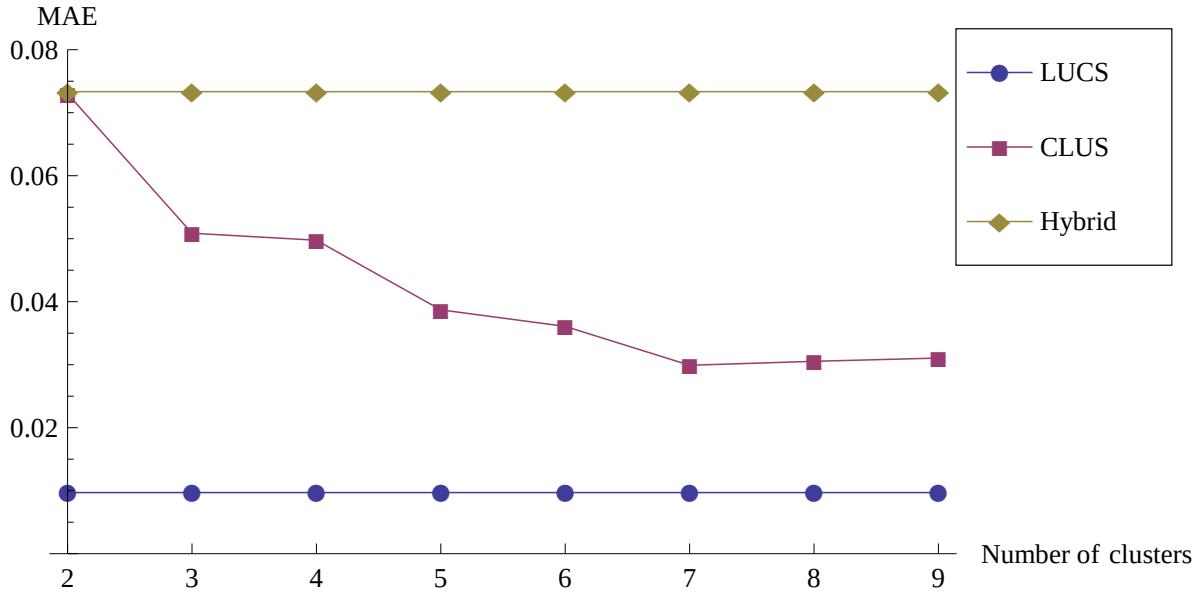
7.22.

The Figures 7.21a and 7.21b depict respected *MAE* and *RMSE* values for *LUCS*, *CLUS* and the collaborative filtering representative - the *Hybrid* approach (labels as *CF predictions* in picture) for the data density of 20%. The performance of the *LUCS* and *Hybrid* approach is not influenced by altering the number of clusters. The *LUCS* approach achieves constant *MAE* value of 0.042 and *RMSE* value of 0.060, while *Hybrid* approach achieves constant *MAE* value of 0.072 and *RMSE* value of 0.093. As can be expected, the prediction accuracy of the *CLUS* is increased as the number of clusters grows (the *RMSE* value of 0.105 for 2 clusters and the *RMSE* of 0.060 for 7 clusters). In fact, the greater number of clusters means less aggregation which improves the prediction accuracy. Note that further increase in the number of clusters after the value of 7 does not improve accuracy. This behavior can be explained by the experimental setup with 7 highly distinct service classes.

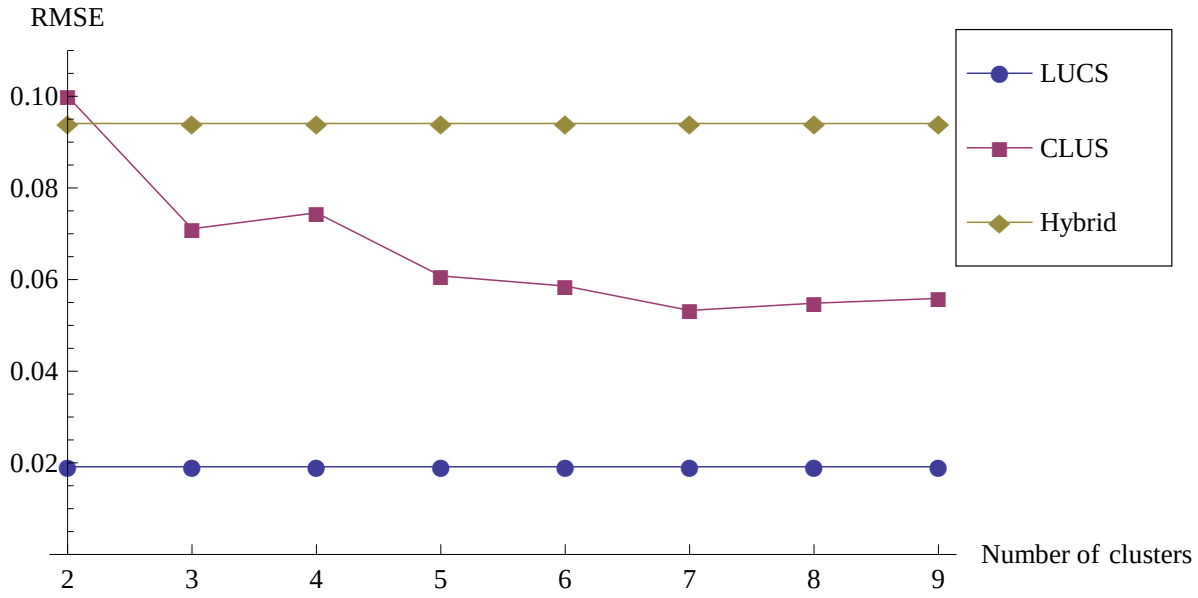
The Figures 7.22a and 7.22b depict respected *MAE* and *RMSE* values for *LUCS*, *CLUS* and the *Hybrid* approach (labels as *CF predictions* in picture) for the data density of 50%. The *LUCS* approach achieves constant *MAE* value of 0.026 and *RMSE* value of 0.047, while the *Hybrid* approach achieves constant *MAE* value of 0.068 and *RMSE* value of 0.089. Similarly, like for the density of 50%, the prediction accuracy of *CLUS* approach is increased as the number of clusters grows (the *RMSE* value of 0.083 for 2 clusters and the *RMSE* of 0.037 for 9 clusters).

The evaluation results capturing the impact of clusters number on prediction accuracy in the environment where users obtain similar network capabilities are shown in Figures 7.23 and 7.24.

Figures 7.23a and 7.23b depict respected *MAE* and *RMSE* values for *LUCS*, *CLUS* and the collaborative filtering representative - the *Hybrid* approach (labels as *CF predictions* in the picture) for the data density of 20%. The performance of the *LUCS* and *Hybrid* approach is not influenced by altering the number of clusters. The *LUCS* approach achieves constant *MAE* value of 0.010 and *RMSE* value of 0.019, while *Hybrid* approach achieves constant *MAE* value of 0.073 and *RMSE* value of 0.094. The prediction accuracy of *CLUS* approach is increased as the number of clusters grows (*RMSE* value of 0.099 for 2 clusters and *RMSE* of 0.053 for 7 clusters) and *CLUS* approach outperforms *Hybrid* approach even for the clusters number value of 3 . Similarly like for the case of an environment with users having different network capabilities, further increase in the number of clusters after the value of 7 does not improve accuracy, which is related to the experimental setup. Note, however, that in the case of an

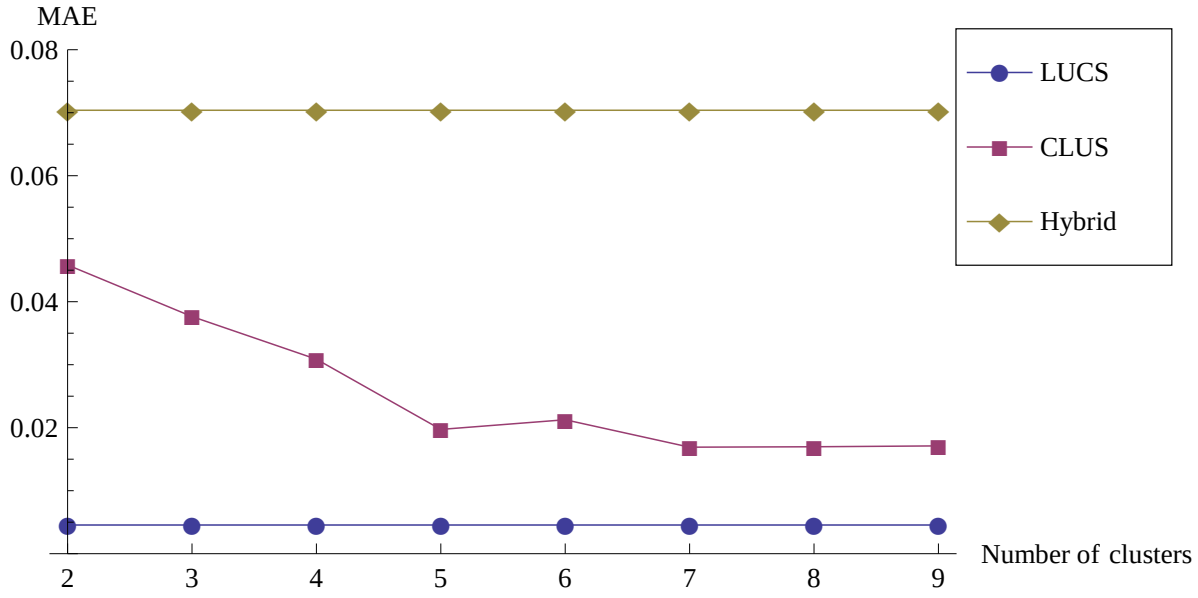


(a) MAE

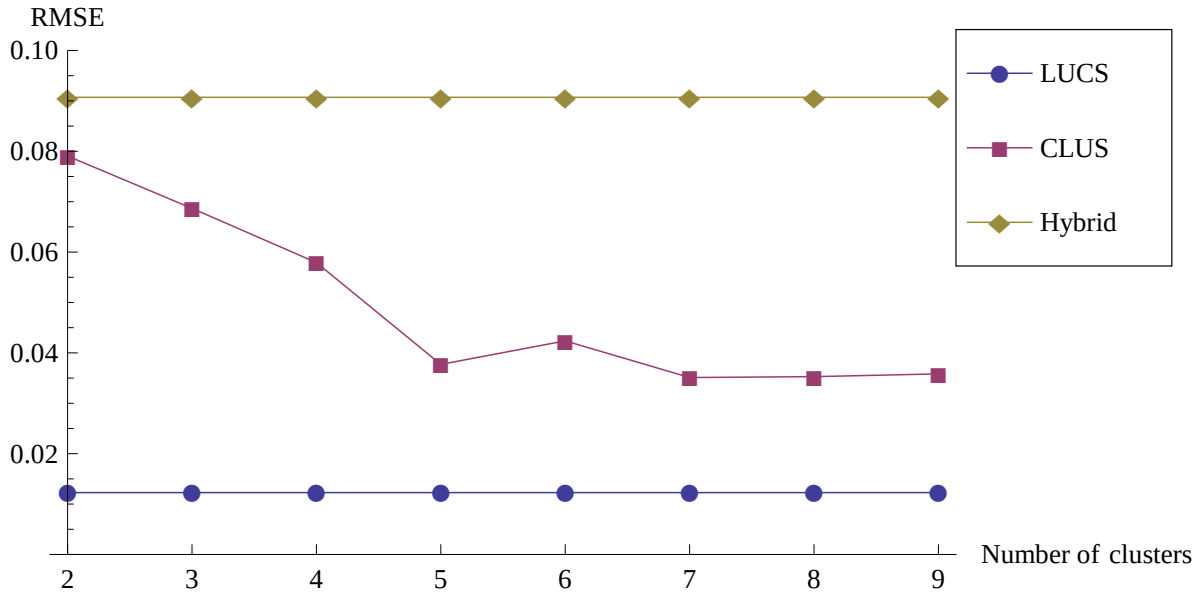


(b) RMSE

Figure 7.23: The impact of number of clusters on prediction accuracy with users having similar network capabilities for the data density of 20%



(a) MAE



(b) RMSE

Figure 7.24: The impact of number of clusters on prediction accuracy with users having similar network capabilities for the data density of 50%

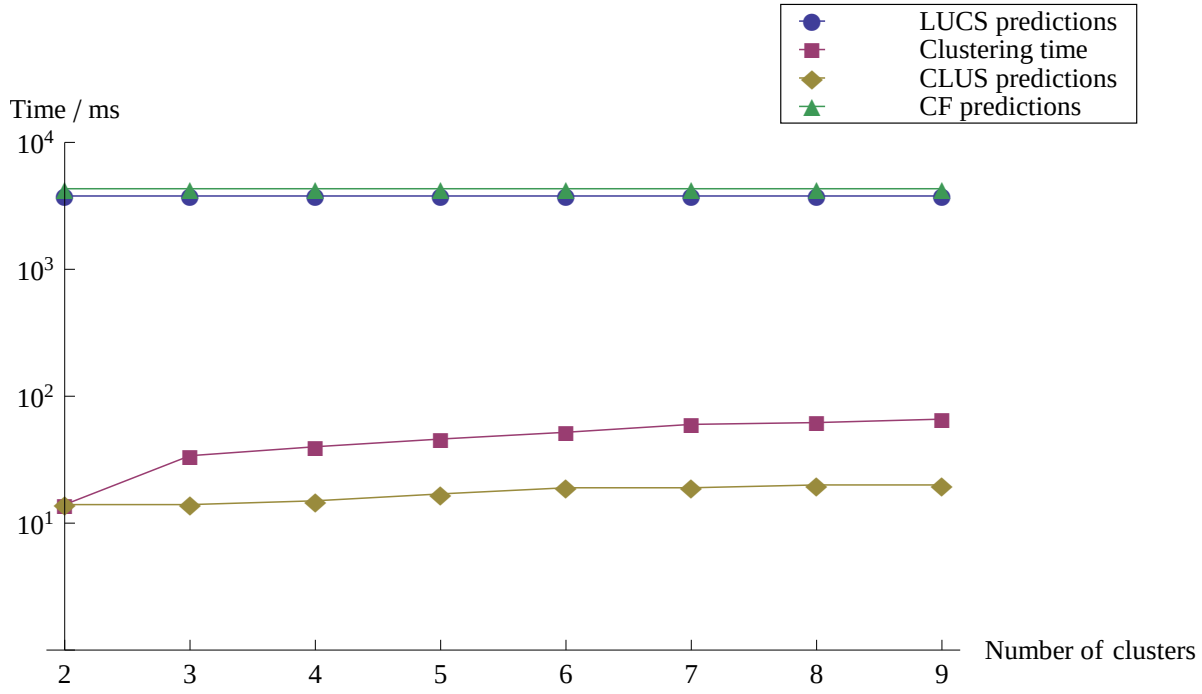
environment with users having similar network capabilities *LUCS* approach provides better accuracy than *CLUS* approach which is expected and explained (see details in Section 7.3.1).

The Figures 7.24a and 7.24b depict respective *MAE* and *RMSE* values for *LUCS*, *CLUS* and the *Hybrid* approach (labels as *CF predictions* in picture) for the data density of 50%. The *LUCS* approach achieves constant *MAE* value of 0.004 and *RMSE* value of 0.012, while the *Hybrid* approach achieves constant *MAE* value of 0.070 and *RMSE* value of 0.090. Similarly, like for the density of 50%, the prediction accuracy of the *CLUS* approach is increased as the number of clusters grows (the *RMSE* value of 0.079 for 2 clusters and the *RMSE* of 0.035 for 9 clusters). However, the *CLUS* approach can not achieve the *LUCS* approach performance in the environment with users having similar network capabilities.

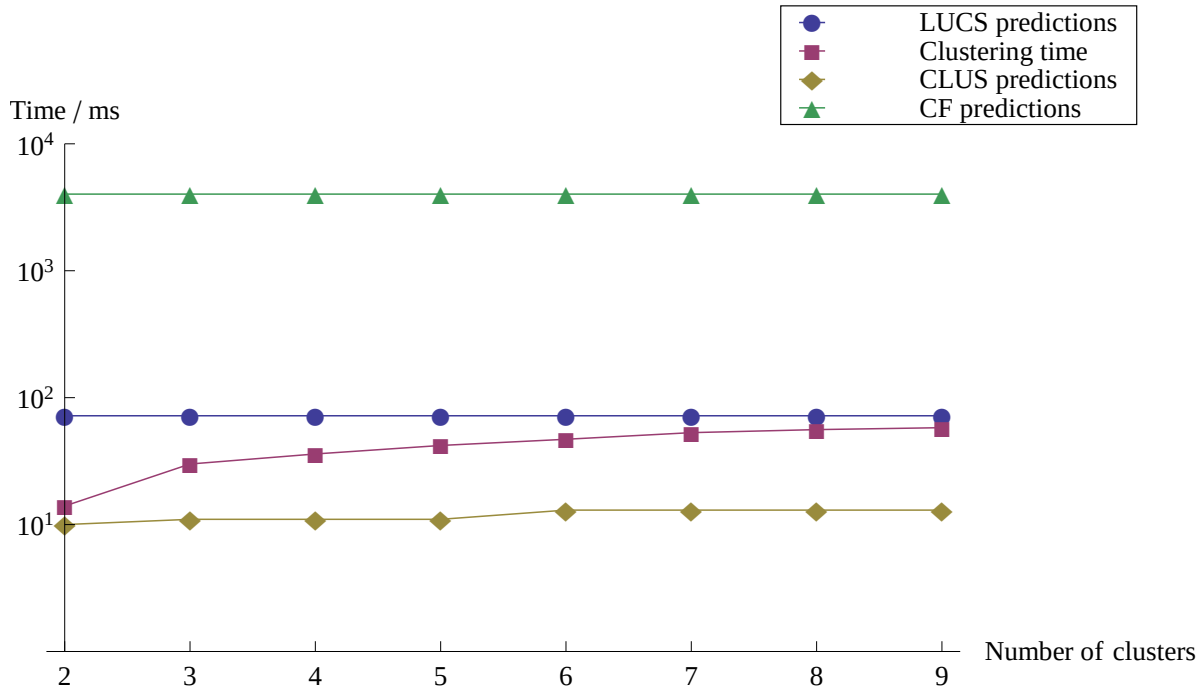
7.8.2 Computational Performance

The impact of cluster number on computational performance of the prediction is presented in Figure 7.25. Each figure depicts prediction time aggregated on the whole testing set for both *CLUS* approach and collaborative filtering representative - the *Hybrid* approach. The results are shown in the logarithmic scale. Note that figures separately capture *clustering* and *prediction* time for each phase of the *CLUS* approach. Figure 7.25a depicts prediction time for the density of 20% in relation to the number of clusters. The *LUCS* and *Hybrid* approach performance is not influenced by the number of clusters. The *LUCS* achieves the prediction time of 3729, while it takes 4367 *ms* for the *Hybrid* approach to calculate the predictions. The *CLUS* prediction time is relatively stable and ti is not influenced by altering the number of clusters. On the other hand, the clustering time increases as the number of clusters grows (from 14 ms for 2 clusters to 65 ms for 9 clusters). Knowing the computational complexity of the *K-means clustering* (see Section 7.9), this behavior is quite expected.

The Figure 7.25b shows prediction time for the density of 50% in relation to the number of clusters. The prediction time of 72 *ms* for the *LUCS* approach is significantly improved when compared to the prediction time for the density of 20%. Also, the collaborative filtering approach achieves slightly better performance with the prediction time of 3928 when compared to prediction time for the density of 20%. Note, however, that these results are expected and already explained (see Section 7.3.2 for explanation).



(a) Prediction time, density of 20%



(b) Prediction time, density of 50%

Figure 7.25: The impact of number of clusters on prediction performance

7.9 Complexity Analysis

To reiterate, the two main aims of proposed approaches, *LUCS* and *CLUS*, are (1) to improve the accuracy of reliability predictions for web services and (2) to improve the scalability of the prediction. The evaluation results presented in previous sections confirm that proposed approaches achieves both of the goals. This section provides analytic evidence, in terms of computational complexity analysis, in support of the claim that proposed approaches achieve their second aim—i.e., improved scalability over other competing collaborative filtering approaches.

The complexity of the proposed approaches is compared with the complexities of the state-of-the-art prediction models *IPCC*, *UPCC*, and *Hybrid*. Note that these complexities represent different parameterizations of the general complexity of collaborative filtering [18]. The complexity of the computations required in the *IPCC* model is:

$$O(n^2 \times m), \tag{7.10}$$

where n is the number of services and m is the number of users. Similarly, the complexity of the *UPCC* computations is:

$$O(m^2 \times n). \tag{7.11}$$

Hence, the complexity of the *Hybrid* approach is:

$$O(n^2 \times m + m^2 \times n). \tag{7.12}$$

It is important to note that real-world service based systems can have millions of users and thousands of services and thus these approaches suffer from data sparsity as well as scalability issues[1].

To address these scalability and data sparsity issues in the *LUCS* model, the *aggregation* principle is applied by grouping services based on their computational demands, request loads, and locations, and by grouping users based on their location. The complexity of the *LUCS* approach is:

$$O(n_u \times n_l \times n_c \times n_s \times (n_u + n_l + n_c + n_s)), \tag{7.13}$$

[1]Note that user-item matrix is extremely sparse for the majority of collaborative filtering tasks which makes these algorithms efficient even with such heavy computational complexities.

where n_u is the number of user locations, n_l is the number of loads, n_c is the number of service classes, and n_s is the number of service locations. While *LUCS* has a greater theoretical complexity compared to the collaborative filtering approaches, the complexity in practice will be significantly lower because the expected number of groups is significantly smaller then the number of individual users and services ($n_u, n_l, n_c, n_s \ll n, m$). In fact, as long as the number of groups is lower than the square root of the number of users, the *LUCS* complexity is lower, while further lowering the number of groups additionally improves scalability. In addition, since every *LUCS* group will include multiple data points, the *LUCS* matrix will be denser, which, in turn, leads to better quality predictions.

Regarding the computational complexity required in the *CLUS* appraoch, in general, the computational complexity required for *K-means clustering* is:

$$O(i \times c \times d \times n), \tag{7.14}$$

where i is the number of iterations performed by the procedure, c is the number of clusters, n is the number of vectors to be clustered and d is the dimensionality of vectors [26, 27]. Although the theoretical worst case may take exponential time for the algorithm to converge [207], in practical cases with data points representing service reliabilities, the algorithm converges very quickly. However, in the K-means clustering implementation used in the evaluation process, the number of iterations is limited to the value of 10.

In the data clustering phase of the *CLUS* approach K-means clustering procedure is performed three times. First, different time windows are clustered, which requires the computational complexity of:

$$O(i \times |E| \times |W| \times 1), \tag{7.15}$$

where i is the number of iterations, $|E|$ is the number of environment conditions clusters and $|W|$ is the number of time windows the day is divided to. Note that the dimension of the vectors representing time windows is 1. Then, users and services are separately clustered. The computational complexity required in user clustering is:

$$O(i \times |U| \times |E| \times m), \tag{7.16}$$

where $|U|$ is the number of user clusters, and m is the number of users.

Similarly, the computational complexity of services clustering takes:

$$O(i \times \ |S| \times |E| \times n), \tag{7.17}$$

where $|S|$ is the number of service clusters and n is the number of services.

The values i, $|W|$ and $|E|$ are constant and do not impact computational complexity. The model assumes that the number of clusters is relatively small when compared to the number of users and services ($|U|, |S| << n, m$). Hence, the practical case computational complexity in the *CLUS* approach requires:

$$O(m + n). \tag{7.18}$$

The presented analysis of complexity strongly supports claims in favor of the proposed approaches better scalability.

7.10 Evaluation Summary

This section aims to summarize all the experiments and evaluation results in one place because of the very detailed and exhaustive evaluation chapter. The following paragraphs briefly overview different evaluation aspects of the proposed models that were analyzed in the previous sections.

Prediction accuracy at the density of 25%

Section 7.2 studies the prediction accuracy for each competing approach having a constant data density of 25%. Also, two different environments are considered, the one where users obtain similar network performance and the one where network capabilities differ. For the environment where users obtain similar network capabilities, the *LUCS* approach provides best prediction accuracy, having at least 81% lower *RMSE* value and at least 88% lower *MAE* value than the collaborative filtering based *Hybrid* approach (see details in Table 7.3). In the case of the environment where users have different network performance, the *CLUS* approach achieves the best prediction accuracy, having at least 44% lower *MAE* and at least 37% lower *MAE* value than the *Hybrid* approach (for details see Table 7.4). In addition, Figures 7.1–7.5 reveal the primary cause of variance in predictions produced by different prediction approaches.

The impact of data density

Section 7.3 analyses the impact of the data density on both prediction accuracy and computational performance. In general, the evaluation results suggest that the data density highly impacts prediction accuracy and performance.

While analyzing the impact on prediction accuracy, three different environments are considered: a dynamic environment (meaning the fluctuations in the service load are present) where users have similar network capabilities, a dynamic environment where users have different network capabilities, and a static environment (without service load intensity) where users have similar network capabilities. In a dynamic environment with users having similar network performance, the *LUCS* approach provides the best prediction accuracy (see details in Figures 7.6a–7.6b). In a dynamic environment where users have different network performance, the *CLUS* approach produces the best prediction accuracy (see details in Figures 7.7a–7.7b). In a static environment, expectedly, collaborative filtering based approaches provide better prediction accuracy for high data density, while *LUCS* and *CLUS* approaches provide better prediction for low data density (for details see Figures 7.8a–7.8b).

Regarding computational performance of the prediction, the *CLUS* approach provides the shortest and the most stable *prediction time*, almost independent of the data density. On the other hand, *LUCS* approach prediction time decreases as the data density increases, and it provides better prediction time when compared to the collaborative filtering based approaches on average (see details in Figures 7.9a–7.9b).

The significance of *service load* and *service class*

Section 7.4 studies the distribution of prediction *RMSE* error over different service loads and classes for each competing approach. The analyses is separately conducted for the dynamic environments with similar and different users network capabilities and for different data densities, density of 20% and 50%.

Regarding the *RMSE* distribution over different service loads, for both environments and both densities, *LUCS* and *CLUS* approaches obtain relatively stable prediction error. On the contrary, the collaborative filtering based approaches produce best predictions at average loads, as the load increases or decreases, the prediction error is drastically increased. The *RMSE* distributions over different loads for different environments and data densities can be seen in Figures 7.10a, 7.10b, 7.11a and 7.11b.

Regarding the *RMSE* distribution over different service classes, for both considered environments and densities, all competing approaches achieve better prediction accuracy for heavier service classes, while the prediction error is highly increased for classes *Class 6* and *Class 7*, which are considered to be *gray sheep*. However, *LUCS* and *CLUS* approaches significantly outperform collaborative filtering approaches with lower prediction error for each service class as can be seen in Figures 7.12a, 7.12b, 7.13a and 7.13b.

The importance of each individual *LUCS*'s input parameter

Section 7.5 asses the importance of each individual *LUCS*'s input parameter on prediction accuracy. The importance is analyzed in two different tests. In the first test, the impact of each individual parameter available on *LUCS*'s prediction accuracy is studied. The second test assesses how lack of each individual parameter influences *LUCS*'s prediction accuracy. The amount of records that have a single parameter, either available for the first test, or missing for the second test, is varied from 0% to 100% of records. Each test is separately conducted for the dynamic environments with similar and different users network capabilities and for different data densities, density of 5% and 25%.

The results for the first test, where a single parameter is available for variable percentage of records show that parameter c, service class, is very important for prediction. In the case even only a 40% of records from the sample have parameter c available (and all others missing), such a limited *LUCS* approach outperforms collaborative filtering for lower data density. However, for higher data densities, this is not possible. Any other parameter in isolation can not be used to predict the reliability better than the collaborative filtering based approaches. The details can be seen in Figures 7.14a, 7.14b, 7.15a and 7.15b.

In the second test, where a single parameter is missing for variable percentage of records, the parameter service class c also proves to be the most important for the *LUCS* approach prediction accuracy. The model can tolerate up to 50% of records missing this parameter for lower densities and up to 10% of records missing it for higher densities. The lack of parameter service load l significantly degrades prediction accuracy. However, the *LUCS* approach manifests considerable tolerance to lack of u and s parameters. The details can be seen in Figures 7.16a, 7.16b, 7.17a and 7.17b.

The sensitivity of *LUCS*'s groupings

Section 7.6 studies the impact of false service invocations groupings on *LUCS* approach prediction accuracy. In order to asses *LUCS*'s sensitivity on false groupings, the *fault injection* technique is used. Hence, it is allowed for the service to be incorrectly grouped into the neighborhood class. For instance, the service belonging to the *class 4* can be incorrectly placed into the *class 3* or *class 5*. Similarly, the service load can be falsely classified into the neighborhood load, while parameters user location and service location are considered to be resilient to false groupings. The impact of false groupings is analyzed by computing *RMSE* varying the percentage of incorrectly classified service invocations from 0% to 100%. The same analyses is separately conducted for the dynamic environments with similar and different users network capabilities and for different data densities, density of 5% and 25%. The evaluation results show that *LUCS* tolerates false groupings for the lower densities for even high percentage of incorrectly grouped records. For the higher densities, *LUCS* can tolerate up to 30% incorrectly grouped records. The details can be seen in Figures 7.18a, 7.18b, 7.19a and 7.19b.

The heuristics for *LUCS* model's parameters α, β and γ

Section 7.7 provides the heuristics for *LUCS* model's parameters adjustment to the specific environment. The evaluation results presented so far have used an equal value of 0.25 for each parameter α, β and γ. These parameters are used in the Equation 5.32 to determine the impacts of different sets of similar entities. However, these values are not optimal for the given environment. In order to determine better coefficient values, heuristics analyzes the *RMSE* of each impact for each specific record in the sample, and determines the coefficients so the impacts with higher error contribute less and impacts with lower error contribute more. To evaluate the heuristics, the heuristics-tunned *LUCS* predictions are compared to the raw *LUCS* predictions ($\alpha = \beta = \gamma = 0.25$). The results show that the heuristics further improved the prediction model, specially for the lower data densities (see details in Figures 7.20a and 7.20b).

The impact of *CLUS*'s number of clusters

Section 7.8 assess how *CLUS*'s number of clusters effects both prediction accuracy and computational performance. To inspect the effect of the number of clusters, the prediction error and time it takes to produce the predictions are computed while varying the number of user and service clusters from the value of 2 to the value of 9, and keeping the number of environment

conditions clusters constant at the value of 7.

Regarding prediction accuracy, the assessment is separately conducted for the dynamic environments with similar and different users network capabilities and different data densities, density of 20% and 50%. The evaluation results for the environment with users with different network performance show that even with the small number of clusters the *CLUS* approach outperforms collaborative filtering based approaches. In fact, the *CLUS* with the higher number of clusters outperforms the *LUCS* approach as well (see details in Figures 7.21a, 7.21b, 7.22a and 7.22b). In the environment where users have similar network capabilities, the *CLUS* approach also outperforms collaborative filtering based approaches. However, in this environment it can not provide better predictions than *LUCS* approach (see details in Figures 7.23a, 7.23b, 7.24a and 7.24b).

Regarding computational performance of the prediction, both *clustering* and *prediction* time of the *CLUS* approach increase as the number of clusters grows as can be seen in Figures 7.25a and 7.25b. Knowing the complexity of the *K-means* clustering, obtained results are quite expected.

Complexity analysis

Section 7.9 provides the analytical proof, as the computational complexity analysis for each competing approach, to argument the claims of better scalability of *LUCS* and *CLUS* approaches. Hence, the complexity required in the collaborative filtering representative – the *Hybrid* approach is:

$$O(n^2 \times m + m^2 \times n), \tag{7.19}$$

where n is the number of services while m is the number of users. The complexity required in *LUCS* approach is:

$$O(n_u \times n_l \times n_c \times n_s \times (n_u + n_l + n_c + n_s)), \tag{7.20}$$

where n_u is the number of user locations, n_l is the number of loads, n_c is the number of service classes, and n_s is the number of service locations. Note, however, that the number of groups is considerably smaller than the number of individual users and services ($n_u, n_l, n_c, n_s \ll n, m$). The practical case computational complexity in the *CLUS* approach requires:

$$O(m + n), \tag{7.21}$$

where n is the number of services while m is the number of users.

It is obvious from the presented complexity analysis that the proposed approaches provide better scalability when compared to the collaborative filtering state-of-the-art.

Chapter 8

Conclusion

This dissertation is focused on reliability modeling of consumer applications in Consumer Computing. Consumer applications are component based applications created by consumers out of existing applications as basic building blocks. Consumers compose the existing applications into a more complex composite applications to support the additional functionality. In order to assess the reliability of the component-based system, the reliability of each comprising component needs to be known. Although consumer applications provide functionality through a simple consumer intuitive GUI interface, they often contain the dynamic part of the code which requires information retrieval or data processing over the Internet. Based on such dynamic characteristics, consumer applications can be considered as a dynamic software artifacts that provide their functionality over the Internet such as services. However, modeling a reliability of services proves to be a very difficult task due to perceived reliability fluctuations caused by the variability of service invocation context parameters.

The service reliability can be defined as a probability that the service invocation will be completed successfully, meaning that the invocation response is retrieved successfully. According to the adopted definition, the service reliability can be computed from the past invocations data sample as the ratio of number of successful service invocations against the number of total performed invocations. The service reliability value computed in this manner is highly dependent on the quality and the quantity of the past invocations sample. However, gaining a comprehensive past invocations sample appears to be a very difficult task in practice. The insight into the solution of this problem is to collect as much as possible feedback from consumers and service providers, and to utilize prediction methods in order to estimate the reliability for the invocation contexts that are lack of a sufficient number of records in the past invocations sample.

The most successful existing approaches for prediction of services reliability are based on collaborative filtering technique which is commonly used in contemporary recommendation systems on the Internet. Although the existing state-of-art approaches achieve promising prediction accuracy and performance, they demonstrate some potentially serious disadvantages regarding scalability and accuracy in dynamic environments. First, having a significantly large number of services (underlying consumer applications) and millions of consumers in Consumer Computing, collaborative filtering approaches do not scale and the real-time prediction performance is questionable. Second, the collaborative filtering produces accurate predictions in static environments in which the collected data records stay up to date for a reasonable long period of time such as movie ratings or product recommendations. However, services are deployed on the Internet which a highly dynamic environment that considerably changes on a daily basis. For instance, the same consumer might experience quite different reliability properties while using the same service at different periods of the same day.

In order to address the main disadvantages of existing state-of-the-art approaches, new prediction models are proposed as part of this dissertation. The first proposed approach, *LUCS*, estimates the reliability for an ongoing request using the past invocations data sample. The model produces predictions based on following parameters values at the time of the invocation: user's location, service's location, service load and service class (describing service internals regarding its computational complexity). The *LUCS* approach improves the existing collaborative filtering approaches by extending the model with parameters service load and class and grouping the service invocations records according to the model parameters, and performing collaborative filtering by discovering similar entities regarding each model parameter. The final prediction is computed as a linear combination of different parameters impacts.

The *LUCS* approach manifests considerable dependence on the explicit availability of model parameters and produces significantly less accurate predictions without correct input parameters values. In order to address this drawback, a more flexible *CLUS* approach is proposed. The *CLUS* approach also uses the past invocations sample to estimate the reliability for an ongoing request. To improve the accuracy of the prediction, the *CLUS* approach introduces environment–specific parameters that describe current load conditions in the system. On the other hand, to improve the scalability, the *CLUS* approach reduces the redundant data by grouping users and services into respected user and service clusters according to their reliability performance using K-means clustering algorithm.

In order to evaluate the proposed models, series of experiments were conducted on services deployed in different regions of the Amazon EC2 Cloud. During experiments, different users were simulated by placing instances of *loadUI* tool running as distributed agents in different regions of the Amazon EC2 Cloud, waiting for the tasks to be delivered and executed. Different environment–specific parameters were introduced by creating different load generators which are also supported by the *loadUI* tool. Also, a special environment comprising users with different network capabilities was simulated by using special tool which supports limitation of network adapter's performance. The reliability data collected during the experiments was used to compare the prediction accuracy and performance of the proposed models to the state-of-the-art approaches: *UPCC*, *IPCC* and the *Hybrid* approach. The evaluation results strongly supports claims of better prediction *accuracy* and *performance* of the proposed models when compared to the state-of-the-art.

Regarding prediction accuracy, in a dynamic environment where users obtain similar network capabilities, the *LUCS* approach achieves the best prediction accuracy with at least 81% lower *RMSE* value compared to the collaborative filtering based *Hybrid* approach. In addition, in this environment, the *CLUS* approach also outperforms collaborative filtering approaches having at least 48% lower *RMSE* value than the *Hybrid* approach. The evaluation results for the environment where users obtain different network capabilities show that *CLUS* approach provides the best accuracy among the competing approaches having at least 44% lower *RMSE* value compared to the *Hybrid* approach. This results is quite expected since *LUCS* approach considers only user's location as a user–specific parameter. Expectedly, the user location is not a significant parameter while determining service reliability on the broadband Internet. Still, *LUCS* approach outperforms collaborative filtering based approaches in this environment too, with at least 37% lower *RMSE* value than the *Hybrid* approach.

From the aspect of the prediction computational performance which is closely related to the scalability, the evaluation results confirm the analytical proofs, which are presented through the analyses of complexity for each competing approach, in favor of better scalability of the proposed approaches. According to the obtained results, the *CLUS* approach provides the quickest and most stable prediction time almost independent of the data density. More specifically, the *CLUS* approach reduces the prediction time for at least two orders of magnitude compared to the collaborative filtering representative - the *Hybrid* approach. The time that takes for *LUCS* approach to produce the predictions is constantly decreasing as the data density increases and it

produces predictions more quickly than the *Hybrid* approach on average.

According to the presented evaluation results and brought analyses of complexity, the proposed approaches successfully implement the main goals of this dissertation which are the addressing the drawbacks and improving the existing state-of-the-art collaborative filtering based approaches regarding prediction accuracy and scalability. In fact, the proposed approaches offer the additional benefit in their flexibility which is clearly reflected in a balanced trade-off between accuracy and scalability. For instance, when considering *LUCS* model, by increasing the number of classes and loads groups, more accurate predictions are produced. By contrast, by decreasing the number of classes and loads groups, more scalable approach is achieved. Similarly, the increase of number of clusters in the *CLUS* approach results in a better prediction accuracy. On the other hand, by reducing the number of clusters, more scalable prediction approach is produced. With such characteristics, these flexible approaches can be applied in different service-oriented environments.

The main motivation for the design of prediction models presented in this dissertation is their appliance and need in Consumer Computing for predicting the reliability of consumer applications. Hence, as part of this dissertations, the architecture of the reliability prediction system in Consumer Computing is designed. The architecture of the reliability prediction system integrates the proposed prediction methods in the Consumer Computing environment. As part of the proposed architecture, the consumer intuitive mechanism is designed as a dedicated consumer assistant application called *Geppeto ReliabilityOptimizeMe*. The role of the *Geppeto ReliabilityOptimizeMe* widget is to provide consumers help and offer assistance about the reliability of potential selection candidates while creating their consumer applications.

Bibliography

[1] CCL, "Consumer computing lab." http://ccl.fer.hr/, 2013.

[2] T. Berners-Lee, "Programming ability is the new digital divide." http://www.computerworld.co.nz/article/452521/programming_ability_new_digital_divide_berners-lee/, 2013.

[3] M. R. Lyu, ed., *Handbook of software reliability engineering*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.

[4] J. D. Musa, A. Iannino, and K. Okumoto, *Software reliability: measurement, prediction, application (professional ed.)*. New York, NY, USA: McGraw-Hill, Inc., 1990.

[5] Z. Jelinski and P. Moranda., "Software reliability research.," in *Proc. of the Statistical Methods for the Evaluation of Computer System Performance*, 1972.

[6] M. R. Lyu, "Software reliability engineering: A roadmap," in *2007 Future of Software Engineering*, (Washington, DC, USA), IEEE Computer Society, 2007.

[7] L. H. Putnam and W. Myers, *Measures for Excellence: Reliable Software on Time, within Budget*. Prentice Hall Professional Technical Reference, 1991.

[8] M. Friedman and P. Tran, "Reliability techniques for combined hardware/software systems," in *Reliability and Maintainability Symposium, 1992. Proceedings., Annual*, 1992.

[9] A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood, "Evaluation of competing software reliability predictions," *IEEE Trans. Softw. Eng.*, 1986.

[10] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early prediction of software component reliability," in *Proceedings of the 30th international conference on Software engineering*, 2008.

[11] L. Cheung, I. Krka, L. Golubchik, and N. Medvidovic, "Architecture-level reliability prediction of concurrent systems," in *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*, 2012.

[12] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 311–327, 2004.

[13] L. Cheung, L. Golubchik, and F. Sha, "A study of web services performance prediction: A client's perspective," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, 2011.

[14] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, UAI'98, (San Francisco, CA, USA), pp. 43–52, Morgan Kaufmann Publishers Inc., 1998.

[15] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, (New York, NY, USA), ACM, 2001.

[16] Z. Zheng and M. R. Lyu, "Collaborative reliability prediction of service-oriented systems," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, (New York, NY, USA), ACM, 2010.

[17] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Qos-aware web service recommendation by collaborative filtering," *IEEE Transactions on Services Computing*, 2011.

[18] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Adv. in Artif. Intell.*, vol. 2009.

[19] A. S. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: scalable online collaborative filtering," in *Proceedings of the 16th international conference on World Wide Web*, WWW '07, (New York, NY, USA), pp. 271–280, ACM, 2007.

[20] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, pp. 331–370, Nov. 2002.

[21] H. Ma, I. King, and M. R. Lyu, "Effective missing data prediction for collaborative filtering," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), ACM, 2007.

[22] H. Guan, H. Li, and M. Guo, "Semi-sparse algorithm based on multi-layer optimization for recommendation system," in *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores*, (New York, NY, USA), ACM, 2012.

[23] C. Wei, W. Hsu, and M. L. Lee, "A unified framework for recommendations based on quaternary semantic analysis," in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, (New York, NY, USA), ACM, 2011.

[24] M. Silic, G. Delac, I. Krka, and S. Srbljic, "Scalable and accurate prediction of availability of atomic web services," *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, p. 1, 2013.

[25] M. Silic, G. Delac, and S. Srbljic, "Prediction of atomic web services reliability based on k-means clustering," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, (New York, NY, USA), pp. 70–80, ACM, 2013.

[26] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (L. M. L. Cam and J. Neyman, eds.), vol. 1, pp. 281–297, University of California Press, 1967.

[27] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[28] D. Skvorc, *Consumer programming*. PhD thesis, Faculty of Electrical Engineering and Computing, University of Zagreb, 2010.

[29] M. Popovic, *Consumer program synchronization*. PhD thesis, Faculty of Electrical Engineering and Computing, University of Zagreb, 2011.

[30] I. Budiselic, *Component recommendation for development of composite consumer applications*. PhD thesis, Faculty of Electrical Engineering and Computing, University of Zagreb, 2014.

[31] K. Vladimir, *Peer tutoring in consumer computing*. PhD thesis, Faculty of Electrical Engineering and Computing, University of Zagreb, 2013.

[32] G. Delac, *Reliability management of composite consumer applications*. PhD thesis, Faculty of Electrical Engineering and Computing, University of Zagreb, 2014.

[33] C. C. Lab, "Geppeto." http://161.53.65.222:8080/geppeto/index.html, 2013.

[34] Wikipedia, "Google play." http://en.wikipedia.org/wiki/Google_Play/, 2012.

[35] Wikipedia, "App store (ios)." https://en.wikipedia.org/wiki/App_Store_(iOS)/, 2012.

[36] Nielsen, "Average number of apps per smartphone now 41." http://internet2go.net/news/mobile-platforms/nielsen-average-number-apps-smartphone-now-41/, 2012.

[37] C. B. Lockard and M. Wolf, "Occupational employment projections to 2020," *Monthly Lab. Rev.*, vol. 135, p. 84, 2012.

[38] G. Inc., "Google glass." http://www.google.com/glass/start/, 2013.

[39] N. Milanovic and M. Malek, "Current solutions for web service composition," *Internet Computing, IEEE*, vol. 8, no. 6, pp. 51–59, 2004.

[40] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[41] W3C, "Html: The markup language." http://www.w3.org/TR/html-markup/, May 2013.

[42] A. Cypher, D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers, and A. Turransky, eds., *Watch what I do: programming by demonstration*. Cambridge, MA, USA: MIT Press, 1993.

[43] D. C. Halbert, *Programming by example*. PhD thesis, University of California, Berkeley, 1984.

[44] S. Bragg and C. Driskill, "Diagrammatic-graphical programming languages and dod-std-2167a," in *AUTOTESTCON '94. IEEE Systems Readiness Technology Conference. 'Cost Effective Support Into the Next Century', Conference Proceedings.*, pp. 211–220, 1994.

[45] R. Wieringa, "A survey of structured and object-oriented software specification methods and techniques," *ACM Comput. Surv.*, vol. 30, pp. 459–527, Dec. 1998.

[46] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[47] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR, 2004.

[48] T. Velte, A. Velte, and R. Elsenpeter, *Cloud computing, a practical approach*. McGraw-Hill, Inc., 2009.

[49] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, 2004.

[50] K. Goševa-Popstojanova and K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems," *Performance Evaluation*, vol. 45, no. 2, pp. 179–204, 2001.

[51] V. Cortellessa and V. Grassi, "Reliability modeling and analysis of service-oriented architectures," in *Test and Analysis of Web Services*, pp. 339–362, Springer, 2007.

[52] W3C, "Web services description language 1.1." http://www.w3.org/TR/wsdl/, March 2001.

[53] OASIS, "Web services business process execution language version 2.0," 2007. OASIS Standard.

[54] D. Skrobo, A. Milanovic, and S. Srbljic, "Performance evaluation of program translation in service-oriented architectures," in *Proceedings of the International conference on Networking and Services*, (Washington, DC, USA), IEEE Computer Society, 2006.

[55] S. Sinisa, S. Dejan, and S. Daniel, "Programming language design for event-driven service composition.," *AUTOMATIKA*, 2011.

[56] W.-L. Wang, D. Pan, and M.-H. Chen, "Architecture-based software reliability modeling," *J. Syst. Softw.*, 2006.

[57] B. Zhou, K. Yin, S. Zhang, H. Jiang, and A. J. Kavs, "A tree-based reliability model for composite web service with common-cause failures," in *Proceedings of the 5th international conference on Advances in Grid and Pervasive Computing*, (Berlin, Heidelberg), Springer-Verlag, 2010.

[58] V. Grassi and S. Patella, "Reliability prediction for service-oriented computing environments," *IEEE Internet Computing*, 2006.

[59] W. T. Tsai, D. Zhang, Y. Chen, H. Huang, R. Paul, and N. Liao, "A software reliability model for web services," in *The 8th IASTED International Conference on Software Engineering and Applications*, 2004.

[60] J. Ma and H.-p. Chen, "A reliability evaluation framework on composite web service," in *Proceedings of the 2008 IEEE International Symposium on Service-Oriented System Engineering*, (Washington, DC, USA), IEEE Computer Society, 2008.

[61] F. Mahdian, V. Rafe, R. Rafeh, and A. T. Rahmani, "Modeling fault tolerant services in service-oriented architecture," in *Third IEEE International Symposium on Theoretical Aspects of Software Engineering 2009*, (Washington, DC, USA), IEEE Computer Society, 2009.

[62] B. Li, X. Fan, Y. Zhou, and Z. Su, "Evaluating the reliability of web services based on bpel code structure analysis and run-time information capture," in *Asia Pacific Software Engineering Conference 2010*, (Washington, DC, USA), IEEE Computer Society, 2010.

[63] L. Coppolino, L. Romano, and V. Vianello, "Security engineering of soa applications via reliability patterns.," *JSEA*, 2011.

[64] G. Candea and A. Fox, "Crash-only software," in *Proceedings of the 9th conference on Hot Topics in Operating Systems - Volume 9*, HOTOS'03, (Berkeley, CA, USA), pp. 12–12, USENIX Association, 2003.

[65] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and T. Yimwadsana, "Predictability of web-server traffic congestion," *Web Content Caching and Distribution, International Workshop on*, 2005.

[66] M. Andreolini and S. Casolari, "Load prediction models in web-based systems," in *International conference on Performance evaluation methodolgies and tools*, (New York, NY, USA), ACM, 2006.

[67] Y. Wang, W. M. Lively, and D. B. Simmons, "Web software traffic characteristics and failure prediction model selection," *J. Comp. Methods in Sci. and Eng.*, vol. 9, pp. 23–33, Apr. 2009.

[68] A. Klein, F. Ishikawa, and S. Honiden, "Towards network-aware service composition in the cloud," in *Proceedings of the 21st international conference on World Wide Web*, WWW '12, (New York, NY, USA), pp. 959–968, ACM, 2012.

[69] N. B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny, "Qos-aware service composition in dynamic service oriented environments," in *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, (New York, NY, USA), pp. 7:1–7:20, Springer-Verlag New York, Inc., 2009.

[70] H. Ghanbari, C. Barna, M. Litoiu, M. Woodside, T. Zheng, J. Wong, and G. Iszlai, "Tracking adaptive performance models using dynamic clustering of user classes," in *Proceedings of the 2nd ACM/SPEC International Conference on Performance engineering*, ICPE '11, (New York, NY, USA), pp. 179–188, ACM, 2011.

[71] A. Bertolino, E. Marchetti, and A. Morichetta, "Adequate monitoring of service compositions," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, (New York, NY, USA), pp. 59–69, ACM, 2013.

[72] L. Baresi, S. Guinea, O. Nano, and G. Spanoudakis, "Comprehensive monitoring of bpel processes," *IEEE Internet Computing*, vol. 14, pp. 50–57, May 2010.

[73] C. Ghezzi and S. Guinea, "Run-time monitoring in service-oriented architectures," in *Test and analysis of web services*, pp. 237–264, Springer, 2007.

[74] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka, "Towards pro-active adaptation with confidence: augmenting service monitoring with online testing," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '10, (New York, NY, USA), pp. 20–28, ACM, 2010.

[75] C. Bartolini, A. Bertolino, G. D. Angelis, A. Ciancone, and R. Mirandola, "Apprehensive qos monitoring of service choreographies," in *SAC*, pp. 1893–1899, 2013.

[76] A. Bertolino, A. Calabrò, and G. D. Angelis, "A generative approach for the adaptive monitoring of sla in service choreographies," in *ICWE*, pp. 408–415, 2013.

[77] X. Amatriain, J. M. Pujol, and N. Oliver, "I like it... i like it not: Evaluating user ratings noise in recommender systems," in *Proceedings of the 17th International Conference on User Modeling, Adaptation, and Personalization: formerly UM and AH*, UMAP '09, (Berlin, Heidelberg), pp. 247–258, Springer-Verlag, 2009.

[78] X. Amatriain, J. M. Pujol, N. Tintarev, and N. Oliver, "Rate it again: increasing recommendation accuracy by user re-rating," in *Proceedings of the third ACM conference on Recommender systems*, RecSys '09, (New York, NY, USA), pp. 173–180, ACM, 2009.

[79] D. Cosley, S. K. Lam, I. Albert, J. A. Konstan, and J. Riedl, "Is seeing believing?: how recommender system interfaces affect users' opinions," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, (New York, NY, USA), pp. 585–592, ACM, 2003.

[80] E. I. Sparling and S. Sen, "Rating: how difficult is it?," in *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, (New York, NY, USA), pp. 149–156, ACM, 2011.

[81] D. Kluver, T. T. Nguyen, M. Ekstrand, S. Sen, and J. Riedl, "How many bits per rating?," in *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, (New York, NY, USA), pp. 99–106, ACM, 2012.

[82] S. Nobarany, L. Oram, V. K. Rajendran, C.-H. Chen, J. McGrenere, and T. Munzner, "The design space of opinion measurement interfaces: exploring recall support for rating

and ranking," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, (New York, NY, USA), pp. 2035–2044, ACM, 2012.

[83] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "The adaptive web," ch. Collaborative filtering recommender systems, pp. 291–324, Berlin, Heidelberg: Springer-Verlag, 2007.

[84] M. Montaner, B. López, and J. L. De La Rosa, "A taxonomy of recommender agents on theinternet," *Artif. Intell. Rev.*, vol. 19, pp. 285–330, June 2003.

[85] L. Shao, J. Zhao, T. Xie, L. Zhang, B. Xie, and H. Mei, "User-perceived service availability: A metric and an estimation approach," in *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pp. 647–654, IEEE, 2009.

[86] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, (New York, NY, USA), pp. 175–186, ACM, 1994.

[87] M. R. McLaughlin and J. L. Herlocker, "A collaborative filtering algorithm and evaluation metric that accurately model the user experience," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, (New York, NY, USA), pp. 329–336, ACM, 2004.

[88] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Inf. Retr.*, vol. 4, pp. 133–151, July 2001.

[89] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans. Inf. Syst.*, vol. 22, pp. 5–53, Jan. 2004.

[90] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.

[91] G. Karypis, "Evaluation of item-based top-n recommendation algorithms," in *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, (New York, NY, USA), pp. 247–254, ACM, 2001.

[92] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *ACM Trans. Inf. Syst.*, vol. 22, pp. 143–177, Jan. 2004.

[93] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, (New York, NY, USA), pp. 230–237, ACM, 1999.

[94] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Analysis of recommendation algorithms for e-commerce," in *Proceedings of the 2nd ACM conference on Electronic commerce*, EC '00, (New York, NY, USA), pp. 158–167, ACM, 2000.

[95] S. H. S. Chee, J. Han, and K. Wang, "Rectree: An efficient collaborative filtering method," in *Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery*, DaWaK '01, (London, UK, UK), pp. 141–151, Springer-Verlag, 2001.

[96] D. Lemire, "Scale and translation invariant collaborative filtering systems," *Inf. Retr.*, vol. 8, pp. 129–150, Jan. 2005.

[97] X. Su, T. M. Khoshgoftaar, and R. Greiner, "A mixture imputation-boosted collaborative filter," in *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15-17, 2008, Coconut Grove, Florida, USA*, FLAIRS '08, pp. 312–316, 2008.

[98] X. Su, T. M. Khoshgoftaar, X. Zhu, and R. Greiner, "Imputation-boosted collaborative filtering using machine learning classifiers," in *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, (New York, NY, USA), pp. 949–950, ACM, 2008.

[99] R. J. A. Little, "Missing-Data Adjustments in Large Surveys," *Journal of Business & Economic Statistics*, vol. 6, pp. 287–296, 1988.

[100] H. Toutenburg, "Rubin, d.b.: Multiple imputation for nonresponse in surveys," *Statistical Papers*, vol. 31, no. 1, pp. 180–180, 1990.

[101] S. Goldman and M. Warmuth, "Learning binary relations using weighted majority voting," *Machine Learning*, vol. 20, no. 3, pp. 245–271, 1995.

[102] A. Nakamura and N. Abe, "Collaborative filtering using weighted majority prediction algorithms," in *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, (San Francisco, CA, USA), pp. 395–403, Morgan Kaufmann Publishers Inc., 1998.

[103] C. Basu, H. Hirsh, and W. Cohen, "Recommendation as classification: using social and content-based information in recommendation," in *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, (Menlo Park, CA, USA), pp. 714–720, American Association for Artificial Intelligence, 1998.

[104] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[105] K. Miyahara and M. J. Pazzani, "Improvement of collaborative filtering with the simple bayesian classifier.," *Transactions of Information Processing Society of Japan*, vol. 43.

[106] K. Miyahara and M. J. Pazzani, "Collaborative filtering with the simple bayesian classifier," in *Proceedings of the 6th Pacific Rim international conference on Artificial intelligence*, PRICAI'00, (Berlin, Heidelberg), pp. 679–689, Springer-Verlag, 2000.

[107] X. Su and T. Khoshgoftaar, "Collaborative filtering for multi-class data using belief nets algorithms," in *Tools with Artificial Intelligence, 2006. ICTAI '06. 18th IEEE International Conference on*, pp. 497–504, 2006.

[108] R. Greiner, X. Su, B. Shen, and W. Zhou, "Structural extension to logistic regression: Discriminative parameter learning of belief net classifiers," *Machine Learning*, vol. 59, no. 3, pp. 297–322, 2005.

[109] B. Shen, X. Su, R. Greiner, P. Musilek, and C. Cheng, "Discriminative parameter learning of general bayesian network classifiers," in *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '03, (Washington, DC, USA), pp. 296–, IEEE Computer Society, 2003.

[110] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Mach. Learn.*, vol. 29, pp. 131–163, Nov. 1997.

[111] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie, "Dependency networks for inference, collaborative filtering, and data visualization," *J. Mach. Learn. Res.*, vol. 1, pp. 49–75, Sept. 2001.

[112] J. Han, *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[113] X. Su, M. Kubat, M. A. Tapia, and C. Hu, "Query size estimation using clustering techniques," in *Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '05, (Washington, DC, USA), pp. 185–189, IEEE Computer Society, 2005.

[114] M. Ester, H. peter Kriegel, J. S, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," pp. 226–231, AAAI Press, 1996.

[115] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: ordering points to identify the clustering structure," in *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, SIGMOD '99, (New York, NY, USA), pp. 49–60, ACM, 1999.

[116] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, SIGMOD '96, (New York, NY, USA), pp. 103–114, ACM, 1996.

[117] M. Connor and J. Herlocker, "Clustering items for collaborative filtering," in *Proceedings of the ACM SIGIR Workshop on Recommender Systems*, SIGIR '99, 1999.

[118] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering," in *Proceedings of the fifth international conference on computer and information technology*, ICCIT '02, December 2002.

[119] L. H. Ungar and D. P. Foster, "Clustering methods for collaborative filtering," in *Proceedings of the AAAI Workshop on Recommendation Systems*, no. 1, AAAI Press, 1998.

[120] L. Si and R. Jin, "Flexible mixture model for collaborative filtering," in *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24,*

2003, Washington, DC, USA (T. Fawcett and N. Mishra, eds.), pp. 704–711, AAAI Press, 2003.

[121] T. Hofmann and J. Puzicha, "Latent class models for collaborative filtering," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, IJCAI '99, (San Francisco, CA, USA), pp. 688–693, Morgan Kaufmann Publishers Inc., 1999.

[122] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, and Z. Chen, "Scalable collaborative filtering using cluster-based smoothing," in *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, (New York, NY, USA), pp. 114–121, ACM, 2005.

[123] A. Y. Ng and M. Jordan, "Pegasus: a policy search method for large mdps and pomdps," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, UAI'00, (San Francisco, CA, USA), pp. 406–415, Morgan Kaufmann Publishers Inc., 2000.

[124] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, pp. 76–80, Jan. 2003.

[125] J. Canny, "Collaborative filtering with privacy via factor analysis," in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, (New York, NY, USA), pp. 238–245, ACM, 2002.

[126] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, vol. 39, no. 1, pp. 1–38, 1977.

[127] D. M. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles, "Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach," in *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, (San Francisco, CA, USA), pp. 473–480, Morgan Kaufmann Publishers Inc., 2000.

[128] D. Billsus and M. J. Pazzani, "Learning collaborative information filters," in *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, (San Francisco, CA, USA), pp. 46–54, Morgan Kaufmann Publishers Inc., 1998.

[129] S. Vucetic and Z. Obradovic, "Collaborative filtering using a regression-based approach," *Knowl. Inf. Syst.*, vol. 7, pp. 1–22, Jan. 2005.

[130] D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering," *Society for Industrial Mathematics*, vol. 5, pp. 471–480, 2005.

[131] R. Bellman, "A markovian decision process," *Indiana University Mathematics Journal*, vol. 6, pp. 679–684, 1957.

[132] R. Howard, *Dynamic Programming and Markov Processes*. Published jointly by the Technology Press of the Massachusetts Institute of Technology and, 1960.

[133] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. Massachusetts, USA: MIT Press, Cambridge University Press, 1998.

[134] G. Shani, D. Heckerman, and R. I. Brafman, "An mdp-based recommender system," *J. Mach. Learn. Res.*, vol. 6, pp. 1265–1295, Dec. 2005.

[135] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1–2, pp. 99 – 134, 1998.

[136] M. Hauskrecht, "Incremental methods for computing bounds in partially observable markov decision processes," in *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, AAAI'97/IAAI'97, pp. 734–739, AAAI Press, 1997.

[137] P. Poupart and C. Boutilier, "Vdcbpi: an approximate scalable algorithm for large pomdps," in *Proceedings of the 18th Annual Conference on Neural Information Processing Systems*, NIPS'04, 2004.

[138] M. Kearns, Y. Mansour, and A. Y. Ng, "A sparse sampling algorithm for near-optimal planning in large markov decision processes," *Mach. Learn.*, vol. 49, pp. 193–208, Nov. 2002.

[139] T. Hofmann, "Latent semantic models for collaborative filtering," *ACM Trans. Inf. Syst.*, vol. 22, pp. 89–115, Jan. 2004.

[140] T. Hofmann, "Unsupervised learning by probabilistic latent semantic analysis," *Mach. Learn.*, vol. 42, pp. 177–196, Jan. 2001.

[141] H. Research, "Eachmovie dataset." http://www.grouplens.org/node/76.

[142] B. M. Marlin, "Modeling user rating profiles for collaborative filtering," in *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*.

[143] B. Marlin, *Collaborative filtering: A machine learning perspective*. PhD thesis, University of Toronto, 2004.

[144] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[145] W. W. Cohen, R. E. Schapire, and Y. Singer, "Learning to order things," *J. Artif. Int. Res.*, vol. 10, pp. 243–270, May 1999.

[146] X. Fu, J. Budzik, and K. J. Hammond, "Mining navigation history for recommendation," in *Proceedings of the 5th international conference on Intelligent user interfaces*, IUI '00, (New York, NY, USA), pp. 106–112, ACM, 2000.

[147] C. W.-k. Leung, S. C.-f. Chan, and F.-l. Chung, "A collaborative filtering framework based on fuzzy association rules and multiple-level similarity," *Knowl. Inf. Syst.*, vol. 10, pp. 357–381, Oct. 2006.

[148] D. Y. Pavlov and D. M. Pennock, "A maximum entropy approach to collaborative filtering in dynamic, sparse, high-dimensional domains," in *Advances in Neural Information Processing Systems*, pp. 1441–1448, 2002.

[149] D. Nikovski and V. Kulev, "Induction of compact decision trees for personalized recommendation," in *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, (New York, NY, USA), pp. 575–581, ACM, 2006.

[150] C. C. Aggarwal, J. L. Wolf, K.-L. Wu, and P. S. Yu, "Horting hatches an egg: a new graph-theoretic approach to collaborative filtering," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '99, (New York, NY, USA), pp. 201–212, ACM, 1999.

[151] B. Marlin and R. S. Zemel, "The multiple multiplicative factor model for collaborative filtering," in *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, (New York, NY, USA), pp. 73–, ACM, 2004.

[152] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999.

[153] N. Srebro, J. Rennie, and T. S. Jaakkola, "Maximum-margin matrix factorization," in *Advances in neural information processing systems*, pp. 1329–1336, 2004.

[154] J. D. M. Rennie and N. Srebro, "Fast maximum margin matrix factorization for collaborative prediction," in *Proceedings of the 22nd international conference on Machine learning*, ICML '05, (New York, NY, USA), pp. 713–719, ACM, 2005.

[155] G. Takács, I. Pilászy, B. Németh, and D. Tikk, "Investigation of various matrix factorization methods for large recommender systems," in *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, NETFLIX '08, (New York, NY, USA), pp. 6:1–6:8, ACM, 2008.

[156] J. Wang, S. Robertson, A. P. Vries, and M. J. Reinders, "Probabilistic relevance ranking for collaborative filtering," *Inf. Retr.*, vol. 11, pp. 477–497, Dec. 2008.

[157] J. Wang, A. P. de Vries, and M. J. T. Reinders, "Unified relevance models for rating prediction in collaborative filtering," *ACM Trans. Inf. Syst.*, vol. 26, pp. 16:1–16:42, June 2008.

[158] M. J. Pazzani, "A framework for collaborative, content-based and demographic filtering," *Artif. Intell. Rev.*, vol. 13, pp. 393–408, Dec. 1999.

[159] T. Zhu, R. Greiner, and G. Häubl, "Learning a model of a web user's interests," in *Proceedings of the 9th international conference on User modeling*, UM'03, (Berlin, Heidelberg), pp. 65–75, Springer-Verlag, 2003.

[160] M. Pazzani and D. Billsus, "Learning and revising user profiles: The identification ofinteresting web sites," *Mach. Learn.*, vol. 27, pp. 313–331, June 1997.

[161] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, pp. 734–749, June 2005.

[162] M. K. Condliff, D. D. Lewis, and D. Madigan, "Bayesian mixed-effects models for recommender systems," in *In ACM SIGIR '99 Workshop on Recommender Systems: Algorithms and Evaluation*, 1999.

[163] B. Krulwich, "Lifestyle finder: Intelligent user profiling using large-scale demographic data," *AI Magazine*, pp. 37–45, 1997.

[164] R. H. Guttman, *Merchant differentiation through integrative negotiation in agent-mediated electronic commerce*. PhD thesis, Citeseer, 1998.

[165] M. Balabanović and Y. Shoham, "Fab: content-based, collaborative recommendation," *Commun. ACM*, vol. 40, pp. 66–72, Mar. 1997.

[166] P. Melville, R. J. Mooney, and R. Nagarajan, "Content-boosted collaborative filtering for improved recommendations," in *Eighteenth national conference on Artificial intelligence*, (Menlo Park, CA, USA), pp. 187–192, American Association for Artificial Intelligence, 2002.

[167] A. Ansari, S. Essegaier, and R. Kohli, "Internet recommendation systems," *Journal of Marketing research*, vol. 37, no. 3, pp. 363–375, 2000.

[168] A. E. Gelfand and A. F. Smith, "Sampling-based approaches to calculating marginal densities," *Journal of the American statistical association*, vol. 85, no. 410, pp. 398–409, 1990.

[169] B. M. Sarwar, J. A. Konstan, A. Borchers, J. Herlocker, B. Miller, and J. Riedl, "Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system," in *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, CSCW '98, (New York, NY, USA), pp. 345–354, ACM, 1998.

[170] R. J. Mooney and L. Roy, "Content-based book recommending using learning for text categorization," in *Proceedings of the fifth ACM conference on Digital libraries*, DL '00, (New York, NY, USA), pp. 195–204, ACM, 2000.

[171] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin, "Combining content-based and collaborative filters in an online newspaper," in *Proceedings of the ACM SIGIR '99 Workshop on Recommender Systems: Algorithms and Evaluation*, (Berkeley, California), ACM, 1999.

[172] J. A. Delgado, *Agent-based information filtering and recommender systems on the Internet*. PhD thesis, February 2000.

[173] X. Su, R. Greiner, T. Khoshgoftaar, and X. Zhu, "Hybrid collaborative filtering algorithms using a mixture of experts," in *Web Intelligence, IEEE/WIC/ACM International Conference on*, pp. 645–649, 2007.

[174] R. E. Schapire, "A brief introduction to boosting," in *Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, IJCAI'99, (San Francisco, CA, USA), pp. 1401–1406, Morgan Kaufmann Publishers Inc., 1999.

[175] B. Smyth and P. Cotter, "A personalised tv listings service for the digital tv age," *Knowl.-Based Syst.*, vol. 13, no. 2-3, pp. 53–59, 2000.

[176] M. Balabanović, "Exploring versus exploiting when learning user models for text recommendation," *User Modeling and User-Adapted Interaction*, vol. 8, pp. 71–102, Jan. 1998.

[177] A. Popescul, L. H. Ungar, D. M. Pennock, and S. Lawrence, "Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments," in *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, UAI '01, (San Francisco, CA, USA), pp. 437–444, Morgan Kaufmann Publishers Inc., 2001.

[178] K. Yu, A. Schwaighofer, V. Tresp, X. Xu, and H.-P. Kriegel, "Probabilistic memory-based collaborative filtering," *IEEE Trans. on Knowl. and Data Eng.*, vol. 16, pp. 56–69, Jan. 2004.

[179] C. ResearchIndex, "digital library of computer science research papers." http://citeseer.ist.psu.edu/.

[180] T. K. Landauer and M. L. Littman, "Computerized cross-language document retrieval using latent semantic indexing," Apr. 5 1994. US Patent 5,301,109.

[181] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *JASIS*, vol. 41, no. 6, pp. 391–407, 1990.

[182] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[183] C.-N. Ziegler, G. Lausen, and L. Schmidt-Thieme, "Taxonomy-driven computation of product recommendations," in *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, (New York, NY, USA), pp. 406–415, ACM, 2004.

[184] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, "Methods and metrics for cold-start recommendations," in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, (New York, NY, USA), pp. 253–260, ACM, 2002.

[185] B. M. Kim and Q. Li, "Probabilistic model estimation for collaborative filtering based on items attributes," in *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '04, (Washington, DC, USA), pp. 185–191, IEEE Computer Society, 2004.

[186] Z. Huang, H. Chen, and D. Zeng, "Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering," *ACM Trans. Inf. Syst.*, vol. 22, pp. 116–142, Jan. 2004.

[187] D. DeCoste, "Collaborative prediction using ensembles of maximum margin matrix factorizations," in *Proceedings of the 23rd international conference on Machine learning*, ICML '06, (New York, NY, USA), pp. 249–256, ACM, 2006.

[188] H. Noh, M. Kwak, and I. Han, "Improving the prediction performance of customer behavior through multiple imputation," *Intell. Data Anal.*, vol. 8, pp. 563–577, Dec. 2004.

[189] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Incremental singular value decomposition algorithms for highly scalable recommender systems," in *Fifth International Conference on Computer and Information Science*, pp. 27–28, Citeseer, 2002.

[190] M. W. Berry, S. T. Dumais, and G. W. O'Brien, "Using linear algebra for intelligent information retrieval," *SIAM Rev.*, vol. 37, pp. 573–595, Dec. 1995.

[191] K. Sparck Jones, "Document retrieval systems," ch. A statistical interpretation of term specificity and its application in retrieval, pp. 132–142, London, UK, UK: Taylor Graham Publishing, 1988.

[192] J. McCrae, A. Piatek, and A. Langley, "Collaborative filtering," *http:// www. imperialviolet. org*, 2004.

[193] P. Resnick and H. R. Varian, "Recommender systems," *Commun. ACM*, vol. 40, pp. 56–58, Mar. 1997.

[194] S. K. Lam and J. Riedl, "Shilling recommender systems for fun and profit," in *Proceedings of the 13th international conference on World Wide Web*, WWW '04, (New York, NY, USA), pp. 393–402, ACM, 2004.

[195] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams, "Effective attack models for shilling item-based collaborative filtering systems," in *Proceedings of the 2005 WebKDD Workshop, held in conjuction with ACM SIGKDD'2005*, 2005.

[196] M. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre, "Collaborative recommendation: A robustness analysis," *ACM Trans. Internet Technol.*, vol. 4, pp. 344–377, Nov. 2004.

[197] R. M. Bell and Y. Koren, "Improved neighborhood-based collaborative filtering," in *KDD Cup and Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, sn, 2007.

[198] B. N. Miller, J. A. Konstan, and J. Riedl, "Pocketlens: Toward a personal recommender system," *ACM Trans. Inf. Syst.*, vol. 22, pp. 437–476, July 2004.

[199] K. Yu, X. Xu, J. Tao, M. Ester, and H.-P. Kriegel, "Instance selection techniques for memory-based collaborative filtering.," in *SDM*, vol. 2, p. 16, 2002.

[200] G. Shafer, *A mathematical theory of evidence*, vol. 1. Princeton university press Princeton, 1976.

[201] D. Cai, M. F. McTear, and S. I. McClean, "Knowledge discovery in distributed databases using evidence theory," *International Journal of intelligent systems*, vol. 15, no. 8, pp. 745–761, 2000.

[202] T. M. Khoshgoftaar and J. Hulse, "Imputation techniques for multivariate missingness in software measurement data," *Software Quality Control*, vol. 16, pp. 563–600, Dec. 2008.

[203] Y. Koren, "Tutorial on recent progress in collaborative filtering," in *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, (New York, NY, USA), pp. 333–334, ACM, 2008.

[204] A. W. Services, "Amazon ec2," 2012. Elastic Compute Cloud.

[205] S. software, "Loadui." http://www.loadui.org/, 2012. Open source load and stress testing tool.

[206] D. Avresky, J. Arlat, J.-C. Laprie, and Y. Crouzet, "Fault injection for formal testing of fault tolerance," *Reliability, IEEE Transactions on*, vol. 45, no. 3, pp. 443–455, 1996.

[207] A. Vattani, "k-means requires exponentially many iterations even in the plane," in *Proceedings of the 25th annual symposium on Computational geometry*, SCG '09, (New York, NY, USA), pp. 324–332, ACM, 2009.

Biography

Marin Šilić was born in 1983 in Sarajevo, Bosnia and Herzegovina. He finished the elementary school and the gymnasium in Makarska. He received his M.Sc. degree in 2007 at the Faculty of Electrical Engineering and Computing, University of Zagreb, under the supervision of Professor Siniša Srbljić. During his studies at the University of Zagreb, Marin was receiving a scholarship from the Croatian Ministry of Science. As an outstanding student he was enrolled in advanced study program with a special emphasis on the research work. He was awarded with the *"Josip Lončar"* award which is given to the top graduating students in computing at the University of Zagreb. In 2007, Marin joined the Faculty of Electrical Engineering and Computing at the University of Zagreb as a research assistant. He actively participated on the research project entitled "End-User Tool for Gadget Composition" sponsored by Croatian Ministry of Science and Google. In 2008, Marin was an software engineering intern in Google Inc., in New York, USA. He was working in the *Google Docs* team on the design and implementation of the *Google Spreadsheets List View*. Marin has presented his research results in the papers that are published in the respected journal and at the top software engineering venue. He is a student member of the *IEEE*.

List of papers

1. ŠILIĆ, MARIN; DELAČ, GORAN; KRKA, IVO; SRBLJIĆ, SINIŠA. Scalable and Accurate Prediction of Availability of Atomic Web Services. // IEEE Transactions on Services Computing. (2013).

2. ŠILIĆ, MARIN; DELAČ, GORAN; SRBLJIĆ, SINIŠA. Prediction of Atomic Web Services Reliability for QoS-aware Recommendation. // IEEE Transaction on Services Computing. (2013).

3. DELAČ, GORAN; ŠILIĆ, MARIN; SRBLJIĆ, SINIŠA. A Reliability Improvement Method

for SOA-Based Applications. // IEEE Transactions on Dependable and Secure Computing. (2012).

4. DELAČ, GORAN; ŠILIĆ, MARIN; VLADIMIR, KLEMO. Reliability Sensitivity Analysis for Yahoo! Pipes Mashups // Proceedings of the 36th International Convention of Information Communication Technology, Electronics and Microelectronics, MIPRO 2013.

5. ŠILIĆ, MARIN; DELAČ, GORAN; SRBLJIĆ SINIŠA. Prediction of atomic web services reliability based on k-means clustering // Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE'13 / Meyer, Bertrand ; Baresi, Luciano ; Mezini, Mira, editor(s). New York, NY, USA : ACM, 2013. 70-80.

6. DELAČ, GORAN; ŠILIĆ, MARIN; SRBLJIĆ, SINIŠA. Reliability Modeling for SOA Systems // Proceedings of the 35th International Convention of Information Communication Technology, Electronics and Microelectronics, MIPRO 2012. 2012. 847-852.

7. PAVLIĆ, ZVONIMIR; LUGARIĆ, TOMISLAV; ŠILIĆ, MARIN. Debugging in consumer-programming oriented environments // Proceedings of the International Conference on Computers in Technical Systems. 2012. 982-987.

8. DELAČ, GORAN; ŠILIĆ, MARIN; KROLO, JAKOV. Emerging Security Threats For Mobile Platforms // Proceedings of the Information Systems Security, MIPRO 2011 / Čišić, Dragan ; Hutinski, Željko ; Baranović, Mirta ; Mauher, Mladen ; Ordanić, Leo, editor(s). Zagreb : Croatian Society for Information and Communication Technology, Electronics and Microelectronics - MIPRO, 2011. 126-131.

9. ŠILIĆ, MARIN; KROLO, JAKOV; DELAČ, GORAN. Security Vulnerabilities in Modern Web Browser Architecture // Proceedings of the Information Systems Security, MIPRO 2010 / Čišić, Dragan ; Hutinski, Željko ; Baranović, Mirta ; Mauher, Mladen ; Pletikosa, Marko, editor(s). Croatian Society for Information and Communication Technology, Electronics and Microelectronics - MIPRO, 2010. 198-203.

10. KROLO, JAKOV; ŠILIĆ MARIN; SRBLJIĆ SINIŠA. Security of Web Level User Identity Management // Proceedings of the Information Systems Security, MIPRO 2009 / Čišić, Dragan ; Hutinski, Željko ; Baranović, Mirta ; Mauher, Mladen ; Dragšić, Veljko, editor(s). Croatian Society for Information and Communication Technology, Electronics and Microelectronics - MIPRO, 2009. 93-98.

Životopis

Marin Šilić rođen je 1983. godine u Sarajevu u Bosni i Hercegovini. Osnovnu školu i Opću gimnaziju završio je u Makarskoj. Diplomirao je 2007. godine na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu pod mentorstvom prof.dr.sc. Siniše Srbljića. Tijekom studija primao je stipendiju Ministarstva Znanosti Republike Hrvatske namijenjenu posebno nadarenim studentima. Kao izvrstan student upisao je poseban program završetka studija s naglaskom na znanstveno-istraživački rad. Nakon završetka studija dobio je brončanu plaketu "Josip Lončar" koja se uručuje najboljim studentima računarstva na Sveučilištu u Zagrebu. Od 2007. godine zaposlen je kao znanstveni novak na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu. Aktivno je sudjelovao na istraživačkom projektu “End-User Tool for Gadget Composition” pod pokroviteljstvom Ministarstva znanosti Republike Hrvatske te kompanije Google. U 2008. godini boravio je na znanstvenom usavršavanju u kompaniji Google u uredu u New York-u, SAD. Tijekom usavršavanja radio je u *Google Docs* timu na oblikovanju i ostvarenju primjenskog sustava *Google Spreadsheets List View*. Rezultate svojih istraživanja Marin je opisao u člancima koji su objavljeni u uglednom časopisu i najjačem skupu istraživača područja programskog inženjerstva. Član je strukovne udruge *IEEE*.

Popis radova

1. ŠILIĆ, MARIN; DELAČ, GORAN; KRKA, IVO; SRBLJIĆ, SINIŠA. Scalable and Accurate Prediction of Availability of Atomic Web Services. // IEEE Transactions on Services Computing. (2013).

2. ŠILIĆ, MARIN; DELAČ, GORAN; SRBLJIĆ, SINIŠA. Prediction of Atomic Web Services Reliability for QoS-aware Recommendation. // IEEE Transaction on Services Computing. (2013).

3. DELAČ, GORAN; ŠILIĆ, MARIN; SRBLJIĆ, SINIŠA. A Reliability Improvement Method

for SOA-Based Applications. // IEEE Transactions on Dependable and Secure Computing. (2012).

4. DELAČ, GORAN; ŠILIĆ, MARIN; VLADIMIR, KLEMO. Reliability Sensitivity Analysis for Yahoo! Pipes Mashups // Proceedings of the 36th International Convention of Information Communication Technology, Electronics and Microelectronics, MIPRO 2013.

5. ŠILIĆ, MARIN; DELAČ, GORAN; SRBLJIĆ SINIŠA. Prediction of atomic web services reliability based on k-means clustering // Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE'13 / Meyer, Bertrand ; Baresi, Luciano ; Mezini, Mira (ur.). New York, NY, USA : ACM, 2013. 70-80.

6. DELAČ, GORAN; ŠILIĆ, MARIN; SRBLJIĆ, SINIŠA. Reliability Modeling for SOA Systems // Proceedings of the 35th International Convention of Information Communication Technology, Electronics and Microelectronics, MIPRO 2012. 2012. 847-852.

7. PAVLIĆ, ZVONIMIR; LUGARIĆ, TOMISLAV; ŠILIĆ, MARIN. Debugging in consumer-programming oriented environments // Proceedings of the International Conference on Computers in Technical Systems. 2012. 982-987.

8. DELAČ, GORAN; ŠILIĆ, MARIN; KROLO, JAKOV. Emerging Security Threats For Mobile Platforms // Proceedings of the Information Systems Security, MIPRO 2011 / Čišić, Dragan ; Hutinski, Željko ; Baranović, Mirta ; Mauher, Mladen ; Ordanić, Leo (ur.). Zagreb : Croatian Society for Information and Communication Technology, Electronics and Microelectronics - MIPRO, 2011. 126-131.

9. ŠILIĆ, MARIN; KROLO, JAKOV; DELAČ, GORAN. Security Vulnerabilities in Modern Web Browser Architecture // Proceedings of the Information Systems Security, MIPRO 2010 / Čišić, Dragan ; Hutinski, Željko ; Baranović, Mirta ; Mauher, Mladen ; Pletikosa, Marko (ur.). Croatian Society for Information and Communication Technology, Electronics and Microelectronics - MIPRO, 2010. 198-203.

10. KROLO, JAKOV; ŠILIĆ MARIN; SRBLJIĆ SINIŠA. Security of Web Level User Identity Management // Proceedings of the Information Systems Security, MIPRO 2009 / Čišić, Dragan ; Hutinski, Željko ; Baranović, Mirta ; Mauher, Mladen ; Dragšić, Veljko (ur.). Croatian Society for Information and Communication Technology, Electronics and Microelectronics - MIPRO, 2009. 93-98.