Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Lahiru Sandakith Pileththuwasan Gallege

Entitled

TRUST-BASED SERVICE SELECTION AND RECOMMENDATION FOR ONLINE SOFTWARE MARKETPLACES - (TRUSSTREMARK)

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Prof. Rajeev Raje
Chair

Prof. James Hill

Prof. Fengguang Song

Prof. Mihran Tuceryan

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Prof. Rajeev Raje

Approved by: Prof. Shiaofen Fang 12/5/2016
Head of the Departmental Graduate Program Date

TRUST-BASED SERVICE SELECTION AND RECOMMENDATION FOR ONLINE SOFTWARE MARKETPLACES (TRUSSTREMARK)

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Lahiru Sandakith Pileththuwasan Gallege

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2016

Purdue University

Indianapolis, Indiana

පිදුම

චන්දිමා සහ රෙහාන් පුතාට

අම්මා සහ තාත්තාට

Dedication

To Chandima and Rehan

To my parents

ACKNOWLEDGEMENTS

It has been an honor to be a graduate student at the Department of Computer and Information Science at IUPUI (Indiana University-Purdue University, Indianapolis). I highly value my experience and memories of working as a teaching and research assistant at the Department of Computer and Information Science (CSCI), IUPUI. As I step into the computer science industry, the knowledge and experience gained throughout my graduate studies will be invaluable. I would like to take this opportunity to remember and thank the people who have been supportive throughout my studies here at IUPUI.

First and foremost, I would like to thank my advisor, Professor Rajeev R. Raje, for his guidance and encouragement during the courses of my graduate studies. He constantly guided and motivated me to achieve higher objectives as a research student.

I would also like to thank Prof. James Hill, and Prof. Fengguang Song and Prof. Mihran Tuceryan for being part of my Dissertation Committee and for providing their valuable feedback. Our research work is supported in part by the US Air Force Research Laboratory (AFRL) through the Security and Software Engineering Research Center (S^2ERC).

I would like to remember and thank my colleagues at our SL 116 lab (especially Dimuthu, Manjula, Ryan, Harold, Ketaki, Aboli, Omkar, Suchata, Neha, Amrutha, Manohar and Tanumoy) for being there to engage with discussions and for providing their valuable feedback.

I thank Dr. Sanjiva Weerawarana and my colleagues from WSO2 Inc for encouraging me to pursue higher studies and I thank all of you for being role models for myself as well as for many other Sri Lankan students. I would like to especially thank Dimuthu Gamage, Manjula Peiris, Ruwan Egodagamage and the Sri Lankan community here at Indiana State for making our time there a wonderful experience.

I would also like to thank Dr. Christopher Groer, Dr. Thilina Gunarathne and our former Link Analytics team at KPMG LLP for providing me a great employment environment and their support to work on my dissertation.

I like to thank the faculty at the Department of Computer and Information Science for their support and I would also like to thank the department staff (especially Nicole Wittlief and Aubrey Reynolds for their proofreading efforts) and IT support staff at IUPUI for their support. Also, I would like to thank the staff of the Purdue School of Science Office at IUPUI (especially Summer Layton) for their help during the thesis formatting reviews.

Finally, I would like to thank my parents, my wife Chandima and my son Rehan as I dedicate this dissertation for their unconditional love and support.

TABLE OF CONTENTS

Page

Page

Page

APPENDICES

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

PREFACE

This dissertation proposes a framework (TruSStReMark - Trust-based Selection and Recommendation for Online Software Marketplaces) to model, quantify, and monitor trust of software applications (i.e., apps – also in the context of this dissertation, we use terms services and apps interchangably) and to perform trust-based service selection and recommendations. It provides methods to analyze and aggregate external reviews, pertaining to specific QoS attributes, of software apps by performing subjective logic-based operations. This framework, first, defines trust of a software apps using theory of belief and extends the multi-level software specifications to represent the trust-based attributes. It, then, proposes enhancements to two prevalent algorithms for selecting and recommending software apps from a marketplace. Finally, the performances of the enhanced selection and recommendation algorithms are improved by parallelizing them. When compared with the prevalent Content-based and Collaborative filtering-based approaches, the results show that, the TruSStReMark is able to produce better results in terms of quality measured using HR (Hit Ratio) and ARHR (Average Reciprocal Hit-Rank) metrics. In addition, the parallelized versions of the trust-based selection and recommendation algorithms improve the end-to-end runtime. The TruSStReMark will enable users to select apps, which are trustworthy from online software marketplaces and use them in composing quality-aware software systems.

LIST OF ABBREVIATIONS

Apps	Software-based Mobile Applications
API	Application Programming Interface
CBF	Content-based Filtering
CLF	Collaborative Filtering
DCS	Distributed Computing Systems
ETL	Extract, Transform and Load
IoT	Internet of Things
MLC	Multi-level Contracts
MLM	Multi-level Matching
MLS	Multi-Level Specification
OS	Operating System
RMSE	Root Mean Squared Error
SOA	Service Oriented Architecture
SS	Software Systems
QoS	Quality of Service
UniFrame	Unified Framework

ABSTRACT

Pileththuwasan Gallege, Lahiru Sandakith. Ph.D., Purdue University, December 2016. Trust-based Service Selection and Recommendation for Online Software Marketplaces (*TruSStReMark*). Major Professor: Rajeev R. Raje.

Nowadays, a vast number of heterogeneous devices exist, which are connected to their corresponding online marketplaces over the Internet. These marketplaces contain large volumes of individually produced and maintained software applications (i.e., apps) which can be dynamically installed into these devices. Hence, these complex devices now contain a dynamic software system where applications should seamlessly work together with underlying hardware platform. When users of these devices need to consume a software service (or to use a software application, i.e., app – also in this context, we use terms services and apps interchangably), they would search and install these apps from those linked online marketplaces. Moreover, app providers and device administrators can recommend apps for users, possibly targeting a selected set of devices. Hence, the selection and recommendation mechanisms play an important role in identifying the necessary (and possibly sufficient) set of individual apps which match to a users' requirements.

Due to the dynamic nature of the software apps, there exists an inherent difference in selecting and recommending physical products and software apps. Therefore, product-based recommendation techniques cannot be directly applied to recommending online apps which are updated frequently by changing their internals such as the source code and libraries.

Analyzing famous software marketplaces reveal that finding the best app that matches with device properties and users' requirements is a task assigned to user. However, online software marketplaces could be more effective if they could provide better search and recommendation results. If the trustworthiness of individual app is identified and quantified, it enables dynamic integration of apps to produce robust composed software systems. Therefore, identifying and quantifying the trust of these publicly available apps become prerequisites for providing better (i.e., more trustworthy) search and recommendation results from online marketplaces. However, this is challenging since the trustworthiness of these apps is affected by the heterogeneity found in associated set of evidence and opinions, the dynamic nature of trust, and the presence of uncertainty. Moreover, the trust of a app is not a static value. If the trustworthiness of an app is quantified, it should then be updated periodically to reflect the dynamic nature of the software development cycle. Therefore, identifying the trust of an individual app and a composed system is not a trivial task. To achieve this task, there is a need to aggregate and monitor the trust of individual apps in a continuing manner.

The approach proposed by this dissertation quantifies the trust of a software app using available data (e.g., external evidence which describe these apps, such as app feature descriptions, developer comments and user reviews etc.) following an evidence-based approach using subjective logic operations. The steps in this quantification process are to conduct a literature survey about existing views of trust, to define a trust model, and to use that model to monitor the trust of public apps available form software marketplaces. The results and the knowledge gained in quantifying trust is employed to improve the app selection and recommendation processes. The proposed approach identifies abstract

structures of prevalent app selection and recommendation algorithms and then enhances these prevalent algorithms to include trustworthy service selection and recommendation principles.

This work (named - *TruSStReMark*) also performs various experiments and compares the enhanced algorithms with prevalent approaches. These trust-based algorithms are categorized into two phases - an offline phase (i.e., ETL -Extract, Transform and Load, categorization and organization of available evidence) and an online phase (i.e., execution of the algorithms to analyze collection of evidence in real-time). Next, the *TruSStReMark* framework is parallelized using MapReduce concepts using the Hadoop framework which improves its performance to produce results in near real-time. Furthermore, with the availability of new technologies the overall performance is improved by removing Hadoop-based batch oriented processing with the adaptation of Spark and Spark Streaming to improve the efficiency of both online and offline phases. Publically available mobile app apps from the Android and Amazon app marketplaces are investigated (as the datasets) to empirically validate the proposed approach.

CHAPTER 1. INTRODUCTION

Nowadays, concepts related to the Internet of Things (IoT) [1] are popular among software research and development communities. IoT concepts allow networking of different physical devices (such as computers, mobiles, appliances, vehicles, etc.) to communicate over the Internet. As the number of these devices increase, it is estimated that by 2020 the IoT global network will consist of almost 50 billion devices [2]. These heterogeneous devices are connected to their corresponding online marketplaces to download their required software updates and install new applications (i.e., apps[1]). Hence, each different type of device (i.e., which hosts several active artifacts inside its environment) can now search, install, update, or get recommended suitable software artifacts seamlessly downloaded to these devices.

A large collection of such publicly available apps from one environment are organized into a software marketplace. Examples of such online service marketplaces are Amazon Marketplace [3] (i.e., Amazon's Software Application Marketplace targeting Android and Fire Operating System (FireOS) based devices), Ubuntu Marketplace [4] (i.e., Canonical's Software Application Marketplace targeting Ubuntu OS based devices), and Google Play Marketplace [5] (i.e., Google's Application Marketplace targeting Android based devices). *Table 1-1* gives a list of popular online software makrtplaces including their targeted platforms.

[1] In the context of this dissertation, the term 'Apps' refers to software applications and/or services installed on those above-mentioned heterogeneous devices. Also in the context of this this dissertation we use terms services and apps interchangeably.

Service Oriented Architecture (SOA) concepts allow many independent service providers to develop different apps for the same set of requirements. *Figure 1.1* indicates how the concepts from Service Oriented Architecture (SOA) are applied to online software marketplaces. The main difference between SOA Concepts (i.e., left section of *Figure 1.1*) and marketplaces (i.e., right section of *Figure 1.1*) is that marketplaces allow limited or no interactions between devices and service builders.

Table 1-1 List of major online software marketplace information [2]

Marketplace	**Owner**	**Base Platform**	**Restrictions**
Amazon Appstore [3]	Amazon Inc.	Android	Fire OS
Ubuntu Appstore [4]	Canonical Inc.	Ubuntu OS	All Versions
Google Play [5]	Google Inc.	Android	All Versions
Apple Appstore [6]	Apple Inc.	iOS	All Versions
BlackBerry World [7]	BlackBerry Inc.	BlackBerry OS	Tablet OS
Microsoft Phone Store [8]	Microsoft Inc.	Windows Mobile	All Versions
Microsoft Store [9]	Microsoft Inc.	Windows	All Versions
Amazon Software Store [10]	Amazon Inc.	Any	All Versions
Google WebApps Store [11]	Google Inc.	Web Applications	All Versions

Online app marketplaces intercept all communication between devices and builders. Third-party app builders who upload/update their artifacts to marketplaces and devices which download and install/update these artifacts from marketplaces. As many instances for a specific app requirement are available, app consumers are presented with the task of selection (i.e., evaluate different instances, compare them and make a choice).

[2] Amazon, FireOS, Ubuntu, Canonical, Google, Apple, iOS, Blackberry, Windows, and Microsoft are trademarks of the respective entities.

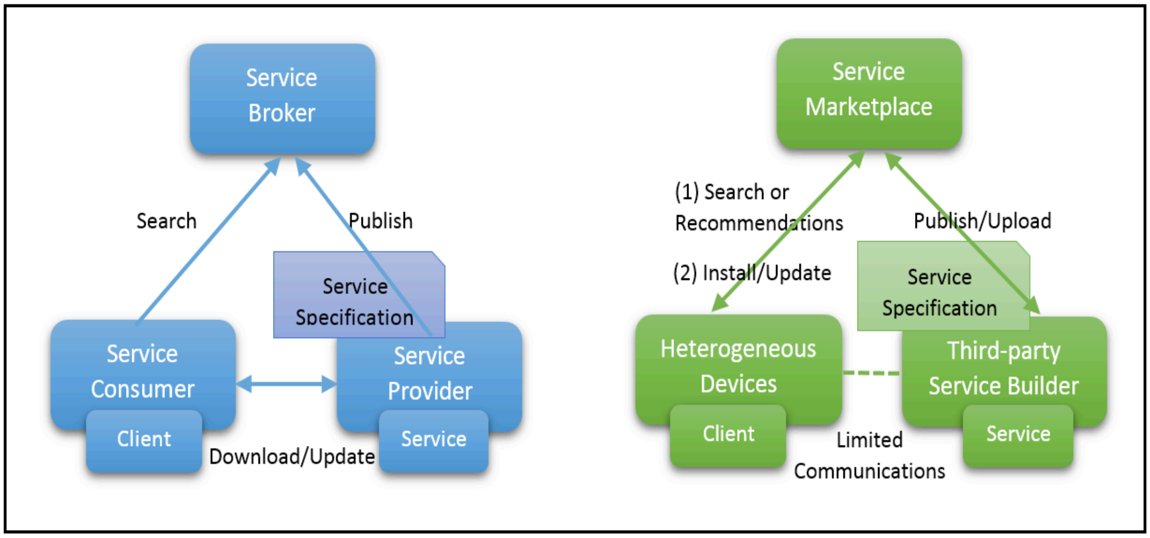


Figure 1.1 SOA based concepts are applied to online serivce marketplaces

Due to limited interactions between device owners and service builders, there is a lack of trust which present a challenge for service selection and recommendations. Therefore, marketplaces should include information related to the trustworthiness of its artifacts, which is the main focus of this dissertation (in *Figure 1.2*).

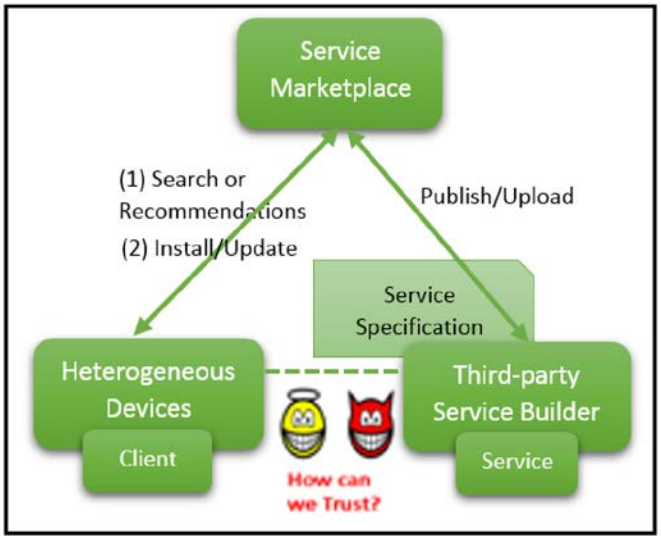


Figure 1.2 Lack of communication between devices and service builders which raises challenges related to trustworthiness

When considering efforts, time and cost factors to build apps themselves, the service consumers are forced to take advantage of current features from these service marketplaces and are encouraged to design and compose software systems by reusing such public apps. These marketplaces could also recommend apps to their consumers based on their domains and interests. As more software systems are designed by composing independently developed apps, identifying the trust of these different service instances becomes a prerequisite for assembling any trustworthy software system (e.g., Plug and play platforms present in mobile devices).

The concept of trust is highly subjective in nature and is affected by many associated evidence[3] and opinions [12]. In contrast to other domains, such as economics and social sciences, the trust aspects of software (specifically in the context of public software apps hosted by different venders) are yet to be standardized. Our previous survey [13] confirmed this fact (i.e., non-existence of a governing definition of trust) by analyzing publications which discuss the trustworthiness of software apps. This survey identified 80 relevant peer-reviewed papers and analyzed how each of them views the trust aspect of the software (or apps). Various trust definitions of software from these papers include views ranging from simple (such as, “Trust can be simplified to two important factors reputation and competence” [14] – Shou-xin et al. 2009) to more complex (such as, “An entity can be trusted if it always behaves in the expected manner for the intended purpose” [15] - Trusted Computing Group 2002).

This dissertation (in CHAPTER 5) includes a summary of the survey of trust views and the proposed principles for developing software systems considering trust from the beginning, rather than consider trust as an afterthought in the context of software

[3] In this dissertation, the term evidence is referring to the plural noun (i.e., to represent collection of evidence).

development process. Based on the findings of our survey [13], our definition of the trust of a software service is also related to the definition proposed by the Trusted Computing Group above. Hence, we define the trust associated with a software service as *"the degree of conformance of its behavior (both functional and non-functional) to its published specification (or set of contracts)"*.

As there is no standardized definition of trust, in the context of software systems, the traditional way to represent trust is to include it as another quality of service (QoS) attribute. However, this representation has limited value at the time of service selection, where individual service contracts and other relevant information are matched against existing system requirements. Considering the trust of a service as one QoS attribute (i.e., confidence about its other attributes and behavior) fails to present many details such as how trust is aggregated over time. Trust being yet another QoS attribute also does not provide the additional rationale necessary to explain why a particular service was given a specific score by its users. As highlighted by the related works section in CHAPTER 2 (i.e., most of the work based on [16] and [17]), this issue is one of the significant limitation of prevalent approaches. If the trust of a service is considered as an aggregated measure of the evidences and opinions about the confidence of the service behavior (under various operating conditions), then the service selection mechanism is presented with additional information and can use this information more effectively.

In addition, the trust of a service is not a static value. Due to changes in operating conditions, the trust of a service can fluctuate over time. For example, in a shorter time frame, because of the change in current work load of a server, the apps' ability to provide certain QoS can change. Similarly, over a long time frame, an updated version of a service may introduce additional bugs which were not present in an earlier version of that service and may change the trust value. Considering public apps, if the trustworthiness of these apps is identified and quantified, these trust values could be updated periodically to address both short-term and long-term variations of the trustworthiness of the software service.

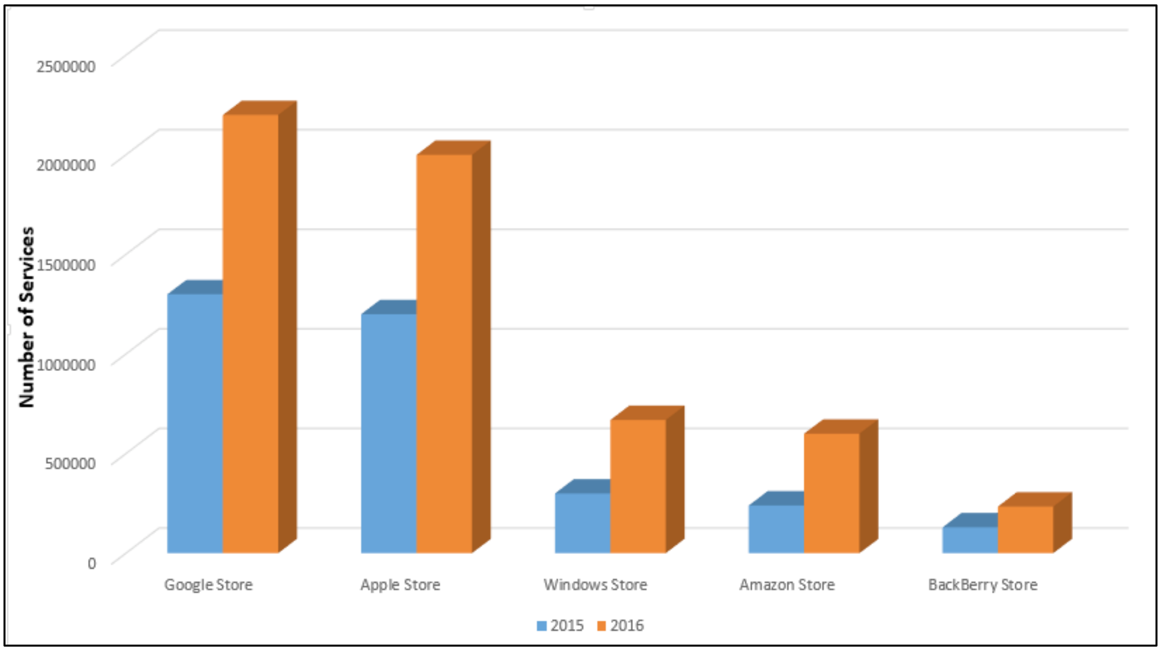


Figure 1.3 Number of apps available from each marketplace (Statistra [18])

For example, let us consider a device platform (i.e., mobile device) searching for a software application which operates on a resource constrained device. The Android platform [19] includes software-based applications which communicate with apps inside the device (e.g., notification apps) and apps outside the device (e.g., data apps typically hosted in the cloud). An example of such a service is a weather forecasting service, and there are many variations of this type of service are publically available from different external vendors such as Yahoo Weather Service [20] and Weather Channel App [21]. However, it is important to note that this service requirement can be filled with either using one such service or using a set of apps. Also, the requirements (for a mobile weather forecasting service) are complex and have both functional (i.e., feature availability, timeliness, and accuracy of the results) and non-functional aspects (i.e., text and image resolution, battery power usage, and response time), and these requirements vary depending on different inputs and output types. However, given a fixed set of requirements, the service selection process can search for suitable candidates for mobile weather apps. The issue is to find the best match given that different vendors could compete to provide same service covering various aspects of the set of requirements. *Figure 1.3* indicates statistics from 2015 and 2016 on how these marketplaces are exploding with new apps and applications forming a big-data challenges.

On the other hand, service marketplaces can recommend candidate apps too. Both cases inspect public service repositories (i.e., service marketplaces listed in *Table 1-1*), such as Amazon software marketplace [3] and Android Marketplace [5], to produce a set of candidate apps from different vendors to the consumer for a specific requirement. Some requirements, such as low power usage, can only be evaluated by evidence based approaches (e.g., no activity or bugs exist related to draining energy rapidly from the device). However, these evidence-based evaluations are also time sensitive and can only be evaluated using up-to-date external information (e.g., user reviews which get updated periodically in real-time). Again, this observation motivates us to aggregate and monitor the trust of individual apps as a continuing activity.

Therefore, the service selection process can benefit from availability of a trust contract which contains a summary of the various trust related information as a part of the service specification. This dissertation proposes to include trust attributes inside the prevalent structure of the software specifications. These trust attributes can correlate to other software attributes (e.g., service API signature) from all levels of software specification. Such a correlation provides a better way to explain why a particular service was given its current trust score. Aggregating these scores over time can provide additional information to effectively address queries about the trustworthiness of apps.

When a software service is selected for a given set of requirements, the process of selection can be measured based on the quality of results (e.g., relative ranking of the results and possible other recommendations). With the recent popularity of both product and software marketplaces, it is natural to compare these marketplaces by the process of selecting and recommending a physical product and a software service. Prevalent approaches of product-based ranking and recommendation do not consider the information about the items (i.e., a product) beyond attribute-value pairs. For example, one widely accepted technique is Item-Item Collaborative Filtering (CLF) [16]. It is based on the concept of "similar users buy similar items". It does not consider item details and the algorithm works on a large (typically very sparse) User-Item matrix. The other prominent technique for ranking and recommendation is Content-based Filtering (CBF) [17]. It does consider item details; however, it models the item details using quantifiable attribute-value pairs. Background concepts of CBF and CLF techniques are presented in CHAPTER 3 (Section 3.2).

The goal of these two techniques (or their combination) is to rank and recommend items using a similarity-function (such as cosine similarity or Pearson correlation comparison techniques). Compared to products, reusable software apps (such as software-based mobile applications and online data apps) are hard to describe using a set of fixed attribute-value pairs or using user-item relationships, due to the dynamic nature of software apps. Despite similarities between software apps and physical products, there are many subtle differences between them. As described in our survey [13], these main differences are: 1) dynamic updates that are very common in software apps, 2) variable quality of service (QoS) agreements between users and the service providers, 3) purpose of these apps being different such as some apps are used to compose systems out of individual software apps, and 4) variable subscription-based payment methods (e.g., based on load, time of the day etc.). Therefore, due to these associated challenges, it is not easy or straightforward to apply product-based ranking and recommendation techniques to the software apps domain.

The challenge can be further emphasized using the following two types of service selection and recommendation sample use cases. Case 1: a user’s goal is to operate a set of apps as individual and distinct entities (i.e., they do not attempt to relate or compose the apps together), and Case 2: a user’s goal is to select apps to integrate with other apps in the context of creating a composed system. The first scenario (Case 1) is somewhat similar to the conditions which are present in product selection, which forces us to view a software service through the lens of a traditional product. However, the dimensions of a product and the dimensions of software do not compare well with each other. For example, a typical product description considers its physical features (such as color, dimensions, weight, etc.) and the confidence associated with these features are certain and measurable. In contrast, software apps consist of more complex features (such as different QoSs - e.g., response time and accuracy of a weather forecasting service) and the confidence associated with these measurements is uncertain and dependent upon various factors (i.e., QoSs vary according to the current state of the execution environment).

Considering the second scenario (Case 2), if the apps are to interact with other apps, a simple attribute-based selection ranking and recommendation, suggested by prevalent approaches, will fall short due to various factors such as input/output compatibility and modes of communication. To provide better selection rankings and recommendations, it is necessary to measure or keep track of additional levels of information about a software service such as details about its contract attributes from its specification. For example, a service's Multi-Level Specification (MLS), including its syntax, semantics, and QoS levels, is needed for such a selection and recommendation process. Hence, to leverage various evidences (e.g., specifications) related to a software service, this dissertation proposes an evidence-based approach to rank and recommend software-based entities to be integrated into a system, while considering dependency and association between apps. Other challenges in quantifying the trust of these apps are: the heterogeneity in associated evidences and opinions, the difficulty associated with leveraging additional external and internal information from a software based service (e.g., aggregating credible evidences from related user reviews and test reports), the dynamic nature of trust, the presence of uncertainty, and the difficulty with mapping existing techniques to the environment of a service marketplace (such as how the user-item relationship is formed with the presence of large number of anonymous users).

To address these challenges, this dissertation has taken a holistic view in quantifying the trust of a service. This view has two aspects – an internal view which uses all internal facets (e.g., programmatic artifacts such as specifications, test logs, association between related apps, etc.) and an external view which uses non-programmatic artifacts such as user reviews posted in public marketplaces. Although the emphasis of this work is mainly focused on external view, these two views are merged using evidence-based techniques. Different evidences available are also aggregated and merged using evidence-based techniques (such as theory of belief [22], principles of subjective logic [23]), and Natural Language Processing (NLP) techniques [24] (such as sentiment analysis [25]). This merger not only assigns a trust score to a service but also reduces the associated uncertainty.

The proposed selection and recommendation method modifies both the CBF and CLF algorithms to include trust-based calculations. This approach (named “Trust-based Service Selection and Recommendation for Online Software Marketplaces” – in short as “*TruSStReMark”*) performs various experiments and compares the enhanced algorithms with prevalent approaches. Our approach is categorized into two phases called as online and offline phases. The offline phase performs scraping of evidences from marketplace, organize and cleansed them into a data warehouse (i.e., ETL -Extract, Transform and Load, categorization and organization of evidences). The online phase uses the data warehouse to generate and update metadata related to trust contracts and to perform analysis (i.e., execution of the algorithms to analyze evidences in real-time). Real-time analysis of evidences for one service is fast enough to perform in a single computing node, however, the next step was to improve our solution to perform parallel quarries from different users and handle the load of a real marketplace. As the number of apps and number of parallel queries (roughly indicated by number of downloads) increases (Figure 1.3), both online and offline phases needed modifications such that they can perform analysis and produce results in real-time. When loaded with parallel user quarries to simulate an online service marketplace, the performance of these evidence-based operations in both phases are affected as a result of these modifications. Hence, we then parallelizes our approach and these enhanced algorithms to improve the performance using the Hadoop echo-system [26] by placing the external evidences (generated as a results of offline phase) in Hadoop Distributed File System (HDFS) [27] and executing MapReduce [28] jobs frequently to update the trust-based scores and service metadata used by the real-time queries. These sets of experiments and analysis of this technique revealed that the concepts related to batch oriented operations in Hadoop is a bottleneck and hinder the overall performance. Therefore, as the final step of this dissertation, the overall performance is improved by removing Hadoop-based batch oriented processing with the adaptation of Spark [29] and Spark Streaming [30] to improve the efficiency of both online and offline phases. Further details about the steps in our approach are presented in CHAPTER 4.

In summary, the methods proposed by this dissertation first define the trust of a software service, suggest a concrete format for a trust contract, monitor the trust of public apps as a long-term activity, update trust contracts periodically (which are maintained as a separate part of the service specification), and use the aggregated trust related attributes over time to improve the service selection and recommendation process. The proposed trustworthy service selection and recommendation algorithms, which are enhancements of the two prevalent approaches, for software marketplaces perform better with respective to the quality of the results in root mean square error, precision, recall, Hit-Rate (HR), and Average Reciprocal Hit-Rate (ARHR) matrices when compared with the same two prevalent (CBF and CLF) approaches. Furthermore, with the availability of new technologies the overall end-to-end performance is improved by using Hadoop and Spark echo-systems to improve the efficiency of both online and offline phases.

1.1 Thesis Statement

The thesis statement can be summarized to a short sentence as “we believe that analyzing trust-based properties of software apps from publically available data can be used to improve selection and recommendations in online software marketplaces”. This statement is expanded more broadly into the following sub-statements:

- Having a comprehensive and holistic view of the trust of a software app or service and representing trust attributes of software apps inside a software specification (as a trust contract) can improve selection and recommendation of software apps and services hosted in current service marketplaces.
- Periodically calculating, aggregating and analyzing trust-based evidences of software apps and services over time (from both public and private data sources) using modified algorithms with necessary parallelization could improve user experience related to selection and recommendation in service marketplaces.

1.2 Thesis Objectives

The specific objectives of this thesis are:

1) To formally define the trust of a software app or service and propose the evidence based model for trustworthy software development.
2) To propose a long term evidence aggregation scheme for software apps and services to quantify trust using theory of belief and sentiment analysis.
3) To represent trust attributes of a software app or service using a trust contract and indicate its application in software selection and recommendation in an online marketplace environment.

4) To modify and improved trustworthy service selection and recommendation algorithms with necessary parallelization for software marketplaces which perform better in precision and recall matrices.

1.3 Thesis Organization

The remaining part of the thesis is organized as follows: CHAPTER 2 presents the related works. Next, CHAPTER 3 includes the background concepts related to this dissertation such as summaries of previous work and related technologies. CHAPTER 4 presents our approach, which includes our contributions and approach in detail. This chapter also defines the trust in relation to a software service and introduces concepts related to "TruSStReMark". As the first part of this thesis contribution, this chapter also provides a summary of the trust-related survey and a case study based trust model for trustworthy software development. The chapter also presents details about the trust contract and the role it plays in long term aggregation and monitoring of trust related attributes of software apps. This section also indicates how the proposed changes to both static and dynamic sections of a service specification and indicates how those changes are useful to improve the service selection and recommendation process. CHAPTER 5 indicates the approach to perform trust-based service selection and recommendation. This chapter also presents an overview of the prevalent selection and recommendation algorithms, highlights their shortcomings, and compares them with the proposed trust-based service selection and recommendation. It also presents the experimentation, results and analysis (of trust-based selection and recommendations algorithms), in terms of HR and ARHR quality matrices. CHAPTER 6 presents the modifications which are needed to parallelize the trust-based selection and recommendation algorithms. It also discusses runtime analysis, which highlights the improvements of the performance of the parallelized versions of both batch and stream algorithms. Finally, CHAPTER 7 concludes the dissertation by indicating the summary of the contributions and also with a glimpse of future directions.

1.4 Thesis Contributions

Based on the above Thesis Objectives, the contribution of this Thesis are listed as follows:

1) Conducted a comprehensive survey and summarized different trust views and evidence-based models of software development and defined what trust of a software service (or an app) is.
2) Proposed a scheme to represent trust attributes of a software service (or an app) using a trust contract in an online software marketplace environment.
3) Proposed an evidence aggregation framework for online software services (or apps) which quantify trust using sentiment analysis and then, propose a technique to convert it to subjective logic to perform aggregation operations.
4) Conducted experiments using the developed framework ("TruSStReMark") to show the improvement in quality (in HR and ARHR matrices) by using the proposed trust-based service selection and recommendation algorithms. When compared with prevent approaches, main advantage of TruSStReMark is that it can capture the dynamic and uncertain nature of software services or apps. This approach presents fine grained trust-based scores about important QoS of apps, so that they can make better choices.

5) Improved the performance of the proposed trust-based algorithms with necessary parallelization and evidence streaming techniques by simulating an online software marketplace.

CHAPTER 2. RELATED WORKS

When considering current the state of the art research, comparisons of techniques used to perform software service selection and service recommendation which use trust related attributes are studied only in a few prevalent publications. However, in general (i.e., not specific to software service selection and recommendation) there exist many prevalent approaches which model trust attributes and specify trust in specifications. In relation to this dissertation, the prevalent works are categorized into four sections.

They are: 1) Related efforts of trust views and trust models in the context of trustworthy software development, 2) Related efforts about the trust contracts and the role they play in long term aggregation of trust related attributes of software apps, 3) Related efforts on trust-based service selection and 4) Trust-based service recommendations. These sections on trust-based selection and recommendation also discuss the related efforts on selection and recommendation algorithmic modifications, end-to-end performance improvements and parallelization of these algorithms. The following sub-sections describes the aforementioned categories of related works. Also in the context of this related work chapter we use both software services and apps interchangeably.

2.1 Trust Views and Trust Models [13]

The discussion of trust presented here is a condensed form of our survey paper [13]. Trust is an important concept that has taken on different views over time.

Any definition of trust is typically either subjective or biased towards the problem under investigation and the context of the application domain. Although this section contains definitions of trust views from other domains, the main goal of this section is to provide the related efforts about trust views and trust models in the context of software systems and services.

Trust is important in many applications and many efforts have provided different views of trust from various domains spanning over the past three decades. Our previous survey (we examined 80 relevant publications from the past 30 years and then extracted different views and definitions of trust) [13] indicates the importance of trustworthiness when considering different domains such as Sociology, Law, Security, Communication, E-commerce, and Software Systems. For example, within the domain of Sociology the trust if defined as -“the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action” [31]. In contrast, in the software systems domain and within the sub-domain of distributed software systems (e.g., in the context of peer-to-peer systems such as file sharing systems) the trust is defined as “an evaluation of the information received about its peers’ service to other peers” [32]. Hence, the view of trust varies according to the domain of interest and intended purpose.

However, irrespective of the domain, in a trust relationship the two parties involved are named as the trustee and truster. Gambetta et al. [31] propose a basic approach for establishing trust relationships between two parties. Their approach is to use either an optimistic view or a pessimistic view. By default, the optimistic view defines all entity relationships are trustworthy, and conversely, the pessimistic view defines all entity relationships are non-trustworthy. In contrast to this basic approach, more complicated approach is proposed by O’Hara et al. [33]. Their approach is to model trust relationships based on an algorithm called the “EigenTrust” which aggregates the local trust scores allocated to each individual entity and yields a global trust vector of all the involved entities. Each entity’s rank in the global vector is based on the recommended trust values from its local peers (e.g., peers with the highest number of local interactions in the past). Therefore, this global trust vector is now frequently updated with the regularized local trust values broadcasted by each entity, which contain details about other entities. However, the challenge of this algorithm is to guarantee convergence and break up conflicts and malicious activities by entities.

One other aspect of viewing trust is to view trustworthiness based entirely on the quantification. Huang et al. [34] view trust as a set of quantifiable attributes expressed as real number and argue that any view of trust should clearly distinguish between three different levels: trust (i.e., the default use of the term ‘trust’), untrust (i.e., the negative opinions of truster on trustee based on negative evidences), and distrust (i.e., the ability not to make a decision about the trustee). This view of trust is more complete than viewing trustworthiness as a value in the range [0, 1] such that 0 represents untrust and 1 represents trust. However, as our survey [13] indicates, there has been a debate in research communities on how to distinguish untrust and distrust in a representation of [0,1]. The confusion arises when there is a significant difference between trust and distrust. Hence, it is necessary to have a proper view of trust that clearly captures the difference. Marsh et al. [35] propose a solution to this problem by representing the trust in the range [-1, +1] where -1 is untrust, 0 is distrust, and+1 is trust. However, the main concern is that when negative values are used to represent trust, the propagation of trust cannot be evaluated via simple multiplication, which is possible when trust is viewed in the range [0, 1].

Also, the view of trust has tacken different perspectives when considering the time of its origin. In 1994, Marsh et al. [35] defined trust as “choosing of an ambiguous path by the truster which could be good or bad, where the occurrence of good or bad is a result of the action performed by an another entity”. In contrast, in 2009, Wang et al. [14] defined their view of trust as “Trust can be simplified to two important factors, i.e., recommendation and competence given from others”. Again, our previous survey [13] indicated that after 2001 there has been an increase in prevalent works related to trust views and models in the software domain. One possible reason for this increase should be the increased presence of various efforts that were aimed at creating generalized models of trust that can be applied to the development of reliable and robust software systems. Hence, considering this aspect of variable view of trust over time and also considering that there are many definitions and views of trust, it is necessary to identify if there exists a commonly agreed upon view and an associated method of quantification that is suitable for software-intensive distributed software systems, especially considering a software service within that complex system (which will be one of the goals of this dissertation).

Having different trust views produces different ways of modeling trust in software systems. Trust models represent the description or portrayal of trustworthiness in a particular context. In order to reason about the trust of individual software or service and the composed software system, as dictated by the requirements, it is necessary to select a suitable and quantifiable model of trust. The trust model explicitly denotes what factors constitute trust of a given entity. Furthermore, the model of trust should be simple yet complete enough to reason about trust of any entity. The trust model is also important when lot of entities are interacting with each other to form a system.

There are many existing trust models in related works, particularly in the subdomains of embedded network systems and peer-to-peer networks. Jouvray et al. [36] define trust in terms of security and dependability and propose a trust model to enforce trust in embedded systems. Their main goal is to avoid communication failures which happen as consequences of attacks, vulnerabilities, and intrusions. The implementation of the model adopts model-driven engineering while proposing their framework for a trust-aware platform, which uses techniques to partition the set of communicating entities present in the embedded applications and decouples entity interactions within a small trusted computing group.

Jøsang et al. [37] provide a formal trust model where trust is seen as a balance between belief, disbelief and uncertainty, which has been recently adopted by many other trust models (including the work proposed by this dissertation). For example, Theodorakopoulos et al. [38] and Herrmann et al. [39] embraced this trust model in their proposed frameworks and highlight its main strength as the easy representation of trust as a tuple, which also inherently includes the uncertainty associated with trust. Moreover, the other main strength of Jøsang's model is the presence of well-defined logical operators of trust such as negation, conjunction, and consensus of trust tuples. These operators allow evaluating, aggregating, and comparing trust values. Improving on the model proposed by Jøsang et al. [37], Herrmann et al. [39] present a trust model which is applicable to the design phase of software services which augments the design of components with trust management aware adapters. Moreover, this approach addresses the production and deployment concerns of the developed software systems by providing a framework for managing and monitoring software compliance with their defined trust policies. Aside from Herrmann et al., there are not many other related works that focus on trust at the deployment and production phase of the software service's lifecycle.

Alagar et al. [40] defined a trust model that primarily targets real-time systems and their trust model defines trust based on safety (i.e., the quality of the operational behavior of the system) and security (i.e., the acceptable quality of the system before, during and after every operation) at the system composition. For example, the evaluation of safety and security of a real-time system is done according to different composing patterns. Alagar et al. [40] also defined a set of trust-related properties at the component-level. Example of these trust-based properties are: event-based security (i.e., authorization of users to trigger a stimulus), and data-driven security (i.e., authorization of users to access data parameters in stimulus).

In contrast, Yan et al. [41] model trust as a tuple of truster, trustee, policy, evidence and opinion. Their definitions of each item in the tuple are as follows: "truster, i.e., the entity who holds a belief that the trustee will provide expected services; trustee, i.e., the entity who has competence to offer the services as the truster expects; policy, i.e., where the subjectiveness of trust is expressed; context, i.e., where any information that characterizes the situation expressed; evidence, i.e., recommendations and quality attributes of the trustee; and opinion, i.e., the trust value held by truster on trustee." Although this model provides trust control and management, the process of collecting evidences, evaluation, and the establishment and re-establishment of trust, primarily focuses on the deployment phase of the software lifecycle. Wang et al. [14] also define a trust model to evaluate trust entirely based on quality-of-service benchmarks. Their model considers the uncertainty, randomness and fuzziness of the trust factors (i.e., reputation and competence of entities). Similarly, AbdulRahman et al. [42] propose a trust model that operates on runtime reputation of entities (i.e., users' ratings, comments and feedback which describes entities in the model).

Also, many of the other trust models in the peer-to-peer networking domain are based on peer reputation and transitivity of trust. For example, Kamvar et al. [43] defined a mathematical model that derives a global trust value for a peer by aggregating local trust values of each peer. Many other related works such as Stakhanova [44], Xiong [32], and Chen [45] have extended the Kamvar et al.'s [43] trust model by suggesting different alternative models. These models can be categorized into three categories based on their characteristics and behavior. Those categories are: 1) Trust views and models based on reputation, recommendation and transitivity algorithms, 2) Trust views and models which are based on graph models where peer evaluations are performed using transitivity and heuristics, and 3) Trust views and models based on peer relationships or past experience and algorithms which calculate the strength of relationships between peers to evaluate the trust of peers.

Considering the different trust views and models identified and analyzed in our survey, this chapter only lists a few important trust views and models. For other trust models are not discussed in this dissertation and we encourage referring to our survey paper [13] for further details. As a result of this survey, we learned that regardless of the application domain or the phases of the software lifecycle, there is no generalized trust view or model. Instead there exist many different trust views and models that address different trust concerns of software systems, such as recommendation, reputation, and management. Also, in the domain of software-intensive distributed systems, it is important to consider various trust-related factors, and the earlier discussion lists these factors according to the truster, the trustee, and other concerns. By knowing these trust concerns during the software life-cycle of the system, the designers and the architects could identify and prevent trust-related issues. As the trust-related aspects are important while developing software-intensive systems and services, the earlier analysis also indicates that there is a need for the creation of a unified trust model that can be used throughout the life-cycle of such systems. Hence, this dissertation presents our approach (i.e., CHAPTER 4) to realizing a complete trust model that is applicable at each phase of an enterprise software service development lifecycle.

2.2 Specification and Trust Contract of a Software Service

A specification precisely describes features of a software, possibly using a formal notation. In the context of distributed software systems, a specification is used to expose or publish different attributes including details about functional and non-functional features to the outside world. These published specifications contain one or more contracts which describe the agreement between trustee and truster, possibly verified by a third party. For example, a software service specification publishes functional and QoS information to interested users of the service and is periodically verified, updated, and maintained, usually by third-party service registries (i.e., in this case online service marketplaces).

Current efforts related to software specifications do not include a trust contract (i.e., motivation and need for a trust contract is presented in CHAPTER 4), and also do not provide details about confidence or trust aspects in relation to other software attributes. Most of the related work in this sub-section takes a simplified view when it comes to expressing trustworthiness of software in their specifications. Also, a majority of the listed related works consider trustworthiness as another attribute in the service specification, hence possibly ignoring trust aspects. They use semi-formal techniques for describing a selected QoS attribute with a corresponding confidence value as a trust contract. Additionally they do not correlate the trust related attributes to the other attributes within the specification and adopt text based representations of software attributes which are expressed as name-value pairs.

Our previous survey [13] indicated that after 2001 there has been an increase in efforts related to trustworthiness in the software domain. One possible reason for this increase is the presence of various efforts aimed at creating generalized models of trust that can be applied to the development of trustworthy software systems. Hence, considering this aspect of changing the view of trust over time and considering there are many definitions, views and models of trust, it is necessary to determine if there exists a commonly agreed upon trust related specification and an associated method of trust quantification that is suitable for software-intensive systems, especially considering a software service within that complex system. Hence, the creation of a trust specification for a software service that can contain multiple trust representations for different attributes, which is one of the goals of this dissertation.

Since this increased attention was given to the trust-related concerns beginning in the early 2000s, and the Trusted Computing Group (TCG) [15] was created in 2003. The TCG initiative was driven and backed by key partners from industry such as AMD, HP, IBM, Intel, and Microsoft. Their main objective was to produce a set of specifications that standardize the commonly used trust-related scenarios including risk management, asset management, E-commerce, security monitoring, and emergency response of a software entity. The first version of the specifications was released in 2008 [15]. The accepted version of this specification from the International Organization for Standardization is called the “Trusted Platform Module” and it contains four modules (i.e., Overview, Design Principles, Structures, and Commands). Over the years, the TCG has shifted the focus of their specification towards cryptographic protocols and hardware standards and now holds the international standard for a secure crypto-processor (i.e., a dedicated microprocessor) designed to secure hardware by integrating cryptographic keys into devices.

Even before this standardizes set of specifications was proposed by the TCG, an initiative named the Trusted Components Initiative was formed around 1998 by Meyer et al. [46]. The main goal of the Trusted Components Initiative is to propose techniques based on a principle called “proof by program construction” [46]. This method advocates several best practices of software development such as object-oriented techniques, design by contract, software component reuse, public scrutiny (i.e., source codes of the components are freely available and open for contributions and criticisms), extensive testing (i.e., to take advantage of design by contract and reuse components), and metric efforts (i.e., to monitor components’ properties in a controlled fashion and ensure trust). This allows the software to be validated using derived proofs at the time of construction instead of waiting until the testing phase to perform such validations. However, this effect does not address the dynamic nature of trust with software updates, which is a major drawback.

In trusted software systems, a specification is different from the trust model because the specification contains a different interpretation of the trustworthiness of its published attributes. Few prevalent approaches consider trust-based specifications. Grandison et al. [47] propose a trust-oriented specification called the Simple Universal Logic-oriented Trust Analysis Notation (SULTAN). The SULTAN specification uses a logic-based notation that allows the users of the system (i.e., the truster, trustee, or trusted third-party) to define trust requirements, and personal recommendation statements. They also claim that the SULTAN specification is the first step towards a framework designed to facilitate analysis and management of trust relationships by using the proposed specification as the common interface. Also, the webservices Trust specification (WS-Trust) [48], is a part of the WS-* specification set that is considered the industry standard for webservices interoperability by the webservices consortium. WS Trust is the first attempt to standardize how meta-data about trust attributes is published. This specification also addresses issuing, renewing, validating security tokens, and providing the means to establish and assess trust relationships between participants using secure message exchange. Comparing these two methods, the SULTAN specification was never adopted since it involved many complexities and manual intervening with ambiguities while representing attribute trustworthiness.

To compose a trusted distributed system, the trustworthiness of the functional and non-functional requirements of each service taking part in that system should be considered based on a specification and its underlying contracts. Yan et al. [41] proposed requirements for the trust specifications and contracts using threshold values. These values indicate the lower bound for user ratings that are required for an entity to be trustworthy. In contrast, Alagar et al. [40] proposed requirements for the trust contracts using a set of conditions and properties related to safety, security, and timeliness. All these conditions and properties must be satisfied by the software components to ensure trustworthiness of the system.

Any software design should involve proper documentation of the system, its sub-systems, and their interactions with respect to a set of requirements. During the software life-cycle, these design documents (i.e., the design artifacts) become a part of a specification associated with that system. Hence, these specifications also contains the underlying set of facts and rules of the system that can be collected, organized, shared, searched, and utilized over time. For example, Uddin et al. [49] propose a trust-aware development framework for building trusted systems. They introduce a Unified Modelling Language (UML)-based specification called UMLTrust which specifies trust scenarios in the design phase of the distributed system. Weth et al. [50] and Stakhanova et al. [44] have proposed two different approaches for realizing a trust knowledge-base: centralized and decentralized. In the centralized trust knowledge-base, the trust-related data are collected, stored, and organized at a central location. Weth et al. [50] further indicate the importance of keeping trust factors in a central queryable knowledge base. They propose an SQL-like query language, which is parameterized by trust requirements, and allows trusters to query the trust factors based on their needs. In contrast, Stakhanova et al. [44] and Sherwood et al. [51] stored evidences at different locations to evaluate trust of peers in P2P networks. In their approaches, meta-data about peer interactions and related trust factors are stored locally by each peer, and the trust evaluation algorithms collect information from associated peers when evaluating the trust of a peer. However, centralized solutions do not scale in open systems because malicious peers can take control of the central servers. In distributed solutions, peers can independently co-operate in the trust evaluation process with their own policies, which can protect against malicious peers.

A typical service specification now contains multiple levels of contracts. This was first advocated by Beugnard et al. [52]. They have proposed four distinct levels in the specification of a service contract, namely syntax, semantics, synchronization, and QoS. In CHAPTER 4[52], Figure 4.6 indicates a modified version of their proposal with Level 0 indicating the type and other inherent attributes before their four levels. These levels may be negotiated at varying degree from non-negotiable at the type and syntax levels to highly-negotiable at the QoS level. Multi-level specifications do offer more details about the services than the basic contracts. However, they still do not include a formal quantification of the trust values. The dynamic nature of software (i.e., frequent updates) and fuzzy nature of the underlying environment makes software features and qualities change over time. As a result, the trust of the software also fluctuates over time. For example, this dynamic and fuzzy nature of software is visible in emerging software marketplaces (e.g., Android-based mobile applications that are available from the Google Play Store [5]). Over time, the software in such marketplaces is updated and frequently becomes pushed to the clients. Hence, it is hard to determine whether a service actually delivers what it promises by merely inspecting its contract. One possible solution to overcome this issue is to define a separate trust contract for each service with appropriate aggregated information about various evidences related to its multiple levels, which is one of the goals of this dissertation.

2.3 Trust-based Service Selection

Reusing software entities is a well-established practice in the software development process. Hence, most software systems re-use third-party software applications and services. If a new software entity with similar features to an existing entity is found and also the new entity provides additional desired features (e.g., higher QoS), then the new entity found could replace the existing one. However, due to frequent updates, the software system should adapt to the dynamic nature of entities or shared libraries constantly being updated, possibly at run-time. Service selection is the process which finds these new entities to be updated. Service selection involves comparing suitable candidate services to estimate possible similarities and differences between them. Hence, selection is the action of choosing an entity as being the best or most suitable for a certain need. Trust is an important factor in this process and the trustworthiness of entities needs to be evaluated. Hence, the service selection process can also be improved by identifying the most trustworthy software entity with a suitable technique for comparing trust-related factors among many other choices that provide a similar functionality and behavior. Few prevalent works study the impact of trust on service selection on automatic service composition and system generation.

The work proposed by Liu et al. [53] tries to capture and use valuable personal evaluations of services. It facilitates the expansion of a service-oriented network into a trust-based, self-organizing virtual collaboration environment. This framework is based on personalized service selections which couples with social network analysis targeting service attributes. Also, this approach views a service-oriented network (i.e., a Grid network or the World Wide Web) as an ecosystem and considers the trust evaluation and propagation as the fundamental driving forces for service selection and then use them for composition. This approach expands to current e-service selections from service registries to make better decisions. However, they note that, it is not possible to capture all possible evaluations and use them without employing intelligent planning for composition, which fails to scale in a large open environment such as the World Wide Web. Hence, they consider trust evaluation and propagation as the fundamental driving forces for service selection and composition, and the proposed framework includes self-organizing capabilities based on trustworthiness in their simulated collaboration environments.

In contrast, the work proposed by Limam et al. [54] describes a rating based framework for reputation-aware software service selection. Here, the main use of trust is to compare trust of entities from a set of candidate entities that perform the same or similar functions and select the one most suitable for a particular need. This can be achieved if the trust values of these entities are comparable with each other, hence their trust representation must be a quantified value. Their service selection algorithm is based on an automated rating model and market science which uses service quality and cost to overcome feedback subjectivity issues. The proposed rating and selection algorithms are validated through simulations, demonstrating that the system can effectively capture service behavior and select the best possible choices. Their work matches the work proposed by this dissertation, however their solution is static and targets specific needs. This could not be used as a trust aggregation and selection method in general and is more suitable for the dynamic nature of a service marketplace.

The work proposed by Hang et al. [55] develops a framework for service selection by promoting trust as the matching scheme that is based on probability distributions and graph theory. The trust aspects of service descriptions are based on the beta probability distribution. Their selection scheme includes two approaches for matching trust attributes: 1) Bayesian networks (which captures the dependency of providing required service quality between other services using acyclic graphs techniques), and 2) beta-mixture mode (which models events that are constrained within pre-defined ranges). When their approach is compared to the evidence-based approach proposed by this dissertation, it is clear that subjective logic based operations do not need any prior knowledge, rather they could aggregate evidences dynamically. Hence, the main drawback of their approach is that some level of prior knowledge is needed about the evidences plus subjectivity and uncertainty of trust are not considered as inherent features.

Similarly, the framework proposed by Yan et al. [56] indicates an approach for selecting software services based on a user-centric perspective, especially from the service consumers' viewpoint. Service consumers' rate software services, and these ratings are incorporated into a software service's reputation. This method classifies reputation into two different categories: 1) local reputation, which is the entity's own assessment based on past history of direct interaction with the particular entity, and 2) global reputation, which is the aggregation of all available assessments by other entities that have had interactions with the particular entity. The two kinds of reputation are then used to generate a reputation metric model that is used to assess the level of trust an entity puts into another entity. Although there are different variations proposed with service reputation, the drawback of this approach is that the only trust related operation used in service selection was reputation. Plus, their approach considers only the static nature of evidences, including ratings and reviews and does not incorporate uncertainty as a part of their approaches.

Li et al. [57] propose a framework which employs a trust-based service selection in service oriented environments. They claim that sometimes a webservice’s reliability is unknown to the service users in current service-oriented environments. They also note that reputation-based systems are popular now because they are able to produce a global score of quality of a service provider to requesters. However, users cannot solely rely on such global information to choose the most qualified services. They tackle this problem by presenting a trust-based architecture with a computational trust model for quantifying and comparing the trustworthiness of webservices. In their framework, the first step is to construct an entity network, which uses the direct trust relations between entities based on the reputation and rating similarity. The next step in their approach is to execute an algorithm for propagating trust in this network, and to produce personalized trust information for service requesters. Experiments are carried out after their implementation of the trust model to simulate various malicious behaviors in not only dense but also sparse service networks. The goal of their experiments was to verify the attack-resistance and robustness of the proposed approach and the experimental results demonstrate the feasibility and benefits of their approach. The drawback of this approach is to assume reputations of services are static, however, in reality this is not true when services get updates dynamically. The framework proposed by this dissertation does not have that assumptions and the proposed trust model consider dynamic nature of the trust of services.

Su et al. [58] present their work on trust-based group service selection in web-based service-oriented environments. They state that multi-agent (i.e., a collection of services which forms a service group) technologies are adopted for the development of many web-based systems such as e-markets, grid computing, and e-government web portals. In this dynamic and open environment with autonomous agents, their work addresses the challenge of how to select the most suitable service groups according to a particular service request. Their selection framework includes a trust model named as the GTrust model. This model is mainly for service groups and the selection mechanism focuses on selecting a service group in general. The GTrust model's trust evaluation is based on the functionality of the group and the dependency relationships among individual services in the group. The selection process uses the ratings of individual services on the attributes of the service request and a similarity measurement of the extent to which reference reports can reflect the service request in terms of the priority distribution of attributes. When compared to the prevalent approaches, their results and analysis indicate an improvement in performance of the GTrust model on service group selection in service-oriented environments.

In summary, service selection is an important part of the development process for a trusted distributed system where software reuse is becoming the norm. Although, only few publications exist which evaluate trust while performing software service selection, they only consider static evidences. Hence, they fail to capture the important dynamic nature of trust with services being updated frequently and environmental fluctuations. Without the long term monitoring and aggregating trust of software services as proposed by this dissertation, it is hard to achieve the benefits and capture the subjective nature of trust. In addition, these models are not capable of providing additional information at the time of service selection with different levels of abstraction (i.e., at overall service level, at the level of a collection of functional attributes and at the single attribute level). In contrast, the approach proposed by this dissertation considers the subjective and dynamic nature of trust, provides long-term monitoring and aggregation of trust attributes and provides information at different levels to effectively perform trustworthy service selection.

2.4 Trust-based Service Recommendation

The discussion presented here is a condensed version of our previous paper [59]. When considering the domain of recommendations, there are many attempts at creating recommendation systems for physical products, but recommendations involving software services are studied in very few publications. Also, there are some related efforts on trust-based searching, ranking and recommending specifically for software services. However, there are many efforts focus on creating different techniques to improve collaborative and content based filtering algorithms. Hence, the following section only describes the related trust-based service recommendation approaches while comparing them to the approach proposed by this dissertation.

The work proposed by Das et al. [59] experiments with selecting online news based on an offline clustering method with reservoir sampling. Their content-based approach uses clustering and does not consider the relative importance of each service. However, it does consider the trustworthiness of the service by calculating from their QoS values. The work proposed by Zhou et al. [60], a real time search and recommender system targeting microblogs which uses only user tags (i.e., hashtags) to compute the similarity. However in the context of software, it is not sufficient to consider the service type and description tags only. Zhang et al. [61] propose a search and ranking model for scientific publication networks. Their trust model is called Knowledge-Social-Trust (KST) and it draws upon various scientific publication repositories (e.g., DBLP) and social networks (e.g., Twitter).

The Knowledge-Social-Trust is a graph-based network model which calculates the relative importance of citations from various publication repositories. Their notion of services comes from the scientific domain while ours is targeted for service marketplaces. In a service marketplace, there is no notion of citing or referring one service by another. However, their trust model can be adopted to our context based on their interest in the QoS in a service marketplace.

Similarly, an approach is presented by Tang et al. [62] where both the user-service relation and the user-user social relation are explored. Our approach of reviewer-reviewer correlation is similar to their approach, however, their social trust is based on social networks and ours is based on sentiments of their written experiences about services. The work proposed by Lin et al. [63] experiments a social-trust-based recommendation mechanism for binding and accessing information from webservices and then using it to recommend trustworthy webservices. It uses legally binding information of webservices based on the social trust of the enterprises to build a trust network. The difference between our approach and theirs is that we consider individual service features and attributes, while their approach considers confidence about a webservice as a whole. Therefore, our approach is at a finer granularity than theirs and thus provides a more comprehensive technique.

Gupta et al. [64] explain Twitter’s recommendation technique, which employs an algorithm and uses tweet tags which are represented as a set of adjacency lists in a social graph. The search and recommendation then becomes a lookup and intersection analysis of these lists. Our approach of service-reviewer correlation is similar to their tweet-tag, however, their trust is based on social network analysis (i.e., followers) and ours is based on sentiments of written experiences about services (i.e., from service users). Also, the work proposed by Zhang et al. [65] discusses a temporal and QoS-aware webservice recommendation using non-negative tensor factorization. Their goal is to address the issues related to QoS-aware webservice recommendations with the rapid growth of webservice in the past decade. Similar to our approach, they mention that QoS information collection requires much time and effort and even impractical, hence these service-QoS values are usually ignored or missing. Compared to the prevalent approaches used to predict the missing QoS value through traditional collaborative filtering methods based on user-service static model, they highlight that the QoS value is highly volatile and often related to the invocation. Hence, their work considers the dynamic contextual information to come up with a temporal QoS-aware webservice Recommendation Framework that is capable of predicting the missing QoS values under various temporal contexts. They formalize this approach as a generalized tensor factorization model and propose a Non-negative Tensor Factorization (NTF) algorithm which is able to deal with the triadic relations of a user-service-time model. Their experiments and results are conducted based on a real-world webservice QoS dataset collected from Planet-Lab (i.e., comprised of service invocation response-time and throughput value from 343 users on 5817 webservices at 32 time periods). Their experimental analysis shows that this approach achieves better prediction accuracy than other prevalent approaches. However, their dataset was very small and does not reflect real world scenarios from online marketplaces such as Google Android Apps Marketplace [5].

Webservices are heavily used to design new approaches, hence better recommendation of webservices recommendation has become critical [66]. Most recommendation approaches use UDDI registries and set of keywords hence these techniques have drawbacks. For example, heavy dependence on user inputs such as frequent activities and submission of detailed queries. Hence, the work proposed by Yao et al. [66] recommends webservices by combining collaborative filtering with content-based features. They propose a novel approach that dynamically recommends webservices that fit users' interests. This approach is a hybrid system in the sense that it combines collaborative filtering and content-based recommendation. In particular this model uses a three-way approach, such that it simultaneously considers both rating the data, the content of webservices and unobservable user preferences (which are statistically estimated and are represented by introducing a set of latent variables). To verify the proposed approach, they have conducted experiments using around 4000 real-world webservices. Results show that their framework performs better, mainly on recommendation quality and performance, when compared to prevalent conventional approaches such as collaborative filtering and content-based filtering recommendation. This approach is similar to our approach however this approach does not contain how these algorithms can be parallelized or could perform efficiently in a real marketplace. The approach proposed by this dissertation include details about parallelizing the hybrid approach with results and their analysis.

Similarly, Chen et al. [67] propose a method of webservice recommendation by exploiting contextual details such as location and QoS. Their focus is on integrated software components, which support interoperable node-to-node interactions over a large network. They also indicate these kinds of webservices have been widely used in recent years for building service-oriented applications in both industry and academia. Their views is that the number of publicly available services is steadily increasing in the Internet, which is similar to our view too. The challenge for the user is the complexity to select a suitable webservice by searching a large set of candidates. To address these issues, Chen et al. [67] propose a collaborative filtering based webservice recommender system to select services with improved performance. Our recommender system also employs the location information and QoS values to cluster users and services, and makes personalized service recommendation for users based on the clustering results. Compared with existing service recommendation methods, their approach achieves improvement on the recommendation accuracy, however they fail to address the quality of the recommendations. Their comprehensive experiments involve more than 1.5 million QoS records of real world webservices to demonstrate the effectiveness of their proposed approach. When compared to the approach proposed by this dissertation, negligence of the quality of recommendations over accuracy is the main drawback of their approach.

In summary, service recommendation is an important part inside the ecosystem of online software marketplaces. Similar to the number of trust-based selection approaches, few recent publications exist on trust-based recommendations. They either treat trust as yet another attribute or they do not indicate trust relations to other existing attributes and update them dynamically as other attributes get updated. Also, these approaches only consider the static nature of evidences and contextual information. It is hard to capture the subjective nature of trust, hence, these prevalent approaches are not capable of capturing additional levels of information at the time of service recommendations. Since they fail to capture the important dynamic nature of trust, the approach proposed by this dissertation uses long-term monitoring and aggregating of trust of software services. Our approach also considers the subjective and dynamic nature of trust, aggregation of trust attributes, and provides information at different levels to improve the quality and effectively perform trustworthy service recommendation.

CHAPTER 3. BACKGROUND RELATED TO OUR APPROACH

The following two sections present a summary of our past work and background concepts. The prevalent work section contains the related work of my M.S. Thesis and the background section contains a summary of related concepts which to this dissertation. Also in the context of this this chapter we use terms services and apps interchangeably.

3.1 Past Work

The discussion presented here is a condensed version of my MS Thesis [69]. Demanding applications have forced the transition of the computing paradigm from a centralized approach to a distributed approach and this shift has led to the concept of Distributed Computing Systems (DCS). Traditional way of software development lacks the ability to address the challenges in software realization of large scale DCS. Out of many methods proposed to develop DCS, one main approach is the Component Based Software Development (CBSD). The UniFrame approach [68], an approach developed at IUPUI, follows the concepts of CBSD and addresses the design and integration complexity of DCS.

3.1.1 URDS and Multi-level Contracts

Discovering appropriate services from a set of available candidate services set is an essential step in developing distributed systems that are composed of individual services.

Various techniques, ranging from simplistic attribute comparisons to more complex multi-level matching, can be used for matching a query against a set of service specifications. The UniFrame approach [68] provides a comprehensive framework which enables the discovery, interoperability, and collaboration of components via generative software techniques. It uses existing and emerging distributed component models to form a common meta-model. This framework enables the creation of high-confidence DCS using existing and newly developed distributed heterogeneous components. One essential part of UniFrame is the UniFrame Resource Discovery Service (URDS) [69]. The URDS is a hierarchical discovery system that uses the principles of multi-level specifications and associated matching.

3.1.2 Service Discovery and proURDS

Due to the limitations of the simple attribute-based representation of contracts and basic textual matching, the URDS uses the concepts of Multi-level contract representation and Multi-level Matching (MLM) [73]. The URDS contract provides information at many levels including: the General, Syntactic, Semantic, Synchronization, and QoS. Matching of component contracts is performed according to the appropriate matching operations proposed for each of these levels. This narrows down the search space according to the individual requirements at a corresponding level. Hence, based on each operator's capability, related components have a better chance of being included in the result list. As described in) [69], [70], and [73], the proURDS was developed as a distributed setup by enhancing the URDS architecture which was deployed over the network with real component contracts.

The proURDS was experimented and the results were presented in [73] which indicated an improved architecture from the previous version of URDS. Two new modules (namely, the Knowledge Base (KB) module and the Service Management and Monitoring (SMM) module) of proURDS enhance the URDS architecture by providing necessary domain knowledge and control, management and monitoring capabilities. Also the case study has presented empirical validation of the proURDS using environmental science services was performed in [72] and [73]. It also compared the performance of the proURDS with jUDDI [71], which is one of the popular public discovery service implementation.

The results described in [73] indicate that the proURDS returns relevant services (i.e., services with better quality) as a result of the multi-level matching semantics at the cost of increased response time. The proURDS referred to a public dataset called QWS dataset [74]. This dataset includes actual information of software components (i.e., webservices), which were harvested from the Internet. The quality of the services returned by the proURDS is measured in terms of precision and recall. Although the average response time of the proURDS is high, this was expected due to the extra work performed by the multi-level matching. This dissertation builds upon the experiences obtained from the proURDS based work to investigate service specification, selection and recommendations in the context of trustworthiness.

Our research group have also analyzed how learning techniques could help service discovery and selection. In [75] we discuss the effects of using learning profiles in the search algorithms of the URDS. The experiences gained from improving this solution for efficiently discovering services using learning techniques has provided additional background for this dissertation.

Our research group have also worked on another service discovery technique [76] which combines the principles of multilevel matching with the reinforcement learning techniques. This discovery process selects services dynamically using criteria based on performance which considers the reinforcement feedback. The entities in this system are not only distributed in nature but also they randomly join and leave the system. Hence, learning of preferred acquaintances are encouraged, which intern helps to improve the capabilities of matching. Main goal of this work was to eliminating a large amount of redundant computation. Experimental results are presented using an information classification system and they show performance improvement and reduction in computational cost. The experience gained from improving the performance and the computation cost of this solution has also provided the necessary background for this dissertation.

3.2 Related Background Concepts

The following sub-sections describe the main concepts that are related to this dissertation work: These concepts are related to 1) Theory of Evidence – which is used as the basis by our approach in long term monitoring and aggregation of evidences related to the trustworthiness of services, 2) Subjective Logic – which provides a mathematical logic to capture quantified values of evidences with a rich set of operators to reason about the trustworthiness of services, 3) Sentiment Analysis – which is used to capture expressed sentiments about evidences by users of services inside the proposed framework, 4) Content-based filtering – which is used as the basis for service selection algorithms and our approach enhances it by incorporating trust-based concepts, and 5) Collaborative

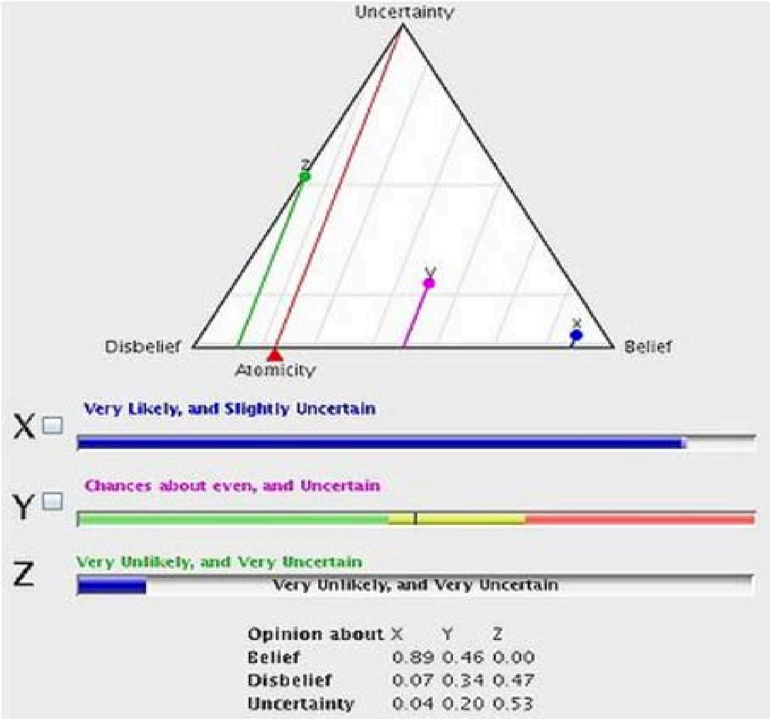
Filtering – which is used as the basis for service selection and recommendation algorithms and our approach improves it by incorporating trust-based concepts.

3.2.1 Theory of Evidence [22] and Subjective Logic [23] [77]

(1) The Theory of Evidence (ToE) [22] is a framework for quantifying evidences with uncertainty [77]. It enables to represent and aggregate difference evidence and quantify these set of evidence in to tuples of belief, disbelief and uncertainty. However, following this theory can result belief values which contradict with those values arrived at using the prevalent probability theories. This theory is an improvement from the original version of the theory proposed by Dempster and Shafer [77]. Many authors also have proposed different rules for combining set of evidence, and also extending this work to handling conflicts in collection of evidence. This theory also started other fields on its own such as, the Theory of Hints [78]. Today, one of the main applications of Theory of Belief is provide a reliable mechanism to fuse different set of evidence from various sensors. However, the main part of this work was initially applied to the legal domain to predict the outcome of court hearings and which as a result the theory was called the Dempster-Shafer theory [77].

Also, the ToE indicates that the degree of belief in a given proposition depends upon the number of evidences related to that proposition [22]. The theory provides a set of rules which are used to combine evidences about two related proposition in a particular system. The work proposed by this dissertation, the TruSStReMark framework (i.e., including the model and algorithms presented in CHAPTER 4) proposed by this dissertation base the related work with evidences related to software services on the basic theory and approach explained above. This dissertation uses the theory of evidence concepts to quantify and aggregate trust and also use them to improve selection and recommendation of services in an online marketplace (i.e., TruSStReMark framework presented in CHAPTER 4).

(2) The Subjective Logic (SL) [23] is based on probabilistic logic which takes belief and uncertainty while its calculations. It also defines a set of logical operations on evidences. In a subjective logic propositions, the arguments are a set of subjective opinions about that particular proposition. When a collection of propositions are present, the subjective logic provides an algebra to combine different opinions which are based on the belief functions.


Uncertainty
Z
Y
X
Disbelief
Belief
Atomicity
X
Very Likely, and Slightly Uncertain
Y
Chances about even, and Uncertain
Z
Very Unlikely, and Very Uncertain
Very Unlikely, and Very Uncertain

Opinion about	X	Y	Z
Belief	0.89	0.46	0.00
Disbelief	0.07	0.34	0.47
Uncertainty	0.04	0.20	0.53

Figure 3.1 Subjective Logic tuple notation of Belief, Disbelief and Uncertainty [23]

Capturing the uncertainty of evidences is the main strength of the SL and thus it improves the well-established Boolean logic to include uncertainty. Due to the human nature of suspicion, there is no absolute certainty whether a proposition about the real world is true or false. Whenever the truth of a proposition is expressed, it is always done by an individual, and it can never be considered to represent a certain belief on that proposition. Figure 3.1 presents sample representations of both graphical and numerical subjective logic tuples which are expressed as Belief, Disbelief and Uncertainty values.

In Subjective Logic the opinions are represented on a triangle as shown in Figure 3.1. A point inside the triangle represents a (B,D,U) tuple. Within this triangle, a very strong positive opinion is represented by a point in the bottom right section. Figure 3.1 also indicates three sample propositions X, Y and Z which are visualized on the triangle with their corresponding numerical values and verbal descriptions of each opinions (i.e., very likely, chances are even and very unlikely). There are three special points in that opinion space. They are i) the full Belief B = (1,0,0), ii) the full Disbelief D = (0,1,0), and iii) full Uncertainty U = (0,0,1).

Apart from the tuple representation, the main concept in Subjective Logic is its operators. Most operators of the SL are based on the corresponding binary logic operators which are then generalized to include uncertainty as an inherent feature. The main operators in SL are Union (improved from Addition operator), Difference (improved from Subtraction operator), Conjunction (improved from AND operator), Disjunction (improved from OR operator), Consensus and Negation (improved from NOT operator). The main advantage of Subjective Logic operators is that if needed they can also preserve related attributes (i.e., the subjects and the contexts) intact while doing the computations.

The implementation of the operators depends on the application. For example, in sensor fusion, it is assumed that two separate items are fused into one using the consensus operator which is preferred over Union and Conjunction. The consensus operation merge two opinions into one opinion by evidence aggregation. In the evidence space, the consensus rule can only be applied if the evidence underlying x and y is independent. Hence, the SL allows efficient computation of mathematically complex models and applications. For example, this approach is suitable when the uncertainty is needed to be carried through and it enables to distinguish between certain and uncertain conclusions. The TruSStReMark framework proposed by this dissertation base the algorithms which model and aggregate evidences which are related to software apps on the subjective logic based operators. More details about the TruSStReMark framework is presented in CHAPTER 4.

3.2.2 Sentiment Analysis Concepts [25] [83]

Sentiment Analysis (SA [25] a.k.a. opinion mining) refers to the use of Natural Language Processing (NLP) techniques to analyze textual content and use computational linguistics criteria to identify and subjectivity (s) and polarity (p) information. It aims to quantify the tone of a writer. The sentiment of a textual sentence is presented using a numerical pair of (p,s). The values of p and s each represent the polarization (i.e., the confidence about an identified named entity) with a value between -1 and +1 and the subjectivity (i.e., the weight of the expressed polarization) with a value between 0 and +1.

The polarity of a given sentence of a document highlights the opinion which would be either positive, negative, or neutral. In contrast, subjectivity is difficult to calculate. The subjectivity of words and phrases usually depends on their context and an objective of the document. For example, calculating subjectivity becomes more complex when the document contain subjective sentences, like a document quoting people's opinions. One way to avoid this complexity is to identify objective sentences from a document before classifying its polarity. One other way to aggregate sentiment from text is to scale the values where the words are associated number on a range from -100 to +100.

Main problem of sentiment analysis is the detection of stop-words. The accuracy of a sentiment analysis system is usually measured by precision and recall matrices. Therefore, some sentiment analysis tasks can return a scalar rather than a binary value. Although there are recent advances of NLP techniques, the sentiment analysis is still considered as a challenging problem. One popular application of SA is to calculate sentiments based on different features of entities. The automatic identification of features can be performed either manually by using a domain expert or automatically using a topic modeling techniques such as TF-IDF. Basic sentiment classifications to go beyond polarity/subjectivity tuples to quantify emotional states such as sadness and happiness.

More advanced techniques are capable of finding the owner (i.e., the entity which the effect of the sentiment is felt) of a sentiment [77]. Moreover, specific classifiers such as the Max Entropy and the Support Vector Models [83], can be used to improve the overall accuracy of the sentiment classifications. Other hybrid approaches [83], use ontologies and knowledge representations. Also, SA is widely applied to reviews and social media for many applications such as advertising, marketing and customer service. The TruSStReMark framework proposed by this dissertation uses sentiment-based techniques to quantify evidence available in online marketplaces. It identifies a set of named entities for users from storing their queries (e.g., which QoS features are important to them). These named entities for apps can be calculated from the keywords, their description, and from user reviews. Using TextBlob library [78] (i.e., a popular NLP library which could calculate sentiments efficiently), the framework analyzes textual reviews from users to calculate the sentiment expressed about QoS properties of a software apps. More details about the TruSStReMark framework is presented in CHAPTER 4.

3.2.3 Content-based and Collaborative Filtering [16] [17]

The concepts of selection and recommendation have become popular in recent years, and are heavily used in domains such as movies, news, and products in general. These techniques produce a list of ordered items in either from collaborative filtering [16] or content-based filtering [17] or hybrid of these two techniques [83].

(1) Collaborative Filtering **(CLF) [16]**. The discussion presented here is a condensed and near-verbatim form of information in Wikipedia [16]. The CLF approaches build a model from a user's past behavior (i.e., items previously selected or purchased with optional numerical ratings given to those items). This technique also considers similar decisions made by other users. This model then uses that to predict items the user would be interested to use. Hence, the collaborative filtering technique is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past. These methods typically collect and analyze large amount of information especially on users’ behaviors, their activities or preferences. From these collected information, then CLF predicts what these users will like in future by analyzing their similarity with other users.

To calculate the similarity, CLF uses many algorithms to measure user (or item) similarity. Popular similarity algorithms are K-nearest neighbor, Cosine Correlation and Pearson correlation based similarity algorithms. One of the popular example of collaborative filtering is item-to-item collaborative filtering which is an algorithm heavily used by industry leaders such as Amazon, Facebook, Twitter and LinkedIn [84]. The main use of this techniques is to select and recommend new entities, groups, and other social connections by examining the network of connections between entities. Despite many advantages, this technique often suffers from three main problems. They are cold start, scalability, and sparsity. The problem of cold start happens when the systems require a large amount of existing data on a user in order to make accurate predictions. Also, the algorithm need to work on millions of items, hence, the scalability of the environments is an issue to produce real-time results. If we consider the number of items exists, this number is extremely large but most users can only rate a subset of the items. Hence, sparsity is also an issue. Considering the above mentioned drawbacks of both content-based filtering and collaborative filtering, these approaches are often combined to produce hybrid techniques.

***(2) Content-based Filtering* (CBF) [17].** The discussion presented here is a condensed and near-verbatim form of information in Wikipedia [17]. The CBF techniques operate on a series of label attributes of items. Then CBF use these information to select and recommend additional items with similar properties. All content-based filtering approaches uses item descriptions with a profile of the user's preference. These descriptions are then reduced to set of keywords which are then used to describe these items. Also, the user profile descriptions are reduced to a set of keyword, which is represented as a vector, which represents the type of items that each user prefers.

To extract the features of the items in a particular system, certain algorithms are applied and one widely used algorithm is the tf–idf [84] (i.e., term frequency and inverse document frequency). In tf-idf, all the features are represented using a fast accessible and storage efficient technique such as vector space representation. For example, creating user profile tf-idf model of the user's preference. The CBF algorithm creates a content-based profile of users based on a weighted vector of item features. These weights represent the importance of each feature to the user. These values of weights can be adhered from individually rated item vectors using a variety of techniques. For example, in order to estimate the probability that the user is going to like or use the item, a naïve approach could use the average values of the rated item vector while other complex approach could use machine learning techniques such as clustering or artificial neural networks. Main issue with content-based filtering is how accurately that the algorithm was able to learn user preferences from users' actions. Direct feedback from the users are highly valuable for success of this techniques. For example users clicking like (or thumbs up) button is to assign higher weights. Novelty of the recommended items is also very valuable. If the CBF is only recommending same content that the user is using now or used in past, then the those recommendations are not valuable for the users.

(3) Hybrid Approach (HA) [84]. The discussion presented here is a condensed and near-verbatim form of information in Wikipedia [84]. The HA combines collaborative filtering and content-based filtering to build techniques which are more accurate and effective. Naive way to develop a hybrid approach is to perform CBF and CLF predictions in parallel and then merge them. When compared with the basic collaborative and content-based methods, the hybrid techniques usually improve the results in both quality and quantity. The advantage of hybrid method is that it could overcome the common problems in CBF and CLF systems such as cold start and sparsity.

Each type of selection and recommendation technique (i.e., CBF, CLF or HA) has its own strengths and weaknesses. For example, some techniques would requires a large amount of information on a user in order to make accurate predictions and others do not need such large collection of data. The TruSStReMark framework proposed by this dissertation uses a hybrid approach to simulate trust-based selection and recommendation in online marketplaces. The framework improves the basic content-based and collaborative filtering techniques to include trust-based augmentations based on expressed QoS properties of a software apps. More details about the TruSStReMark framework is presented in CHAPTER 4.

3.2.4 Big Data Analytics Concepts [79]

(1) Big Data [79]. The discussion presented here is a condensed and near-verbatim form of information in Wikipedia [79]. The Big Data is the term for dealing with large data sets that are complex than traditional datasets. Therefore, the traditional data processing techniques are insufficient to work with them. Analysis of these large data sets can often g find new information to understand trends and insights. Since data storage is cheap, large amounts of data is generated each day. Once example where large volumes of data is generated is online software app marketplaces, for example, Android and Amazon marketplace where software-based services and apps are hosted. Whenever a new platform is released (e.g., updates done to the hardware and software of the platform such as android platform enabling its new features to use the newest sensor of a device), all existing services and apps tend to release a new version targeting this new platform update. Hence, it generates lot of artifacts and data, such as community reviews about each versions of these apps presenting a big data analysis challenge.

The definition of Big Data is being updated each day, one accepted definition is from Gartner, that is "Big data is high volume, high velocity, or high variety (i.e., also knows and 3Vs of Big Data) of information which require new forms of processing to enable enhanced decision making" [79]. The definition of 3Vs are: volume (i.e., amount of data), velocity (i.e., in and out speed of data), and variety (i.e., heterogeneity of data types). Additionally, a new V for Veracity (i.e., quality and uncertainty of data) which is added by some organizations to describe it and making it 4Vs to describe the challenges of Big Data.

The Lambda Architecture (LA) [80] is coupled with the growth of big data and real-time data analytics. This architecture is heavily used in Business Intelligence (BI) use-cases. The usage of LA is to perform data-processing, which is designed to handle very large quantities of data by taking advantage of both batch- and stream-processing methods. The main goal of this architecture is to balance latency, throughput, and fault-tolerance by using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide views of online data. The TruSStReMark framework proposed by this dissertation gathers and analyzes external user reviews generated by online marketplaces which is now considered as a big data challenge. More details about the TruSStReMark framework is presented in CHAPTER 4.

(2) MapReduce, HDFS and Hadoop Ecosystem [28]**.** The MapReduce is a distributed programming model together with an associated implementation for processing large data sets using parallel, distributed algorithms on large computing clusters. A MapReduce program consists of a Map Task (i.t., which performs filtering and sorting of data) and a Reduce Task (i.e., which performs a summarizing operations). The MapReduce software framework uses the provided hardware infrastructure to orchestrates parallel tasks on these distributed computing nodes, while managing all communications and also providing means for, scalability and fault tolerance. Also, optimizing the communication load is critical for efficient MapReduce-based algorithms.

The discussion presented here is a condensed and near-verbatim form of information in Wikipedia [27]. Hadoop framework [26] is an open-source implementation of MapReduce echo-system and it has now evolved into a software framework for distributed storage and distributed processing of very large data sets on common computer clusters (i.e., built using commodity and cheap hardware). Main strength of this framework is that all component in this are designed with the assumption of frequent hardware failures. The storage part of this framework is called the Hadoop Distributed File System (HDFS) [27], which splits files into large blocks and distributes them across nodes in a cluster. The main concept is to minimize the data transfer and send the processing to the data by distributing executable code to the data nodes (i.e., sending the programs to the locations of the data).

The Hadoop framework is mainly written in the Java programming language with minor native code in C++ and some utilities written using shell scripts. To orchestrate and coordinate this framework, a resource manager is an essential component to this framework. Hadoop YARN (i.e., Yet Another Resource Negotiator) is accountable for managing computing resources in hardware clusters and scheduling of computing tasks. This framework is now evolved to a rich collection of additional packages, which are installed alongside Hadoop Framework (such as Apache Pig, Apache Hive, Apache HBase, Apache Phoenix, Apache Spark, Apache ZooKeeper, Apache Flume, Apache Sqoop, Apache Oozie, Apache Storm), which yield into a complete ecosystem to handle challenges associated with executing big data applications. The TruSStReMark framework proposed by this dissertation uses HDFS to store its data and uses Hadoop ecosystem to execute the parallelized versions of its trust-based selection and recommendation algorithms (i.e., presented in, CHAPTER 5, and CHAPTER 6).

(3) Data Streaming Processing and Spark Ecosystem [29] [79]**.** The discussion presented here is a condensed and near-verbatim form of information in Wikipedia [79]. Stream processing [79] is a computing paradigm, which allows applications to use limited form of parallel processing and take advantage of them by streaming the data in to them. Usually these applications tend to use multiple computational units (e.g., Graphics Processing Units (GPU) or Fields Programmable Gate Arrays (FPGA)) without explicitly managing allocation, synchronization, or communication between them. The main goal of the stream processing unit is to apply a set of transformation to each item in the input.

Apache Spark [29] is an open-source framework for stream processing using large clusters. It was originally proposed as a set of interfaces for processing large volumes of data streams, with the capability for data parallelism and fault-tolerance. Main components of Apache Spark is its Application Programming Interfaces (APIs) and its special data structure names as the Resilient Distributed Dataset (RDD). The RDD is a read-only collection of data items which are distributed over a cluster of machines which also provides fault tolerance. This type of data-model reduce drawbacks in the MapReduce paradigm by eliminating batch processing and sequential dataflow and also lets iterative algorithms to operate over the dataset many times inside a loop to perform its analysis. Apache Spark and Spark Streaming concepts [30] use fast scheduling capabilities of its core project to perform advanced streaming analytics. The idea is to input datasets in small sets such RDD transformations on those mini-batches of data is fast, however the latency is equal to the mini-batch processing duration. The TruSStReMark framework proposed by this dissertation uses further improve the performances of its both online and offline phases of its trust-based selection and recommendation algorithms (i.e., presented in CHAPTER 5, and CHAPTER 6).

CHAPTER 4. TRUSSTREMARK APPROACH

The work presented here is based on our previous publications [88] [89]. The framework proposed by this dissertation is named TruSStReMark (i.e., Trust-based Service Selection and Recommendation for Online Software Marketplaces). This chapter introduces the main ideas and parts of this framework in the context of software apps and online marketplaces.

To help in identifying the challenges associated with trustworthy service representation, selection and recommendation, the operation of an ideal system is considered in the following paragraphs. In social sciences, trust is defined as the relationship between entities such as person, people, communities, organizations and nations. It considers the dynamics of inter- and intra-entity interactions. The definition of trust is based on one entity (i.e., truster) willing to rely on the actions of another entity (i.e., trustee) performing functions and this situation is considered from the past to the future. From the truster's view, collection of evidence about completed functions as expected can contribute to the trustworthiness of the trustee. Evidence can both increase and degrade trust, additionally, more evidence will increase the confidence in the judgment of the trustworthiness of trustee. When this scenario is mapped to assembling a software system, entities are thought of as software apps. In an ideal system, all evidence and related opinions about the trustee should be visible to all. However, the term 'all evidence' is weak (i.e., not practical), as there could be infinitely large number of parameters related to an entity.

Therefore, additional contextual information is needed to help prune the number of related parameters. For example, the comparison between the overall trustworthiness of a person against the trustworthiness of a persons' driving capability is based on their history of accidents and/or friends' opinions about driving. If all evidence related to a particular context is visible for all entities, then the trustworthiness could be inferred in an ideal system. However, in practice all evidence related to the context is not shared due to the public/private nature of attributes and relationship status between entities.

In contrast to the other domains such as social sciences, trust related concepts are not well-defined in software apps and systems (which are composed out of apps). Also, if the truster has control over the action performed by the trustee, then, the truster has no uncertainty about the outcome of that action. In reality, this situation rarely exists and the truster is always uncertain about the outcomes of the other's actions. It can only assume and evaluate expectations or receive recommendations from others hence, this uncertainty involves the risk of damage (or even failure) to the truster, if the trustee does not behave as desired. On the other hand, any online service operation is highly dependent on the environment. If static views of service parameters are matched against service requirements, then important details are ignored such as whether the service delivers what it promises. Therefore, it is worth investigating modeling, specifying, selecting and recommending trust related aspects in integrating complex software systems.

4.1 Motivation for TruSStReMark framework

Due to of inherent features (such as, reuse, frequent updates and dynamic nature of the hosted environments) associated with the software domain, it is challenging to apply existing product based approaches (such as ranking, selecting and recommending entities) to software marketplaces as they are. Compared to products which are described using a list of physical features, software is described using a list of virtual attributes. For example, most products have strict physical features such as weight and dimensions. However, software is usually described using an interface with a set of quality of apps (QoS features). These QoSs values may at times have associated ambiguity because of changes in the execution environment and presence of bugs. Therefore, in an online marketplace the comparison of consumer products to online software apps yields a key difference: that products have stable features while software evolve continuously over time. Appendix B of this dissertation presents a more detailed comparison physical products vs software.

Moreover, while the selection of a product usually happen in isolation, selection of software more often happens due to system requirement (i.e., need to use inside a more complex and distributed software system). Unlike products, software evolves from its initial release with continuous updates. Hence, information available about software needs to be evaluated in a time sensitive manner by keeping the uncertainty aspects of its behaviour in mind. For example, compare a physical product (e.g., a data backup storage device) and a software service (e.g., a cloud-based backup storage service). Physical product is self-contained, however, software service operated in an environment which is shared by other apps. Therefore, software apps has its associated uncertainty and time sensitiveness about its behaviour to work inside a system and the recommender system also need to consider these aspects while recommending any software. The selection and recommendation of software apps becomes easy when the environment of a product is fairly static, but becomes more complex when the software service has to work in a system where the environment is dynamically changing. Figure 4.1 shows the interactions of stakeholders who are involved in service creation, publishing, indexing and consuming in marketplaces. Currently, the role of the service broker is optional and is replaced by large service marketplaces.

The concept of trust is subjective in nature and is influenced by many related factors. The trustworthiness of an entity not only depends on its behavior, but also on anything that has a direct or indirect relationship with it. Hence, the trust of a service collaboration between a truster and a trustee depends on various factors such as previous direct experiences, indirect experiences, and common beliefs.

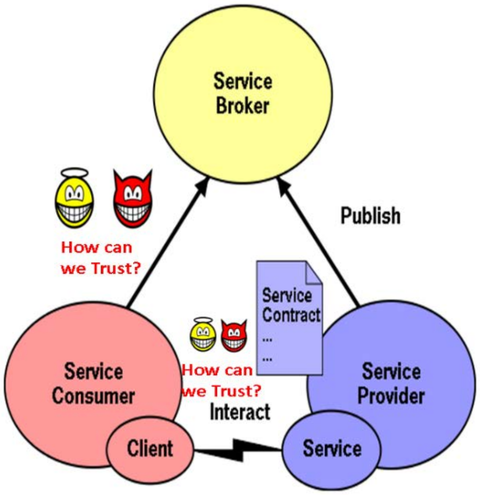


Figure 4.1 Stakeholders’ interactions in online marketplaces [87]

For example, the trustworthiness of a software service (which is a part of a larger software system) depends on both the trustworthiness of its features (e.g., evidence about its functional and QoS attributes) and the trustworthiness of its other interacting apps (e.g., evidences about successful collaborations). Calculating the trustworthiness of a software service is normally considered as an afterthought, rather than considering it as an essential step during the selection of a service. These facts provided the motivation to investigate how to identify and quantify the trust of a software service.

Uncertainty is associated with software systems due to the following reasons. It is hard to verify that a software system is free of bugs. Moreover, the contexts and associated environments are changing and possibility of failures of components of the system. Since the trust and uncertainty are inter-related, the trustworthiness of a software system can be expressed as a measure of a lack of uncertainty. Therefore, trustworthiness is a measure of confidence and also a measure of certainty. It is natural for developers and consumers to express conflicting opinions about their trust of certain attributes of a software. Moreover, different consumers have subjective opinions about the software systems based on their experience. Such situations are becoming more common in online marketplace environments. These reasons provid the motivation to investigate about a suitable trust model and to quantify trustworthiness of a service from software marketplaces.

As a step towards creating trustworthy software systems involving heterogeneous apps, it is important to quantify the trust of individual apps and represent this by aggregating associated evidence over time. A software service may also provide different levels of service level agreements. For example, a cloud-based storage service could provide a basic efficient storage service and additional negotiable levels of the service based on other features, such as improved capacity and speed. The trust of these levels of varying QoS features need to be represented in a service specification which is accessible to consumers. A proper representation of trust aspects of apps (i.e., as a trust contract) can address the mismatch between self-description by service providers and expectations of the service consumers. These facts provided us with the motivation to propose a trust contract that extends the concepts of multi-level specifications and represent the quantified trust values of service attributes inside it.

At the end of the service selection, or completion of service recommendations, the consumer needs to have confidence about the apps. Although it is not available at this time, in the future, appropriate automatic negotiations will be initiated based on both service and systems requirements between consumers and marketplace. The main challenge in these service negotiations between service providers and consumers will be the lack of trust, which may lead to incorrect decision and designs. Hence, the service selection and recommendation algorithms provided by the marketplace need to consider trust aspects of apps, in addition to the functional and QoS features. Such a trustworthy service selection and recommendation processes will act as a precursor to service interactions. These facts motivated us to augment the marketplace-based prevalent selection and recommendation algorithms (which are based on techniques related to content-based and collaborative filtering techniques) to consider trust-based attributes of software apps.

Based on the above motivations, let us consider the real-world scenario of searching for suitable public camera/sensor apps to be used in a distributed object tracking system. At the service selection (and also recommendation) time, investigating independent camera/sensor apps based on their specifications does not provide certainty of their behavior over a period of time. Instead, if the trust associated with the camera service is properly quantified and represented, it will provide a better foundation during the selection process and will also aid during the recommendations. However, this process is not trivial due to the heterogeneous nature of evidence, the time sensitive and dynamic nature of trust, and the presence of uncertainty. The following subsection provides a list of associated challenges and assumptions made by prevalent works. The work performed by this dissertation aims to provide the necessary foundations for overcoming these challenges.

4.2 Shortcomings of Prevalent Approaches and Associated Challenges

1) Compared to certain other domains, the trust of a software service is not well defined in the context of online software marketplaces. This is because understanding, evaluating, and resolving trust related issues are complex, costly, and time-consuming (especially during late phases of the software lifecycle).

2) It is assumed that a static and complete view of information about software apps is available from the providers (i.e., from the marketplaces). However, in reality the information available about software apps and the software itself gets dynamically updated.

3) It is also assumed that there is no uncertainty or ambiguity about the representation or behavior of the software and the environment. However, in reality there is uncertainty and distrust about the varying levels of QoS features.

4) There are not many studies on evaluating trust in software systems, and these approaches do not provide a generalized trust model that is parameterizable with different application contexts. Moreover, the prevalent models are not evolvable across different phases of the software apps.

5) The representations of service parameters based on temporal aspects are ignored. Given the dynamic nature of apps in an online marketplace, it is important to aggregate service behavior over time.

6) Not many approaches have attempted to quantify and represent trust related service attributes in such a way that this information can be updated frequently and the selection and recommendation algorithms to access them efficiently.

7) The apps may not always behave as indicated by their specifications and contracts. However, evidence about service behavior violations and user feedback (describing these violations) is usually ignored.

8) Consumers are responsible for inferring the trustworthiness of a service’s ability to meet requirements by analyzing available evidence (e.g., reviews). Hence, the confidence about the selected apps ability to provide required QoS values is not addressed in existing techniques of selection and recommendation approaches (such as Content-based Filtering (CBF) and Collaborative Filtering (CLF)).

9) Analyzing external evidence presents a challenge when the number of apps are increasing in marketplaces each day. However, the performance of service selection and recommendation should be near real-time and should be transparent to the consumers of software marketplaces.

10) Waiting too late in the service lifecycle to understand, evaluate, and resolve trust related issues can be a costly and time-consuming process. Therefore, trust-related concerns should be identified, aggregated and presented to the consumers by the marketplace in an appropriate manner.

Because the concept of trust crosscuts all functional and QoS aspects of the apps, it presents a challenge. The above mentioned challenges and limitations are evaluated as the starting points of this proposal of TruSStReMark framework. The trust aspects related to the challenges and the ambiguous nature of the limitations are addressed in this dissertation.

Hence, the TruSStReMark framework incorporates trust-related evidence and opinions during modelling, representation, selection and recommendation of confidence related to software apps.

4.3 Overview of TruSStReMark Framework

Before trust-based enhancements are presented by the TruSStReMark approach, the abstract problems of service view and model, representation, selection and recommendation are formally defined in the following paragraphs.

(1) Abstract view of a service and model in a marketplace are defined as follows:

- Service Marketplace (M) consists of Apps (i.e., Service Universe (U)) which are uploaded by a set of individual Service Providers. The difference between M and U is that M is always a subset of U, because apps get moved from one marketplace to another.

- It also has a set of Service Consumers (C), each member of which has different Service Requirements (R). The problem of Service View is how the Marketplace defines a particular characteristic of a software service (e.g., the trust of a software service)

The problem of Service Model is how to govern of the lifecycle the Service Universe which can facilitate efficient service representation selection and recommendation.

(2) Service Representation is defined as follows:

- Service Universe (U) consists of the set of all publicly available service specifications as a list which is denoted as (S).

- Any Service Specification (S_i) is $S_i \in U$ such that

 - S_i contains ‘n’ service parameters $S_i = (P_0, P_1 \ldots. P_n)$

Service Representation is the problem of identifying suitable representation mechanisms to organize, categorize and publish different service parameters (P) such that it facilitates efficient service selection and recommendation.

(3) Service Selection Problem is defined as follows:

- Systems Blueprint (B) contains individual Service Requirements. e.g., a Service Requirement (R_i) is $R_i \in B$

- R_j contains m service parameters $R_j = (P_0, P_1 \ldots. P_n)$

- Marketplaces contain public service specifications (a service is named as S_i).

- There can be multiple apps which match to a particular set of requirements such that $R_j \subseteq S_i$ or $R_j \supseteq S_i$ or $R_j = S_i$.

The Service Selection problem identifies a set of apps by matching against a given set of requirements (ranked in the order of relevance).

(4) Service Recommendation Problem is defined as follows;

- There exist many apps which could suggest or provide additional apps which can support or replace entities in the system according to a given $R_{i.}$

The Service Recommendation Problem suggests service recommendations based on service parameters (ranked in the order of relevance).

The following subsections present how the TruSStReMark framework enhances the abstract problems to include trust-based aspects by incorporating an evidence-based approach. The first step in the TruSStReMark framework is defining a concrete trust model for software apps.

4.4 Trust Views and Trust Model [13]

This subsection contains a summary of our server paper [13] on trust views and models. As there are many definitions and views of trust, it is necessary to identify if there exists a commonly agreed upon view and an associated method of quantification that is suitable for software-intensive systems. In [13] we have performed a comprehensive survey of different views of trust of software apps spanning over 25 years. This survey revealed the following findings: (1) trust of a software system is subjective in nature; (2) uncertainty is an inherent part of trust evaluation; (3) it is important that the trust attributes be aggregated during all phases of software lifecycle; (4) trust evaluations of prevalent approaches do not consider both internal view (i.e., the developer's point-of-view) and external view (i.e., the user's point-of-view) of software apps; and (5) it is important to capture the temporal and dynamic nature of trust, especially when considering online software services within a changing environment. We started to investigate both of the individual service level (bottom up) and the system level (top down) of how to integrate trust. In this dissertation, we have taken the bottom up approach. Also, the survey has aggregated and analyzed the trust definitions and models of software to understand how to define the trustworthiness of a software service.

4.4.1 Trust Views of a Software Service

In our previous survey [13], the differences and commonalities between various trust definitions from 80 peer reviewed publications have been identified and categorized. It also presents both automatic and manual categorization of existing trust definitions and these categories show that the research community has accepted the inherent attributes of trust of software such as subjectivity and uncertainty. In that survey, the trust definitions listed are categorized based upon different factors. Most definitions follow the view that trust is a relationship between two entities: the truster (i.e., the entity that initiates the relationship) and the trustee (i.e., the entity whose trust is being evaluated). In this relationship, third-party entities can also be involved in the form of past users, recommenders, and certificate authorities. With this basic understanding of trust, an automatic classification using Carrot2 tool [79] on the collected set of trust definitions was carried out. The main goal of this classification effort was to determine if a prominent definition or a set of related views about trust are accepted as the main choice by researchers.

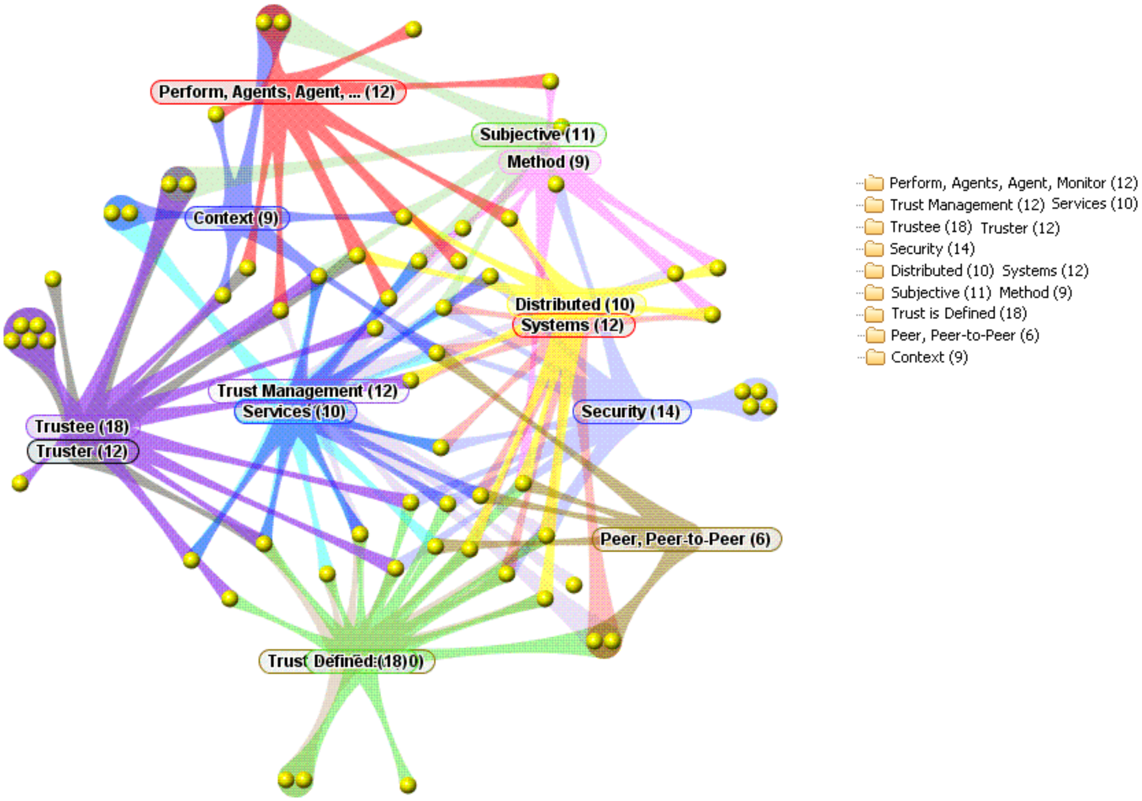


Figure 4.2 Clustering of the trust views of different publications [13]

The open source Carrot2 classification tool [79] was used to classify these aforementioned research efforts because it supports many clustering algorithms (e.g., K-means clustering algorithm, Lingo clustering algorithm, and Suffix Tree clustering (STC) algorithm). All the identified trust definitions were transformed into a suitable input format for Carrot2, then K-means and STC algorithms were performed on them. These two algorithms were used because Carrot2 selected them as the best performing clustering algorithms against the given inputs. From the analysis presented in [13], it is possible to acquire some insight about which trust definitions (and the associated publications) share similar views on trust. However, it is difficult to obtain in-depth information about the similarities and differences between different trust definitions from the classification alone. For example, it is difficult to identify which trust definition, or set of trust definitions, is more prominent than others. As there are many similar sized clusters, and many publications that share different clusters, clustering algorithms fail to find trust model similarities using the different phrases of software life-cycle, which are explained in natural language. In the case of the K-means clustering, the result illustrates which trust-related publications are within similar domains.

For example, there is one cluster for peers' reputation and one cluster for service monitoring aspects of trust. However, each cluster does not clearly indicate any further details, such as the perception of trust. When compared with the K-means results, the STC results provided additional levels of details such as which trust views are shared among clusters. More specifically, the STC results showed which trust definitions are related (i.e., in dictionary meaning) to each other and which clusters are not. This relation is determined by the distance between the clusters. Based on these results, only high level information about the trust models can be discovered, and there is not enough information to identify a dominant trust definition or a trust model. Therefore, further analysis was performed manually (i.e., by reading and analyzing these publications) to further evaluate the existing trust definitions.

Table 4-1 Trust-related factor with respective to owner of the view [13]

Association	List of Factors
With Truster	1. Decision (subjectivity) 2. Ability to monitor trustee 3. Functional requirements 4. QoS requirements 5. Willing to be vulnerable 6. Willingness for benefits 7. Context 8. Knowledge 9. Experience 10. Policy
With Trustee	1. Reputation 2. Functional capability 3. Non-functional capability 4. Security and safety 5. Cost or competence 6. Development cycle 7. Owner's reputation
With Neither	1. Compatibility with others 2. Third-party experience 3. Dynamic nature of trust

The manual analysis carried out in [13] considered various trust definitions and models based on factors such as different entities involved in the trust relationship. The list of selected publications were manually categorized based on the trust factors and the publications that associates with these factors. The critical question to be answered in the context of complex software system is as follows: at which points in the software life-cycle of these services should trust be taken into account? This is crucial because the integration of trust into the software life-cycle of such service enables evaluating and reasoning about their trust-related properties. As indicated in Table 4-1, there are three main ways of categorizing trust associations: associations with truster, associations with trustee, and associations with others (i.e., the views that are not related to either the truster or trustee). The analysis provides a detailed categorization of each of these factors. The factors associated with the truster are various concerns such as benefits, experiences, and the subjective choices. The factors associated with the trustee are concerns such as owner's reputation, the software life-cycle, and (non-)functional capabilities. The third category includes the factors not belonging to the two previous categories, such as the compatibility between truster and trustee, and experience about other agents. The analysis section of this survey paper then summarizes each of these factors that resulted from the manual categorization.

This survey identified factors that reinforce the key idea that trust is a subjective concept. This paragraph briefly presents the definition of each factor in the context of software services (in depth details are presented in the journal publication [13]). The following are the list of related factors involved with truster: **(1) Truster's decision:** A truster's decision is a predicate that the truster perceives of the trustee. **(2) Truster's (in)ability to monitor trustee:** A truster's (in)ability to monitor the trustee is whether the truster is able to monitor the trustee (i.e., collect indirect evidences) before directly interacting with the trustee. **(3) Truster's functional requirements:** A truster's functional requirements are the set of functionalities that the truster expects to consume to fulfill its computational requirements. **(4) Truster's non-functional (QoS) requirements:** A truster's non-functional requirements are the set of QoS constraints that the truster requires to be fulfilled when consuming the trustee's functionality. **(5) Truster's willingness to be vulnerable:** A truster's willingness to be vulnerable is defined as it being aware that its interactions with a trustee may fail and cause itself a loss. **(6) Truster's benefits:** A truster's benefit is the profit or advantage that the truster expects to gain by interacting with the trustee. (7) **Truster's context:** A truster's context is the environment of the truster when interacting with the trustee. The context includes the truster's execution platform, location, and mobility (i.e., the truster's ability to move freely). **(8) Truster's knowledge:**

A truster's knowledge is the evidence accessible to the truster about a trustee that can be used in evaluating its trust about the trustee. **(9) Truster's experience:** A truster's experience is the knowledge that the truster has gained from all direct evidence about a trustee (i.e., the evidence from past interactions with the trustee and observations of the trustee about interactions with others). **(10) Truster's policy.** A truster's policy is the set of principles or rules that guide the truster in evaluating trust about a trustee.).

Following are the list of related factors involved with trustee: **(1) Trustee's reputation:** A trustee's reputation is a measure of positive feedback or negative feedback (e.g., ratings and comments) that the trustee has received from its past interactions with different trusters. **(2) Trustee's functional capability:** A trustee's functional capability is its ability to perform a desired action or behavior. **(3) Trustee's non-functional attributes:** Trustee's non-functional attributes are the set of QoS guarantees provided by the trustee. **(4) Trustee's security and safety:** A trustee's security is the set of security properties provided by the trustee. **(5) Trustee's cost or competence.** A trustee's cost is the amount that a truster should pay to obtain the service of a trustee. **(6) Trustee's development cycle:** A trustee's life-cycle is a trust factor that shows how the trustee is constructed throughout its life-cycle. **(7) Trustee's owner's reputation:** A trustee's owner's reputation is a measure of the positive feedback received by the owner of the trustee.

Finally, the following are the list of related factors involved with neither truster nor trustee: **(1) Compatibility between truster and trustee:** Compatibility between the truster and the trustee is the ability to work together towards achieving a common goal. **(2) Third-party experience:** Third-party experience is the available evidence of interactions between the trustees with the third-party entities. **(3) Dynamic and fuzzy nature of trust:** Software becomes updated frequently; hence, their features and qualities change over time.

Also, the categorization shows that the following important factors are related with trust definitions: experience, willingness to be vulnerable, willingness to acquire benefits from truster’s side, plus reputation, QoS, and functional capabilities from trustee’s side. More details about each of these factors, with examples, can be obtained by referring to the sections the Survey paper [13].

4.4.2 Trust Model of a Software Service

The trustworthiness of a service not only depends on its behavior, but also on anything that has a direct or indirect relationship with it. For example, as part of a composed software system, the trustworthiness of a service depends on: 1) the trustworthiness of its features, such as all reported evidence about its service guarantees, and 2) trustworthiness of all other related entities, such as evidence about cloud services with which the service often communicates. Considering these aspects, and based on the views from the Trusted Computing Group [15], (i.e., “An entity can be trusted if it always behaves in the expected manner for the intended purpose”), we have defined the trust associated with a software service as the *“degree of conformance of its behavior (both functional and non-functional) to its published specification (all the levels of contracts)”*.

Also, our previous work has presented a trust model focusing on different artifacts produced at each phase of the software lifecycle [80]. Artifacts are conceptual or physical outcomes of a particular phase of the software lifecycle. An example of an artifact is a test-case created during testing phase. Each artifact has a set of properties that can be evaluated by the developers, by the users or by both. These represent internal and external evaluations. The evaluation of a property of an artifact could be different for each artifact and involves different stakeholders. This evaluation creates a set of evidence which needs to be aggregated using a suitable quantification model.

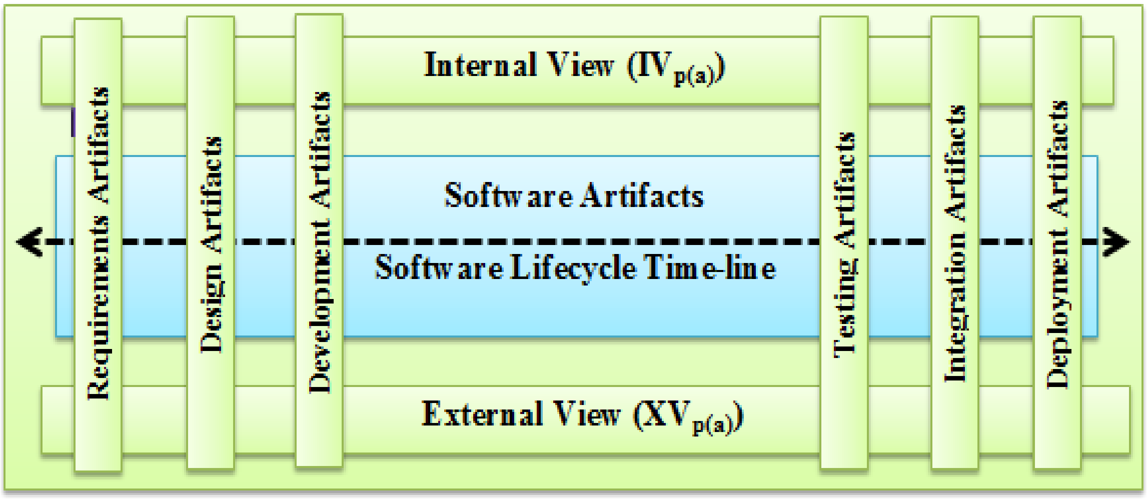


Figure 4.3 Trust model proposed for evaluating trust along the service lifecycle [80]

Figure 4.2 presents the outline of the model used in the TruSStReMark framework [80]. The internal view of the properties is related to the developer’s interactions with the artifacts throughout the lifecycle, such as coding conventions and unit testing. In contrast, the external view of the properties represents end-user's measures of the artifact based on all evidence such as user comments and ratings about the artifact.

Let us consider applying the above model to a complex software system with strict QoS constraints on timing and resources, such as processing power, memory, and battery power. The model evaluates trust of artifacts in each phase of the software cycle—without limiting to the artifacts in deployment phase. In this evaluation process, for each phase, the artifacts and their relative importance have to be identified. For example, in the development phase, the classes and components that affect real-time and resource constraints have to be given higher priorities. After the artifacts are identified, their trust values have to be evaluated from both internal and external perspectives. These evaluations can use a quantification mechanism, such as subjective logic [23] base tuples of <Believe;Disbelief;Uncertainty> (for more details on Subjective Logic refer to Subsection 3.2.2). After quantification both internal and external views are represented in two (B, D, U) tuples. Defining policies to evaluate the trust of each of these views is a complicated task as it should consider many factors, such as the identifying properties and their importance, quantifying trust values of the properties and defining operators on composing trust values.

For example, when evaluating components in the development phase, we can assign weights to each class and function based on their effect in realizing the QoS constraints. We can then use weighted averages to compose trust values. These weightages can be adjusted over time from the feedback of the users. That is, if the trust value evaluated from the internal view disagrees considerably with the trust value evaluated from the external view, then the internal trust policy should be adjusted by updating the relative weights of the properties. Similarly, each of the properties can be incrementally quantified as (B, D, U). For example, when evaluating the internal trust of functions in development artifacts, the properties such as proofs of time and space complexities of the algorithms, which directly affect the real time and resource constraints, can be considered. As the amount of all evidence are less in number at initial stages, the uncertainty (U) associated with the artifact will have a higher value. When more evidence get collected, the belief (if the evidence agrees with QoS constraints) or disbelief (if the evidence disagrees with QoS constraints) associated with the artifact will improve over time. Finally, to evaluate the overall trust, the (B, D, U) values derived from both internal and external policies have to be aggregated. It is possible to assume when an artifact is released to its user base (before users have experienced the artifact), the uncertainty of the external trust is very high. At early stages of the artifact, the consideration of internal trust should be high when evaluating the overall trust. As user experience grows, the external trust should be given higher consideration.

In order to evaluate trust at each phase of the software lifecycle, we have to evaluate trust at each intermediate phase. Moreover, the results (or output) of one phase must serve as the properties (or input) of another phase. With this in mind, we denote the intermediate result of a phase as an artifact (a). Likewise, there could be multiple artifacts as the result of a phase. Artifact trust is a measure of how much the artifact meets its expected outcomes. Also, artifact trust can impact the overall trust of the system. At each phase of the software lifecycle, different artifacts interact with different users, who have certain views about state and behavior of artifacts. It is important to identify what is essential for evaluating trust from these complex interactions and views. We call these properties of the artifact, which are denoted as p_a.

Each artifact contains a set of properties that can be viewed either internally or externally. The evaluation of a property of an artifact could be different for each artifact and involves different stakeholders. This evaluation creates lot of evidence that allow the developer and users to provide their views of a service’s trust, and accounts for the subjective nature associated with a service. The internal view of properties (IVp_a) represents processes associated with realizing the artifact throughout the lifecycle, such as coding conventions and unit testing. In contrast, the external view of properties (XVp_a)

represent user experiences about the artifact. Using the above definitions, our trust model will build upon the subjective logic [23] defined by Jøsang (for more details on Subjective Logic refer to Subsection 3.2.2, where trust is defined as the tuple of (B;D;U). We selected this trust model as the foundation for our approach, because it addresses uncertainty associated with trust, while providing essential operators for trust aggregation and comparisons.

Table 4-2 Internal and External Views [12]

Artifact	Phase	Internal View Properties	External View Properties	Trust
(1)Requirement specification	Requirements	Use cases, State charts, Data flow diagrams	Infeasible or Contradicting/-Supporting requirements	IVp_a (B;D;U); XVp_a (B;D;U)
(2)Design specification	Design	Class Diagrams, Component Diagrams, Requirements	Anti-patterns/code-smells	IVp_a (B;D;U); XVp_a (B;D;U
(3)Classes/ Components	Development	Functions, Source code, Code Coverage, Design	Features, profiling results	IVp_a (B;D;U); XVp_a (B;D;U
(4)Test Oracle, Test Suites	Testing	Test scripts, Test cases	Test coverage, Test Results and Remarks	IVp_a (B;D;U); XVp_a (B;D;U
(5)Integrated system	Integration	Classes/ Components, Integration Framework	The integration test results	IVp_a (B;D;U); XVp_a (B;D;U
(6)Deployed service	Deployment	Integrated system, Deployment Framework	Features, Reputation (Rating/Feedback), Behavioral knowledge	IVp_a (B;D;U); XVp_a (B;D;U)

For a complex software system, at a particular software lifecycle phase, and for a particular artifact a, there exists an internal view of properties (IVp_a) that could be defined as an aggregation of set of internal properties PI where $p_a(PI_i)$. Each property can be evaluated with respect to a tuple of (B)elief, (D)isbelief, and (U)ncertainty as $T_i = (B_i;D_i;U_i)$. Trust of the internal view of an artifact $T(IVp_a)$ is therefore defined as the collection of (B;D;U) tuples of internal properties. The internal view's evaluation is based on the inherent properties of the artifact, such that it is made of (i.e., factors). This evaluation should give prominence to the properties related with time and resource constraints. The partial list of such inherent properties to be considered during each phase of the software lifecycle for different artifacts is provided in Table 4-2. According to this table, artifacts in early phases are part of the inherent properties of artifacts in later phases. For example, a component's design in the design phase is an inherent property of the same component's implementation in the development phase.

In contrast, the external view's evaluation is based on external user experience of the artifact, e.g., set of evidence that the user collected when interacting with the artifact. The external users of an artifact during a particular phase of a software lifecycle include internal and external users of artifacts in following phases. For example, external users of the requirement specification can be identified as design engineers who try to achieve the requirements from their designs, the software service developers who implement the requirements, and the service consumers who use the features mentioned in the requirements. Each user may have a differing opinion about the requirements based on their interests and context. The user can collect the evidence from direct experience, e.g., first hand usage, or the indirect experience, e.g., user ratings/recommendations. The external view of different artifacts is provided in Table 4-2.

Finally, for a software service, at a particular software lifecycle phase, and for a particular artifact (a), overall trust of the artifact phase T(a) is as an aggregation of both $T(IVp_a)$ and $T(XVp_a)$. When evaluating the trust of a software service, which is also an artifact, using this model, the trust of artifacts in preceding phases of the software lifecycle are implicitly considered in its internal trust value. The trust of a software service is therefore an aggregation of trust of artifacts in all the phases. We have used this model in our TruSStReMark framework, however during the experimentation and analysis external views were the main focus.

4.4.3 Trust-based principles for reusing services

There are two kinds of important trust-related factors related to the software lifecycle of service: direct and indirect. Direct factors are related to a particular phase of the software life-cycle. For example, if a trustee is a reusable service, then its functional and non-functional capabilities are directly related to the selection of that service to be integrated to the system. In contrast, indirect factors, in addition to affecting a specific phase, also impact other phases of the software lifecycle. For example, if a selected service has trust issues, then there will be trust issues during the phase of composition of that service with other services. Both direct and indirect trust factors may increase the uncertainty associated with the outcome of the software system. Therefore, adhering to a set of principles for identifying, describing, quantifying, and evaluating trust during all associated phases of the software life-cycle is critical in the design or reuse of services recursively for each sub-service which they will depend upon.

Hence, we have identified the following eight principles for identifying, describing, quantifying, and evaluating trust for the software lifecycle of service [13]. These principles are as follows: (1) trust requirements, (2) trust representation, (3) trusted design and associated knowledge base, (4) trust specification, (5) comparison and selection based on trust, (6) countermeasures for trust, (7) composition of trust attributes, and (8) evaluation and maintenance of trust. Each trust-based principle is evaluated using a set of set of evidence that is obtained either internally or externally. Figure 4.4 illustrates how these trust related factors can be applied to identified related a principles, within the lifecycle of individual software services and to a complex software system.

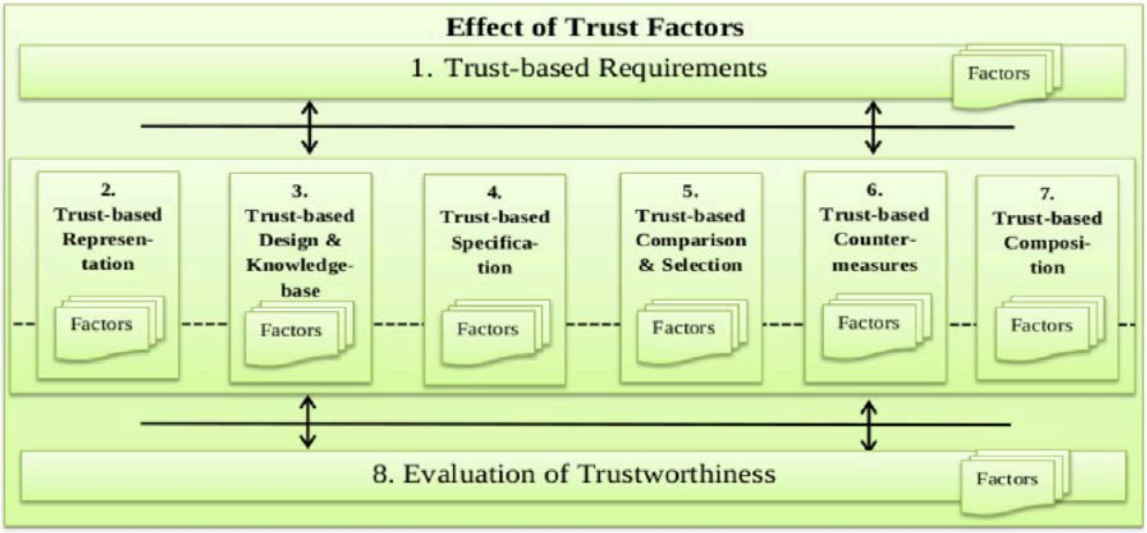

Effect of Trust Factors
1. Trust-based Requirements
Factors
2. Trust-based Represen-tation
Factors
3. Trust-based Design & Knowledge-base
Factors
4. Trust-based Specifica-tion
Factors
5. Trust-based Comparison & Selection
Factors
6. Trust-based Counter-measures
Factors
7. Trust-based Composi-tion
Factors
8. Evaluation of Trustworthiness
Factors

Figure 4.4 Application of trust factors to the principles while developing a complex software system [13]

All the above mentioned principles together present a structured method of realizing a software-intensive, trusted, and complex systems created from individual software services. An adherence to these principles allows reasoning about the trust value of such systems – reinforcing the need for the consideration of trust as a continuous objective instead of an afterthought while realizing such complex software systems such as a Distributed Tracking System presented in Figure 4.5.

The next set of paragraphs briefly describes the incorporation of these principle to a complex system as a case study to assemble a Distributed Tracking System (DTS) (i.e., a system which can track moving targets using set of distributed sensors). The idea is to reuse available services using marketplaces and assemble this complex system. To assess the overall trust (T_{DTS}) of the DTS, trust of each reusable service and their trust values need to be investigated. These services and their corresponding trust quantifications (as seen in Figure 4.4) are named as: object markers (T_{OM}), sensor services (T_{SS}), discovery service (T_{DS}), control service (T_{CS}), and fusion service (T_{FS}).

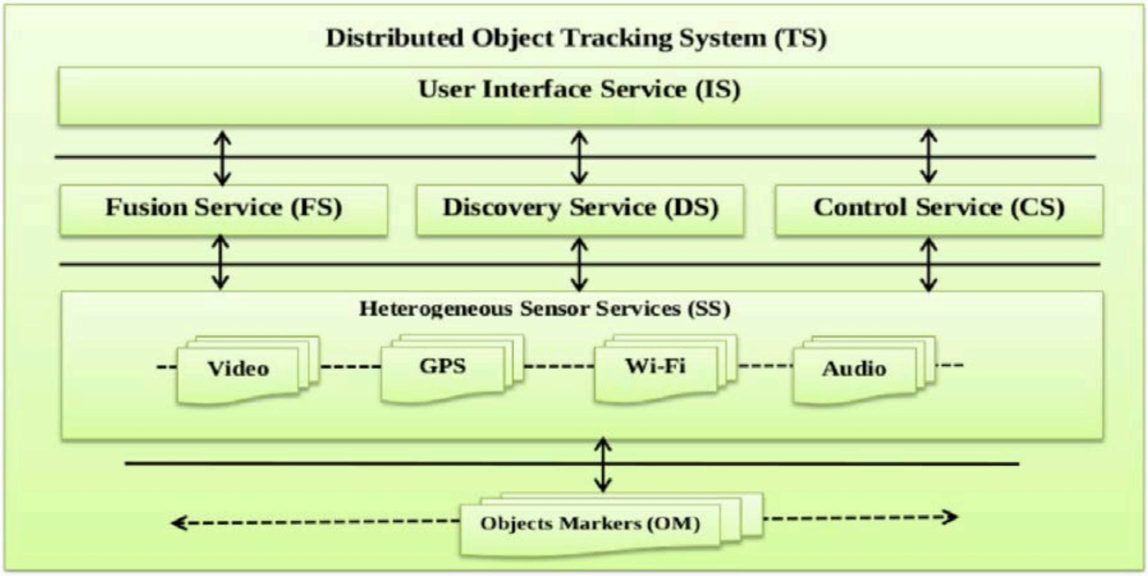

Distributed Object Tracking System (TS)
User Interface Service (IS)
Fusion Service (FS)
Discovery Service (DS)
Control Service (CS)
Heterogeneous Sensor Services (SS)
Video
GPS
Wi-Fi
Audio
Objects Markers (OM)

Figure 4.5 Illustration of required services of Distributed Object Tracking System and their interaction and communication patterns [13]

The following discussion only focuses on the sensor service, specifically, the reuse of a Webcam service, and how the aforementioned trust-related principles are used in the context of a Webcam service. Hence, a similar approach can be followed to evaluate other services (referring to Figure 4.4 and Figure 4.5).

Application of Principle 1 (trust-based requirements): As indoor tracking is a quality-aware application, it has strong non-functional requirements such as an overall response time of around 30 milliseconds (ms) and an accuracy of 5 meters (m). These non-functional requirements need to be decomposed into appropriate requirements for each services such as the Webcam service. The truster is the DTS integrator and the trustee is the specific Webcam service. The truster needs to define its trust-related requirements (e.g., a specific opinion about a Webcam service, an ability to monitor status of a Webcam service, and an adherence to requirements such as 30 ms of response time). Similarly, the trustee needs to specify its trust-related factors such as maintain its reputation, meeting its functional and non-functional requirements, and compatibility with other sensor services. In this situation, the trust-based requirements of the truster and the trustee need to be compatible with each other. **Application of Principle 2** (trust-based representation): The Webcam service developer may decide to select the subjective logic-based trust representation suggested in the previous subsection, because of its advantages that are listed in Subsection 4.5.2. **Application of Principle 3** (trust-based design artifacts and knowledge base): An example of the design artifact for the Webcam service is its loose coupling nature with the other services (i.e., ability to act independently), while cooperating with other sensor services or semantics associated with its replacement by another Webcam service. The corresponding knowledge base would aggregate related information such as a list of which camera services have better trust attributes and reputations. **Application of Principle 4** (trust-based specification): A specification of a Webcam service will indicate, in a formal manner such as a trust-contract described below [81], the functional and non-functional attributes of that service along with, for example, the [B, D, U]-based quantification of these non-functional attributes. The [B, D, U] tuple associated with the ability of the Webcam service to meet the response time deadlines may be quantified as (0.8, 0.1, 0.1) based on all evidence available from the empirical testing of that Webcam service by the developer (*i.e.*, the internal view) and the reviews provided by the users of this Webcam service (*i.e.*, the external view). Similarly, other trust attributes of the Webcam service will be quantified and specified. **Application of Principle 5** (trust-based comparison and selection): When comparing two Webcam services, the service with the higher B, lower D, and lower U is preferred when compared with others – and if the selection and matching criteria (such as described in [80]) also include trust-based matching operators, then the application of

Principle 5 is complete. **Application of Principle 6** (trust-based countermeasures): After selection, an investigation of selected services for their future threats and listing countermeasures against possible risks of the selected services can be viewed as adhering to Principle 6. **Application of Principle 7** (trust-based composition): Because the DTS is composed of many sensor services, a judgment needs to be performed about the overall trust of the DTS from the trust-specification of the selected services. For example, our recent work has used a composition model that considers individual [B, D, U] tuples and interaction patterns between services to predict the trust of the composed distributed system. **Application of Principle 8** (trust evaluation): Finally, the evaluation of the trust of the Webcam service can be performed by its periodic assessment to decide if it is meeting (or not meeting) its role and specification due to either the change of DTS requirements or internal updates within the Webcam services, then take actions needed to either update or replace the services. Similarly, the other entities (Objects Markers (OM), Discovery Service (DS), Control Service (CS), and Fusion Service (FS) in Figure 4) in the DTS can follow this approach, and the trust of each subsystem can be investigated. Therefore, following this model, the overall trust of the DTS (T_{DTS}) will be defined as, $T_{DTS} = T_{SS} \cap T_{OM} \cap T_{DS} \cap T_{CS} \cap T_{FS}$. The earlier discussion includes a sample application of the proposed trust model which used by this dissertation our previous publications provide more details of this approach ([12], [13], [81] [80], [82]).

4.5 Trust Representation in the specification

In general, a specification describes a feature precisely, using a formal technique. Specifically in the software domain, a service specification publishes syntax, semantics, and QoS information to interested users of the service and is periodically verified usually when the service is getting updated. In the context of trusted software systems, a specification should expose, or publish, different attributes including details about trust-related factors to the outside world. Hence, the service providers creates services and publish service metadata (more broadly a service specification) to the outside world. The outside world consists of trusters (i.e., service consumers) who are now possibly exposed to a limited set of information about a trustee. The published information describes the trustee and is approved by the trustee itself or by a trusted third party (i.e., in this case the online service marketplace).

As services and user requirements for these services become complex, such a simple specification and selection scheme is far from being adequate. Hence, A typical service specification now has multiple levels of contracts and this was first advocated by Beugnard et al. [52]. They have proposed four distinct levels in the specification of a service contracts, namely, syntax, semantics, synchronization, and QoS. The levels include functional attributes, (*i.e.*, syntax and semantics contract information), non-functional attributes (*i.e.*, QoS) and synchronization attributes (*i.e.*, concurrent access policies).

These levels are at a varying degree of negotiations – non-negotiable at the type and syntax levels to highly-negotiable at the QoS level. Multi-level specifications do offer more details about the services than the basic contracts. However, they still do not include a formal quantification of the trust values. Hence, it is hard to determine whether a service actually delivers what it promises by merely inspecting its contract. One possible solution to overcome this issue is to define a separate trust contract of each service with appropriate aggregated information about various evidences related to its multiple levels. The concept of trust permeates all these levels and this dissertation proposes a trust contract which refer to other levels of contracts and their attributes. However, service developers are expected to create such a multi-level contracts along with the deployment of their services. It is widely noticed that the developers typically tend to over specify service qualities (either using the basic or multi-level contracts) and the users presume that a service delivers what it specifies, often leading to a mismatch between promises and expectations. Therefore, it is hard to determine whether a service actually delivers the contracts by inspecting a service specifications unless there exist a separate trust contract of that service with appropriate aggregated information.

If we are to include a trust contracts with details as we proposed in the previous subsection, it will be quite different from the existing contracts such that the specification can contain different trust representations about its published attributes relating to its other contracts such as functional and non-functional contracts. For example, a service specification can indicate as an attribute which indicates the observed response time is between [10, 30] milliseconds. The trust representation of the same attribute, however, can be represented using (B, D, U) as (0.9, 0, 0.1), or a user rating of (4.5/5.0), or a confidence of recommendations as 90%. A trust specification for a software service, therefore, can contain multiple trust representations for different attributes.

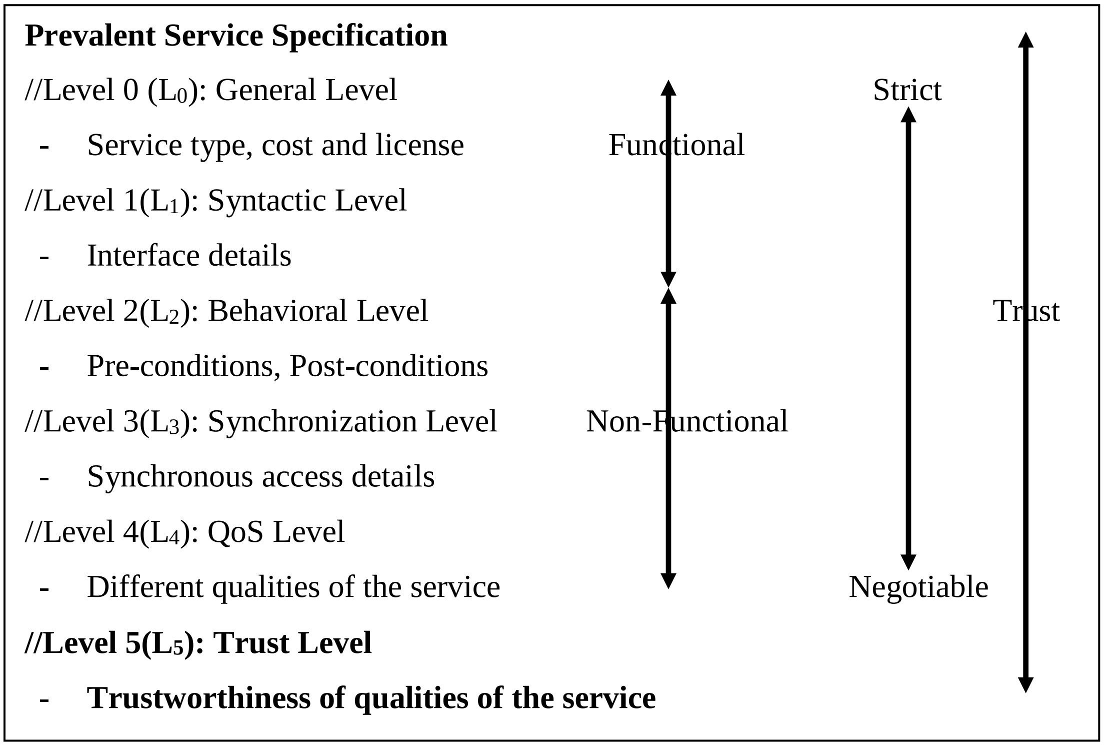
These services get updated frequently hence, their features and qualities change over time. As a result, the trust of the software also fluctuates over time. For example, this dynamic and fuzzy nature of software is visible in emerging software marketplaces (e.g., Android-based mobile applications that are available from the Google Play Store). Over time, the software in such marketplaces is updated and frequently becomes pushed to the clients. When such updates happen, the tones of user reviews may change. Our previous work [89] [88], [90] describe a scheme to evaluate user reviews and recommend software based on their calculated trustworthiness. The work treats Android-based applications as software services and quantifies the trust associated with a service. This quantification of the trust is achieved by monitoring and aggregating various available evidences about Android applications from text-based user reviews. The usefulness of this approach is shown by the quality improvements carried out through the service selection process.

A first step in trusted service selection and recommendation is to require the individual service specifications to incorporate a proper quantification of trust. However, current approaches to service specification ignore this trust aspect and use semi-formal techniques for describing a set of functional and QoS attributes with corresponding values. After these specifications are deployed, along with the services instances, a typical selection process uses simple matching schemes to compare available service specifications against user provided requirements. This is the simplest form of service specification and selection. We refer to such a service specification as the *basic contract* of a service. However, the important questions for the service consumers are: (1) does the service always deliver its functional attributes, with associated QoS, described in its specification? and (2) is the service selection mechanism capable of incorporating the principles of trust while delivering relevant services to its consumers for specific queries?

Hence, this dissertation proposes an evaluation model and provides the necessary basis for trust quantification, addresses a number of issues present in other representations (such as capturing uncertainty), and is simple enough to be used extensively in trust calculations of large and complex software system. Based on this reasoning, this dissertation considers trust-based evaluation to be another basic principle that affects all the other principles and should be considered as early as possible during the development of software-intensive systems.

4.5.1 Structure of the proposed modifications to service specification

Complex queries impose higher weightages on a particular service requirement such as low power usage for a mobile weather application (i.e., an example service from an online marketplace). Therefore structure and organization of information published in the service specification plays an important role. Providing a set of attributes and values as the service specification and matching it against a user provided values is the simplest form of service selection. As services become complex, the organization of the service specification is important for the performance of the service selection process.



Prevalent Service Specification

//Level 0 (L_0): General Level

- Service type, cost and license

//Level 1(L_1): Syntactic Level

- Interface details

//Level 2(L_2): Behavioral Level

- Pre-conditions, Post-conditions

//Level 3(L_3): Synchronization Level

- Synchronous access details

//Level 4(L_4): QoS Level

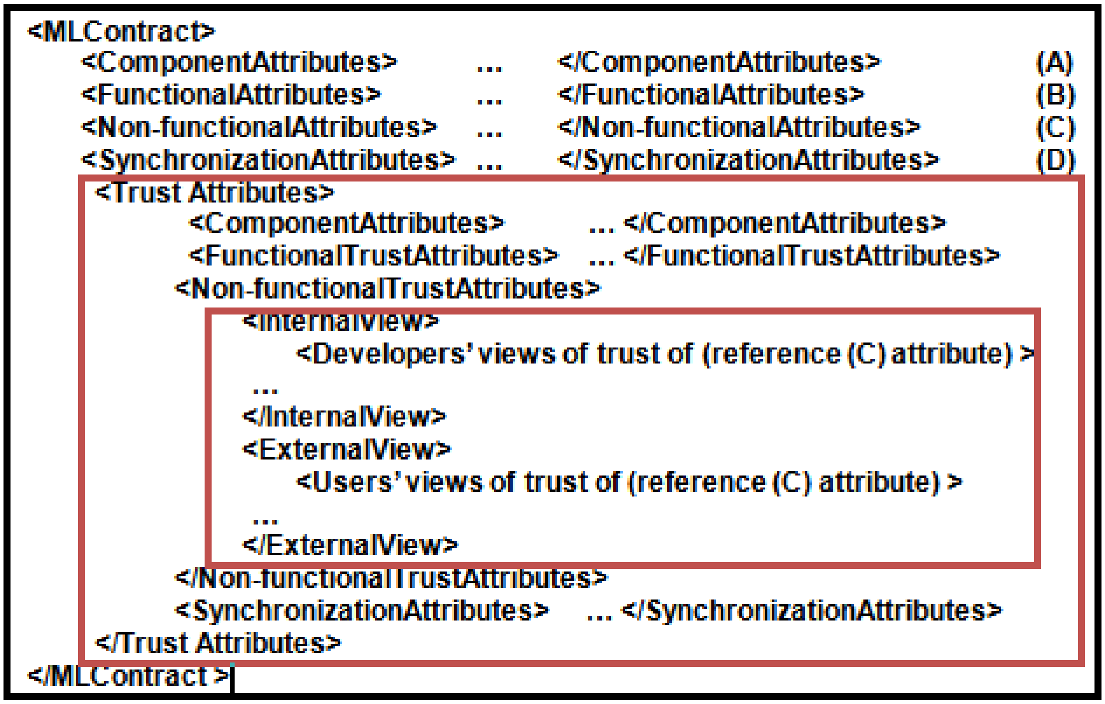
- Different qualities of the service

//Level 5(L_5): Trust Level

- **Trustworthiness of qualities of the service**

Figure 4.6 Structure of the modified multi-level service specification [90]

Multi-level specifications do offer more details about the services than the basic contracts. However, they still do not include a formal quantification of the trust values. Hence, it is hard to determine whether a service actually delivers what it promises by merely inspecting its contract. One possible solution to overcome this issue is to define a separate trust contract of each service with appropriate aggregated information about various evidences related to its multiple levels as indicated in Figure 4.6.



```
<MLContract>
     <ComponentAttributes>          ...    </ComponentAttributes>            (A)
     <FunctionalAttributes>         ...    </FunctionalAttributes>           (B)
     <Non-functionalAttributes>     ...    </Non-functionalAttributes>       (C)
     <SynchronizationAttributes>    ...    </SynchronizationAttributes>      (D)
      <Trust Attributes>
               <ComponentAttributes>         ... </ComponentAttributes>
               <FunctionalTrustAttributes>   ... </FunctionalTrustAttributes>
              <Non-functionalTrustAttributes>
                    <InternalView>
                         <Developers' views of trust of (reference (C) attribute) >
                     ...
                    </InternalView>
                    <ExternalView>
                         <Users' views of trust of (reference (C) attribute) >
                     ...
                    </ExternalView>
              </Non-functionalTrustAttributes>
              <SynchronizationAttributes>    ... </SynchronizationAttributes>
      </Trust Attributes>
</MLContract >
```

Figure 4.7 Structure of the proposed specificaiton with added trust contract [88] [90]

Such an enhancement will enable to answer the two questions (‘1’ and ‘2’) raised earlier. Since, there is no way to quantify the trust about published service metadata, the current service delivery and selection mechanisms tend to largely ignore both these questions. Such an approach is clearly not acceptable as more and more complex systems, composed out of available services, are demanding high-confidence. To address this issue, the TruSStReMark framework is proposing a trust contract which aggregate the evidences based on the trust model described in the previous subsection. These aggregated evidences are numerically quantified using subjective logic to be presented near the QoS attributes of the service and periodically being updated.

Hence, as a part of the specification, it is necessary to a separate trust contract (related to the other levels) of the service specification to indicate the trust-related evidences and opinions Figure 4.6. With a correct organization and aggregation of trust, the service selection process can monitor and aggregate the success or failures of the service as a measure of delivering what is mentioned in its specification both as a single value and also in finer levels of detail. If a comprehensive trust contract is available as a part of the service specification, some smartphone service developers who’s target is to earn by forcefully display advertisements can be identified. For example, mobile battery life is targeted by developers to trick the users to install custom apps by informing their app charges the battery. The evidences aggregating in the trust contract can detect that the service is not delivering what is specifies. However, the challenge is to control, manage and filter exploding number of evidences.

4.5.2 Structure of the proposed Trust Contract

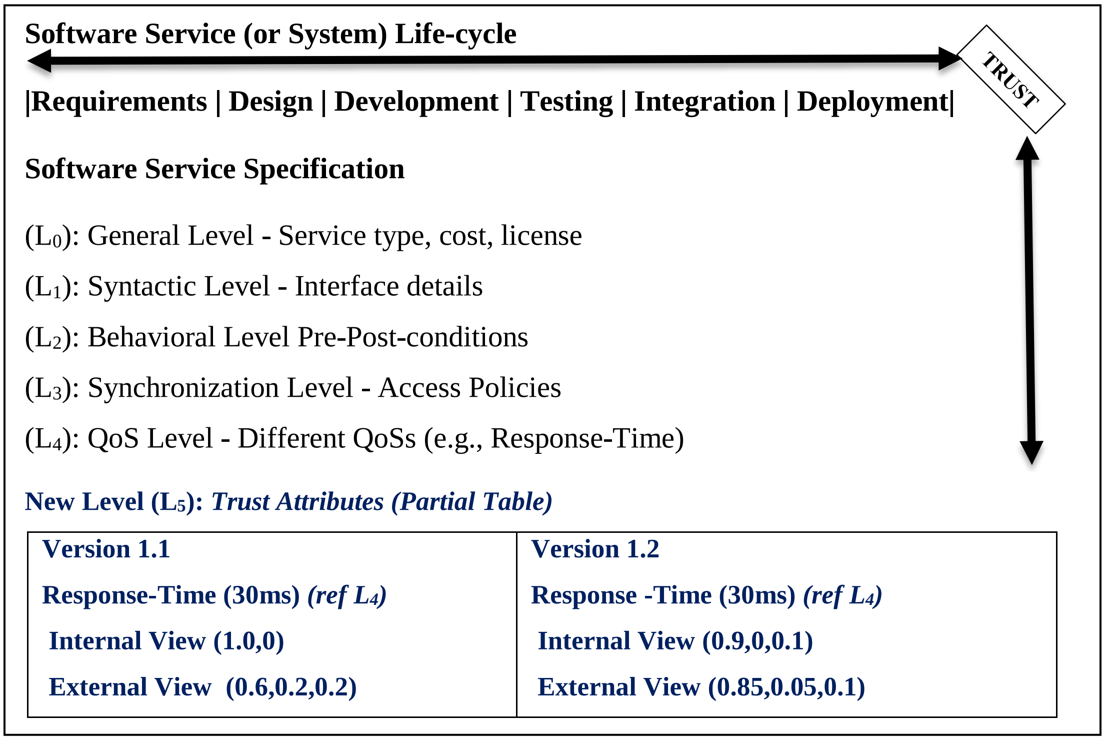
There could be multiple approaches to represent trust attributes within a service contract. One approach is to add trust related information along with the other information of the multi-level contract. This approach, however, breaks the progression of contract information (from non-negotiable to dynamically negotiable). To preserve the subjective and negotiable nature of the trust, it is more appropriate to define the trust contact as a separate level and associate it with other levels of multi-level specification as shown in

Figure 4.7. Our multi-level specification is similar to the one preposed by Beugnard et al. [52] and consists of four levels (*i.e.*, component (general), functional (syntactical and behavioral semantics), non-functional (QoS) synchronization and trust). We augment these four levels with an initial level that describes bookkeeping information about a service (e.g., version number) and the attributes making up that level are called as component attributes.

- S_i is typically arranged into levels L = (L_0, L_1 …. $L_{5...}$).
- Each service parameter P_i has an associated level from $L_{j.}$
- For example, if Quality of Service (QoS) is considered as L_4, and accuracy (P_0) is considered as: $P_0 \in L_4$.

The trust aggregation is always in relation to a reference of another level. Theory of evidence [22] and subjective logic [23] are used to aggregate trust and represented as trust contact of a service. Based on our trust model presented in the previous subseciton, a tuple of B,D,U (Belief, Disbelief, and Uncertainty) is calculated using evidences of a property of an artifact (i.e., conceptual or physical outcomes of a particular phase of the lifecycle).

Figure 4.7 indicates how the test contract refers to each section of the existing contract and include bother internal and external views of evidences. For example, a mobile app could indicate the battery usage as a non-functional attribute at the level C. Based on the current evidences (such as developer testing results) available internally, the developers can calculate a B,D,U value such as (0.95,0,0.05). Although, the internal view of trust tend to over specify based on developers confidence, the external view aggregates and shows users views on the app’s battery life based on evidences they provide as an aggregated B,D,U value such as (0.6,0.3,0.1). The aggregation of these individual BDU tuples is done by subjective logic operators such as the conjunction and consensus. For example, when multiple opinions are present, such as (e.g., consistent or conflicting user reviews about Response-Time of a service) then the consensus operator is used to aggregate these opinions.



Software Service (or System) Life-cycle

|Requirements | Design | Development | Testing | Integration | Deployment|

Software Service Specification

(L_0): General Level - Service type, cost, license

(L_1): Syntactic Level - Interface details

(L_2): Behavioral Level Pre-Post-conditions

(L_3): Synchronization Level - Access Policies

(L_4): QoS Level - Different QoSs (e.g., Response-Time)

New Level (L_5): *Trust Attributes (Partial Table)*

Version 1.1	**Version 1.2**
Response-Time (30ms) *(ref L_4)*	**Response -Time (30ms) *(ref L_4)***
Internal View (1.0,0)	**Internal View (0.9,0,0.1)**
External View (0.6,0.2,0.2)	**External View (0.85,0.05,0.1)**

Figure 4.8 Dynamic nature of the Trust Contract [89] [90]

It is expected that the trust contract will change over time. A well-accepted method to address this change is to update the contract at each milestone associated with the software service (*i.e.*, with each new version). Considering history, however, is an important part in the process of reasoning about trustworthiness of any service. The trust contract therefore should allow for maintaining the history associated with the service unlike the other levels of the service contract. This dynamic nature of trust can be captured by the trust contract proposed by the TruSStReMark framework as indicated in Figure 4.8. For example, the BDU tuple of Response Time is computed as a reference to another level attribute (i.e., referring to QoS attribute) and placed as a part of trust contract according to an agreed temporal aspect (e.g., versions). The calculation of the BDU tuples is done in both internal (i.e., developers) and external (i.e., users) perspectives with respective to that particular attribute of the specification.

4.5.3 Benefits of Long term aggregation of trust attributes.

As mentioned in the previous subsection, aggregation and monitoring of different trust attributes (related properties from Levels 0-4 of the service specification) provide the following benefits for the service selection and recommendation process. The trust aggregation process is general to be included in to any service representation and selection technique. Also, the aggregation process does not impose any restrictions on the software development or reviewing processes. It also does not alter the existing service specification details, however, it proposes to periodically update a separate trust contract based on the new evidences. The problems of over specifying and under specifying the trust are normalized in the long-run over time.

The aggregation process uses the operators presented in Table 4-3. to combine trust values given to different evidences. However, these operators work best when they are presented with lot of evidences. Long-term monitoring of services should generate lot of evidences when the services are used by the general public (e.g., number of services reviews generated for a mobile app by the external users in a service marketplace is now considered as a big data aggregation problem). However, the problem with evidence based trust evaluation is over (or under) specifying of evidences. If the majority of the evidences are not over (or under) specified then the trust aggregation can be normalized by the other evidences by considering them as the outliers. For example, while aggregating the user reviews of a service, if one reviewer is observing that the app does not last for 2 hours of battery life in his type of device (while all the others agree that the app works for more than 2 hours), then that reviewer's evaluation of trust becomes a clear outlier. However, there exists a small risk of considering a legitimate review as an outlier too.

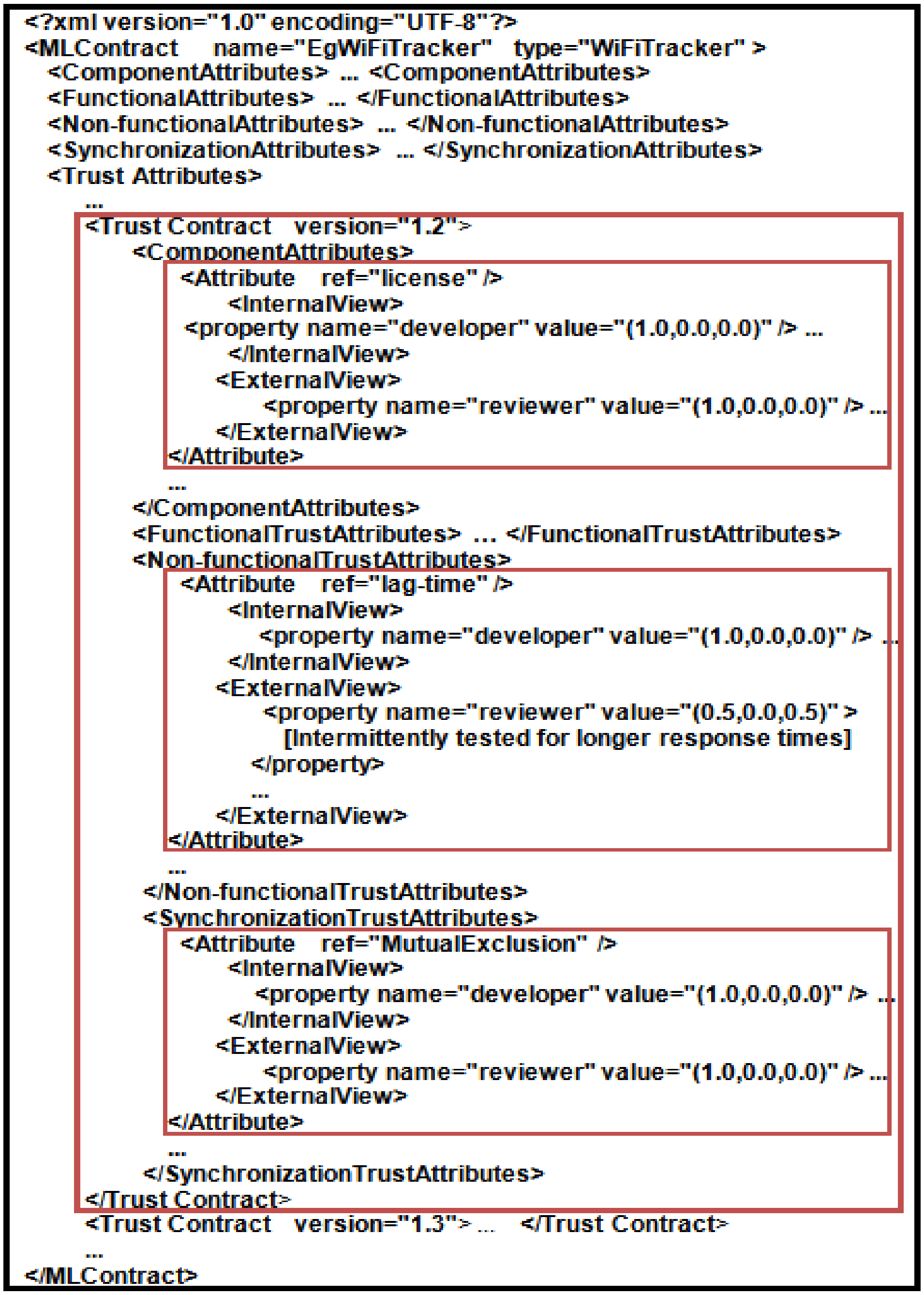
Table 4-3 Definitions of subjective logic operators and their aggregations [23] [37]

Operator	**Definition**	**Aggregation (Given B,D,U of p and q as T_p = [$b_p \vert d_p \vert$ u_p]and T_q =($b_q \vert d_q \vert u_q$)**
Conjunction	Used to aggregate two opinions to reflect the conjunctive truth of both.	$T_{p\wedge q}$ = [$b_{p\wedge q} \vert d_{p\wedge q} \vert u_{p\wedge q}$] where, $b_{p\wedge q} = b_pb_q$ $d_{p\wedge q} = d_p + d_q - d_pd_q$ and $u_{p\wedge q} = b_pu_q + u_pb_q + u_pu_q$
Disjunction	Used to aggregate two opinions to reflect the disjunctive truth of both.	$T_{p\vee q}$ = [$b_{p\vee q} \vert d_{p\vee q} \vert u_{p\vee q}$] where, $b_{p\vee q} = b_p + b_q - b_pb_q$ where $d_{p\vee q} = d_pd_q$ and $u_{p\vee q} = d_pu_q + u_pd_q + u_pu_q$
Consensus	Used to combine many independent opinions about the same proposition into a single opinion.	Let T_p and T_q be two opinions by agents p and q about the same proposition. $T_{P,q}$ ={$b_{P,q}$, $d_{P,q}$, $u_{P,q}$ } be the opinion such that $b_{P,q} = (b_pu_q+ b_qu_p)/k$ $d_{P,q} = (d_pu_q+ d_qu_p)/k$ and $u_{P,q} = (u_pu_q)/k$ where $K = u_p + u_q – u_pu_q$ such that k=0
Recommen-dation	Used to transfer opinions about one propositions from one evaluation to another	Let p, q and be agents where $T^p_q = [b^p_q, d^p_q, u^p_q]$ is P's opinion bout Q's recommendations. T^{pq} be is Q's opinion expressed in P recommendation T^{pq} = [$b^{pq} \vert d^{pq} \vert u^{pq}$] where, $b^{pq} = b^p_qb_q$ $d^{pq} = b^p_qd_q$ and $u^{pq} = d^p_q+ u^p_q- b^p_qu_q$
Negation	Used to invert opinion keeping the ignorance component unchanged.	$T_{\neg p}$ = [$b_{\neg p} \vert d_{\neg p} \vert u_{\neg p}$] where $b_{\neg p} = d_p$ $d_{\neg p} = b_p$ and $u_{\neg p} = u_p$

Ordering	Used to order opinions based on their strongest belief.	IF T_p and T_q have different (b + u)/(b + d + 2u) ratios
		THEN RETURN opinion with greatest (b+u)/(b+d+2u) ratio
		ELSE RETURN opinion with the least u

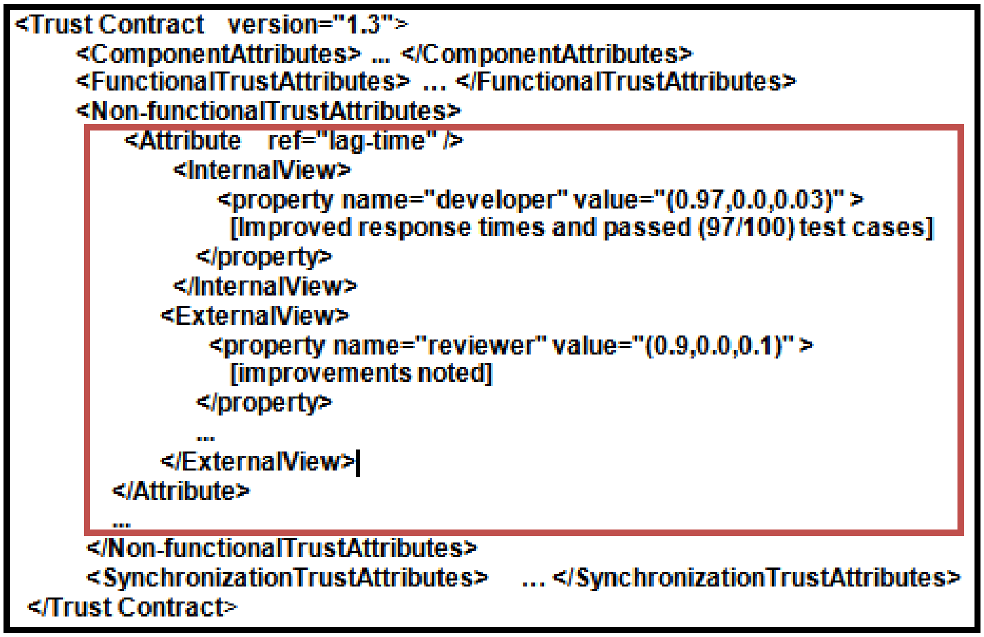
The important advantage of the trust aggregation process is that it keeps track of the behavior of the service with respective to users' trustworthiness. For example, let us consider a service and its weekly collected external reviews (as evidences). Apart from the overall trust aggregation from the beginning, weekly trust can also be independently calculated. If there is a sudden drop/increase of reviewers’ confidence in a particular week about the service, then an alert can be raised and evidences can be further evaluated. By doing this the TruSStReMark framework enables the ability to select the trustworthy service by the service selection process which matches the service requirements together with trust-based requirements. Although the service matches to the given functional and non-functional requirements indicated by the service requirements, that particular service may not be the one which is most valued by the external reviewers whom have the same requirements. Hence, long-term trust aggregation enables this improvement in service selection and recommendation algorithms to find the service which behaves according to all of the users’ requirements [89].

This contract, as indicated earlier, is an aggregation of trust tuples associated with all artifacts generated during the service creation. Now, consider a partial multi-level specification of a Wi-Fi tracker service deployed in a distributed tracking environment as shown in Figure 4.9 which was presented in our previous work [89]. It indicates three types of attributes – component, QoS, and synchronization – with associated (B, D, U) tuples. For example, for the license attribute (which is a component attribute of Wi-Fi tracker), both views indicate same value of the (B, D, U) tuple – the value of the tuple being (1.0, 0.0, 0.0). Such a consensus indicates that the evidence presented for that attribute shows there is no degree of disbelief or uncertainty associated with that attribute. In the case of the license attribute, it is expected that the developer’s documentation is the evidence and accepted by the external users. In contrast to this attribute, the QoS attribute (lag time) of the Wi-Fi tracker shows two significantly divergent opinions—the (B, D, U) tuple indicated by the developer (shown as the internal view in Figure 4.9) does not match with the (B, D, U) tuple evaluated by the users. Figure 4.9 shows the trust associated with the synchronization attribute and that case is in between the two cases of license and lag-time attributes.



```
<?xml version="1.0" encoding="UTF-8"?>
<MLContract   name="EgWiFiTracker"   type="WiFiTracker" >
  <ComponentAttributes> ... <ComponentAttributes>
  <FunctionalAttributes> ... </FunctionalAttributes>
  <Non-functionalAttributes> ... </Non-functionalAttributes>
  <SynchronizationAttributes>  ... </SynchronizationAttributes>
  <Trust Attributes>
      ...
     <Trust Contract   version="1.2">
         <ComponentAttributes>
             <Attribute   ref="license" />
                 <InternalView>
             <property name="developer" value="(1.0,0.0,0.0)" /> ...
                 </InternalView>
                <ExternalView>
                    <property name="reviewer" value="(1.0,0.0,0.0)" /> ...
                </ExternalView>
            </Attribute>
            ...
         </ComponentAttributes>
         <FunctionalTrustAttributes> ... </FunctionalTrustAttributes>
         <Non-functionalTrustAttributes>
             <Attribute   ref="lag-time" />
                 <InternalView>
                    <property name="developer" value="(1.0,0.0,0.0)" /> ...
                 </InternalView>
                <ExternalView>
                    <property name="reviewer" value="(0.5,0.0,0.5)" >
                      [Intermittently tested for longer response times]
                   </property>
                   ...
                </ExternalView>
            </Attribute>
            ...
          </Non-functionalTrustAttributes>
          <SynchronizationTrustAttributes>
             <Attribute   ref="MutualExclusion" />
                 <InternalView>
                   <property name="developer" value="(1.0,0.0,0.0)" /> ...
                 </InternalView>
                <ExternalView>
                    <property name="reviewer" value="(1.0,0.0,0.0)" /> ...
                </ExternalView>
            </Attribute>
            ...
          </SynchronizationTrustAttributes>
     </Trust Contract>
     <Trust Contract   version="1.3"> ...   </Trust Contract>
     ...
</MLContract>
```

Figure 4.9 Partial multi-level contract of a Wi-Fi service [89]



```
<Trust Contract   version="1.3">
     <ComponentAttributes> ... </ComponentAttributes>
     <FunctionalTrustAttributes> ... </FunctionalTrustAttributes>
     <Non-functionalTrustAttributes>
         <Attribute   ref="lag-time" />
             <InternalView>
                 <property name="developer" value="(0.97,0.0,0.03)" >
                   [Improved response times and passed (97/100) test cases]
               </property>
             </InternalView>
            <ExternalView>
                <property name="reviewer" value="(0.9,0.0,0.1)" >
                  [improvements noted]
               </property>
               ...
            </ExternalView>
        </Attribute>
        ...
      </Non-functionalTrustAttributes>
      <SynchronizationTrustAttributes>   ... </SynchronizationTrustAttributes>
 </Trust Contract>
```

Figure 4.10 Wi-Fi Service behavior over time [89]

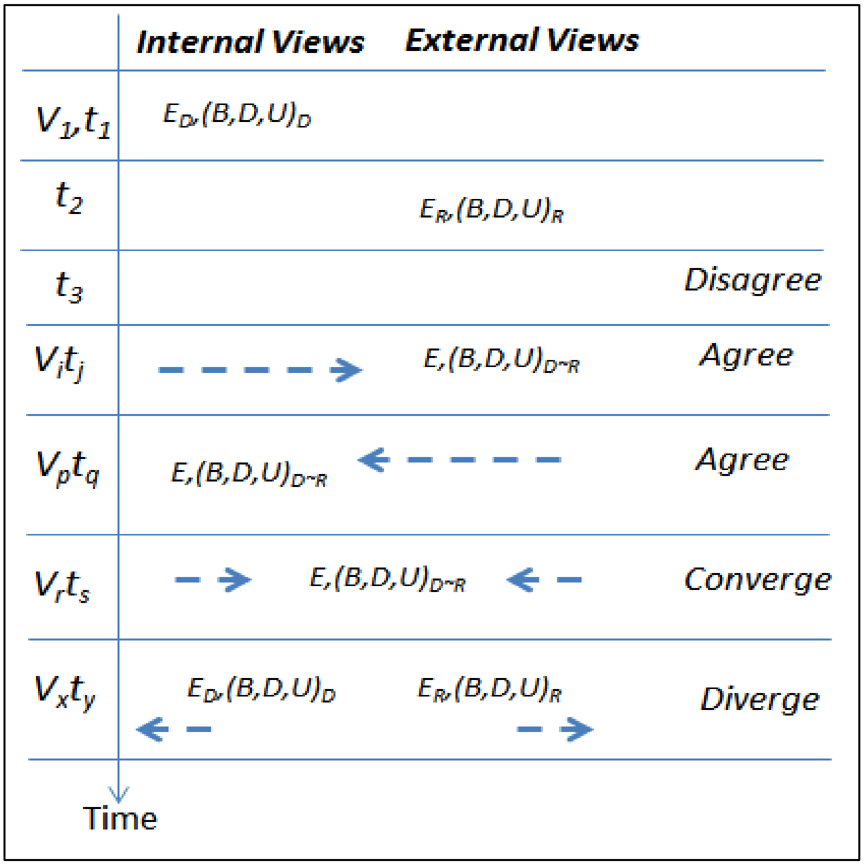


Figure 4.11 Consensuses and disagreements of internal and external views [89]

As indicated earlier, the trust needs to be monitored over a period of time and necessitates the need to store history associated with a service. Figure 4.9 and Figure 4.10 show the variation of the trust contract (and history) over time for the Wi-Fi tracker. The current version of the contract is 1.3 (Figure 4.10) and the user is interested in lag-time of the service. By looking at the service, the trust attributes for versions 1.2 and 1.3, the user can track the direction of movement of trustworthiness (*i.e.*, convergence or divergence). As displayed in version 1.3 (Figure 4.10), developer's view has reduced the values in the trust tuple, but at the same time, the reviewers have improved their view and that is evident in the higher value of the trust tuple.

At a particular time instance, the aggregated values of the (B, D, U) tuples representing the internal and external views can be in a state of either agreement or disagreement (Figure 4.11). If they are in agreement, it shows that both the developer and users have a very high confidence about the behavior of a particular service. On the other hand, there are many possible reasons (*e.g.*, different interpretations of the evidences associated with trust attributes) that will result in the divergence of trust tuples indicating the internal and external views. In such cases, a developer may decide to present further evidence, or modify the service to provide a new set of evidences that may change the trust tuple value associated by the external uses. The proposed trust contract therefore provides an effective way to capture the trust associated with each attribute and provides an automatic way of determining the evolution of trust over a time period.

In summary, the following are the identified strengths and benefits of the TruSStReMark framework's ability to long-term monitor and aggregate the quantified trust-based attributes of services for the service selection and recommendation processes.

- The trust aggregation process is general and can be included into any service representation and selection technique.
- The problems of over specifying and under specifying the trust are normalized in the long-run over time.
- To track the trust progression and reason of trust variations over time of different views.
- The ability to select trustworthy apps by the service selection process which matches the service requirements.

4.6 Quantification and Specification of Trust for Apps

An android marketplace is a location where the details about the apps related to portable devices are published. In this case study, TruSStReMark is experimented with a selected subset of service types from the Android Marketplace in relation to a trip planning system. The goal of this case study to show the effectiveness of the long-term aggregation and monitoring of the trust-related attributes. Any service in this marketplace has two views – internal (i.e., developer's view) and external (i.e., user's view). The former indicates the confidence that a creator has in the behavior of her service, while the latter depicts the first-hand experiences of users of that service.

Therefore, the computation of the trustworthiness of a service must consider both these views. These users act as reviewers when they express their views (e.g., text and numerical quantifications) in associated public forums (e.g., Software Marketplaces). We use the terms "users" and "reviewers" interchangeably. Due to the limited availability of internal evidences, mainly external reviews are considered as aggregating evidences. If available, the internal evidences are also used in the calculations of the internal view of trust.

However, it was not used as a long-term monitoring criteria, since the values are aggregated from a limited set of evidences as compared to external reviews of the service. Therefore, in this case study only external reviews available about these apps are considered. The selected service dataset includes 36204 apps and 1108343 reviews. We used a default value for the internal view of apps, since the internal evidences are not available.

4.6.1 Case-study from an online marketplace

The assumption for the scenario is that all the users are equipped with android capable devices and are connected over the network. Instantly, a group of users decide to plan a trip using apps available from the android marketplace. To speed up the planning process, they assign each member with different tasks related to planning (Figure 4.12) such as travelling (flights and rent a car), maps and navigation, weather and traffic, hotels and reservations and financial (gas, banks, etc.).

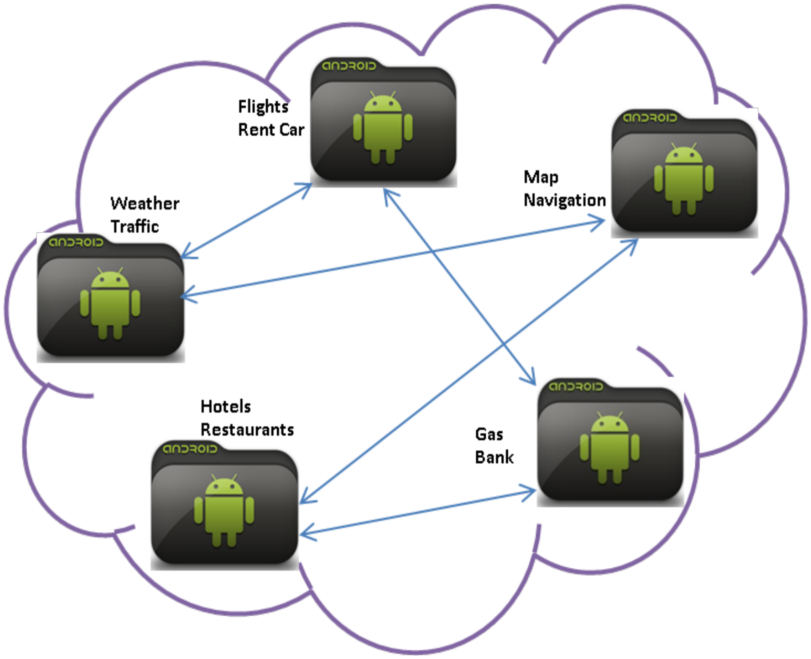


Figure 4.12 Abstract scenario of trip planing collaboration using android devices and apps from the android marketplace.

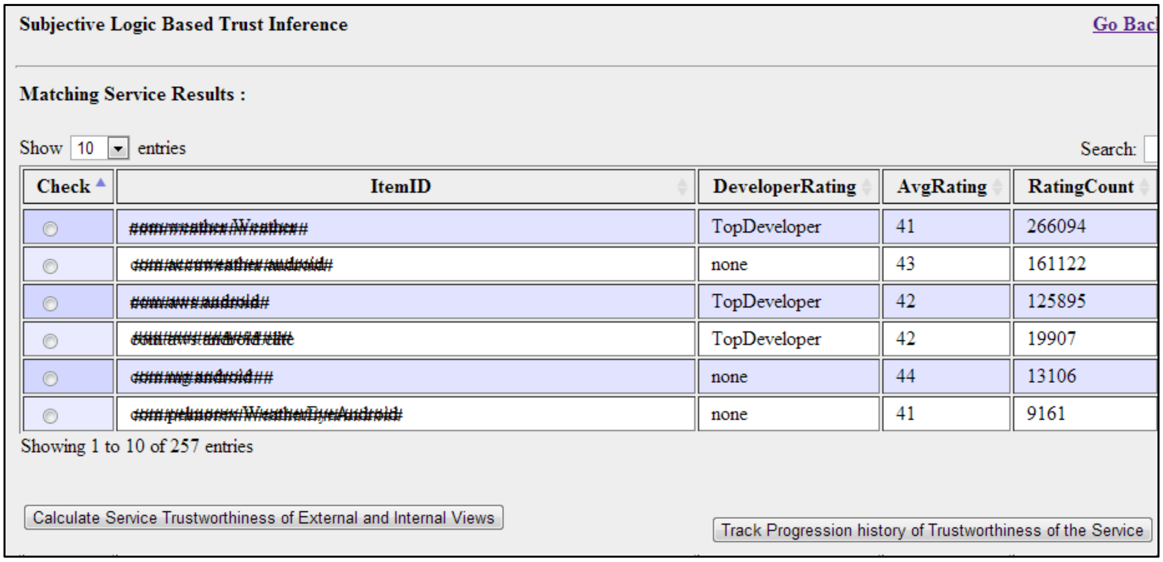
The users need to select the best apps from the marketplace for their requirements and the trust (of the selected apps) play an important role in such scenario. If any of the apps that they have chosen fail to provide what the users require, then the plan would have to be redone. Although the developers mention that they provide reliable apps as shown in their specifications, here are some examples which the apps may fail to meet the trust requirements of this set of users (Figure 4.12). By showing routes via currently overbooked flights, by showing outdated location information from maps with non-existing point of interests (hotels, banks, gas stations, restaurants, etc.) and by forecasting inaccurate weather and traffic.

In this kind of scenario, the only way the users can be confident about a service is by accurately matching and selecting apps according to their trust requirements (in addition to their functional and non-functional requirements) based on the evidences provided by others who were in the same situation before. Therefore, the results described in the following subsection show how TruSStReMark can improve the service monitoring process by performing long term trust monitoring and aggregation of reviews.

4.6.2 Analyses of Data Aggregation and Presentation

The TruSStReMark framework continuously monitors and aggregates trust of a selected sub-set of apps from the Android marketplace. These apps are selected as the possible candidates for the scenario mentioned above. When the data i.e., service descriptions and reviews) is loaded to the TruSStReMark, the goal is to track the progression of trust for each service, based on specific temporal intervals (i.e., versions, weeks) and select proper set of apps.

Although, the data we extracted from the marketplace is publically available for everybody to see, due to conflicts of interest and necessity to preserve privacy, we had to censor out (i.e., distort) some of the information (i.e., apps names and personal details) shown in the following set of images. When a service query is submitted, TruSStReMark displays a list of possible instances (for each type of service needed, e.g., travel apps).



Subjective Logic Based Trust Inference

Go Bac

Matching Service Results :

Show 10 entries

Search:

Check	ItemID	DeveloperRating	AvgRating	RatingCount
○	[redacted]	TopDeveloper	41	266094
○	[redacted]	none	43	161122
○	[redacted]	TopDeveloper	42	125895
○	[redacted]	TopDeveloper	42	19907
○	[redacted]	none	44	13106
○	[redacted]	none	41	9161

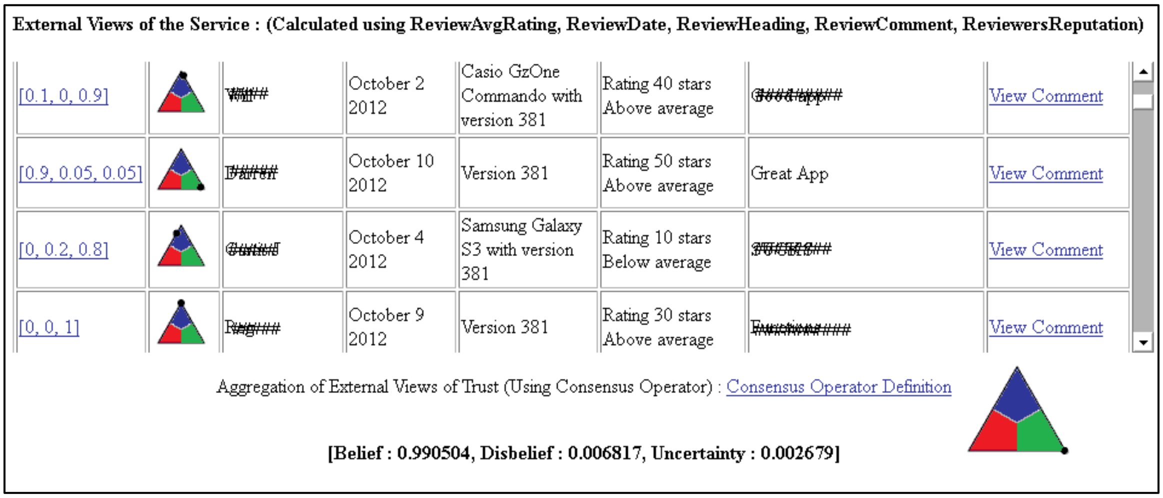
Showing 1 to 10 of 257 entries

Calculate Service Trustworthiness of External and Internal Views
Track Progression history of Trustworthiness of the Service

Figure 4.13 Apps for a type of service are ready for evaluation of their trust (with referencing the scenario in Figure 4.12).

Figure 4.13 displays a screenshot of TruSStReMark which provides a list of apps (for a particular type of service needed for the scenario presented in Figure 4.12) after matching their other requirements. This list is used to evaluate trust of interested apps by the users. From this page, the users can select a service either by evaluating them based on the table details or decide to select interested apps after further evaluating them.

From the external reviews TruSStReMark can calculate trustworthiness of each service based on associated evidences and also can track the progression of trust. Figure 4.14 presents a screenshot of how the trust of a selected service (from the list of apps in Figure 4.13) can be aggregated based on the external reviews. In each row, the table displays each review along with their calculated trust using the sample aggregation algorithm presented in Figure 4.15.



External Views of the Service : (Calculated using ReviewAvgRating, ReviewDate, ReviewHeading, ReviewComment, ReviewersReputation)

[0.1, 0, 0.9]	▲	[redacted]	October 2 2012	Casio GzOne Commando with version 381	Rating 40 stars Above average	[redacted]	View Comment
[0.9, 0.05, 0.05]	▲	[redacted]	October 10 2012	Version 381	Rating 50 stars Above average	Great App	View Comment
[0, 0.2, 0.8]	▲	[redacted]	October 4 2012	Samsung Galaxy S3 with version 381	Rating 10 stars Below average	[redacted]	View Comment
[0, 0, 1]	▲	[redacted]	October 9 2012	Version 381	Rating 30 stars Above average	[redacted]	View Comment

Aggregation of External Views of Trust (Using Consensus Operator) : Consensus Operator Definition

[Belief : 0.990504, Disbelief : 0.006817, Uncertainty : 0.002679]

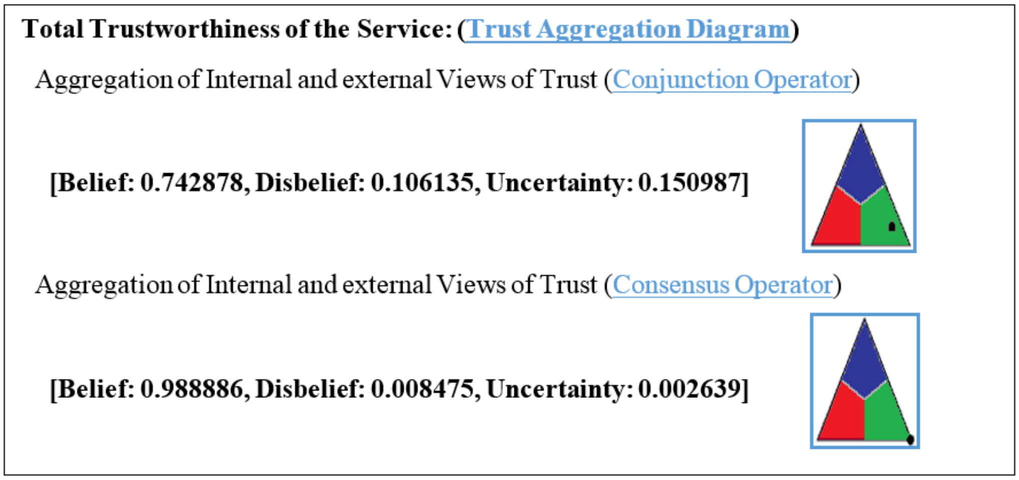
Figure 4.14 Week by Week Monitoring of Trust using B,D,U based evidences

```
External View: BDU_Aggregation_Algorithm
Input: User reviews of a service (optional date range)
Output: Aggregated trust value of the service (B, D, U)
('AvgRating' out of 5)%
For Each REVIEW
        (('ReviewDate'-'ServiceFirstAvailableDate') /
                ('Today'- 'ServiceFirstAvailableDate'))%
Evidence based B adjustment %
Evidence based D adjustment %
Evidence based U adjustment %
Recommendation Adjustment %
  [Symbol % indicates when the specific parameter value is applied as an adjustment
                          based on a pre-agreed weightage]
```

Figure 4.15 Structure of the algorithm used for review aggregation using SL

The resulted value of trust (B,D,U tuple) are displayed graphically by using subjective logic-based BDU triangle. Each edge of the triage represents each extreme view of trust (Green edge – [1, 0, 0], Red edge – [0, 1, 0], Blue edge – [0, 0, 1]) and other area inside the triangle represents any value between them.Finally, the total trust aggregation is calculated using on the consensus operator shown in Table 4-3 and is displayed using both numerical and graphical representations. When a large set of independent opinions are available, then the consensus operator can be used to aggregate even conflicting opinions to make an overall judgment.

Figure 4.15 presents the algorithm used to aggregate trust based on external reviews about a service. Based on individual service attributes which are present in user reviews, the B,D,U adjustments are calculated. For example, as displayed in Figure 4.15, starting from total trustworthiness [1,0,0], first the trust is adjust based on the apps' average user star rating (based on previously agreed importance of the average rating as a weightage).



Total Trustworthiness of the Service: (Trust Aggregation Diagram)

Aggregation of Internal and external Views of Trust (Conjunction Operator)

[Belief: 0.742878, Disbelief: 0.106135, Uncertainty: 0.150987]

Aggregation of Internal and external Views of Trust (Consensus Operator)

[Belief: 0.988886, Disbelief: 0.008475, Uncertainty: 0.002639]

Figure 4.16 Total trustworthiness of a service is presented using two operators (conjunction and consensus from Table 4-3) [88]

The evaluators can include any number of evaluation criteria either at the service level or at the reviews level. After the service level trust aggregation, next the trust is adjusted based on each review. The novelty of the review is calculated and depending of the value of that the impact of this review is determined. Calculating the impact of the textual review can be based on simple techniques (such as keyword counting) or complex as theory of information retrieval domain (such as natural language processing techniques, later which

TruSStReMark adopted). We did not specify a particular technique to determine the impact of a review with the algorithm initially. Hence, this approach is general enough to use any suitable technique (form the domain of machine learning and information retrieval).

Figure 4.16 presents a screen capture of TruSStReMark indicating the computed trust value of a selected service. We use two methods for calculating the total trustworthiness of a service. The consensus operator is preferred when the evaluated views fall within a range. Otherwise the conjunction operator is preferred. Final aggregation values are displayed using both numerical and graphical representations. We also use a BDU triangle which is similar to the visual representation used by Jøsang et al. in [23]. If the information related to internal view is available, it also can be calculated using similar algorithm. The important thing to note is that the subjective logic operators (from Table 4-3) can be used to aggregate the evidences to evaluate trust depending on the situation. For example, again Figure 4.16 presents two ways of calculating the total trustworthiness of a service by using different operators to aggregate the values of internal and external views of trust.

Figure 4.17 presents a screen capture from the trust progression page of the framework which provides the weekly summarized trust of a selected service. Using the BDU triangles sorted by weeks, this page graphically shows fluctuations of the trust (of a service) based on the service behavior in the past. The knowledge gained by observing this progression helps to make important decisions during service ranking and selection. For example, week 2 of the service indicates an abnormal value of trust compared to other weeks. If the users encounter such behavior, then they can further inspect the events related that particular time period of the service to conclude the causes. For example, in this case, the service released an updated version during that week and the users (who upgraded) experienced major problems.

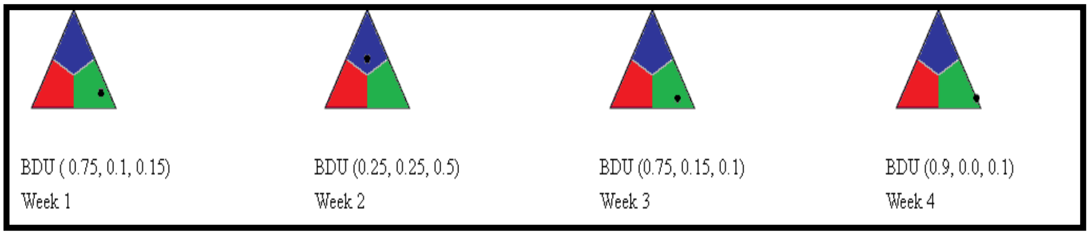


Figure 4.17 Week by Week Monitoring of Trust using B,D,U aggregations from evidences available within that time period

Then we were able to find out that the issues are due to presence of newly promoted user interfaces. However, the developers acted quickly and released another version within a week and was able to solve the previously reported issues which degrades the trust of the service. Therefore, within the next week (the week 3) the posted reviews got back to within the normal range of trust (tracked by the framework during the history of the service). Considering the types of apps needed for the abovementioned case study, the users could avoid a service with such issues by after selecting their service using this framework and benefit by having more confidence. Because within the framework selected set of apps, it is very unlikely that there will be any new release related issues (having seen the history and trust progression of these apps) during their time of planned vacation.

A reviewer may introduce apps to other reviewers. For example, a reviewer may ask their collaborators about a particular type of service and receive a set of choices. This situation requires the use of the recommendation operator to transfer an opinion (as indicated by the corresponding (B, D, U) tuple) about the service from one reviewer to another [89]. The asking reviewer can then use the transferred opinion in the matching process. If many conflicting opinions are present then the negation operator (from Table

4-3) can be used to negate of one or more opinions about another opinion by inverting while keeping the ignorance component unchanged [89]. For example, a user requests a service with response time 10 msec or less (req_i). Some reviewers, however, may express their confidence in (B, D, U) for the response time greater than 10 msec ($\sim req_j$). Negating (~) later ($\sim req_j$) reviews and aggregating with the matching reviews (req_i) may provide a better confidence with no change in the ignorance part of the tuple.

The experience gained from this case study gave us an initial empirical validation of the TruSStReMark framework and it provided the foundation for next steps of improvements. There are modifications necessary to the prevalent trust-based service selection and recommendation algorithms to include trust-based matching. These contributions and modifications present in TruSStReMark framework are presented in the next Chapter.

CHAPTER 5. TRUST-BASED SELECTION AND RECOMMENDATION

The work presented here is based on our previous publications [88] [89]. The TruSStReMark framework focuses on improving algorithms that perform trustworthy service representation, selection and recommendation in the context of service marketplaces. The main goal of the framework is to facilitate service reuse and trustworthy integration of software systems. This is achieved by identifying the complete spectrum of trust of apps at the time of service selection and recommendation. For that, it tracks and records the fluctuations of both the apps and their environment using information from customer satisfaction, robust updates and stable new releases. This additional information collected by the framework improves the selection and recommendation processes such that users can make decisions based on the trustworthiness and the reputation of the apps.

Prevalent approaches, which are listed in Subsections 2.3 and 2.4, of product-based ranking and recommendation do not consider information about items beyond attribute-value pairs. For example, one widely accepted technique of ranking, selection and recommendation of items is item-item Collaborative Filtering (CBF). This technique is based on the concept “similar users buy similar items”. It does not consider item details and algorithmic processing on a large (typically very sparse) item-user matrix. Another accepted technique is Content-based Filtering (CBF) for ranking, selection and recommendation. Although it considers item details, CBF first models the item details as quantifiable attribute-value pairs, which may not be possible for every item details.

The final goal of these two techniques (or their hybrid model) is to rank and recommend items using a similarity-function such as cosine similarity. Compared to a product, reusable software-based items (such as, online apps) are hard to describe using a set of attribute-value pairs or using a user-item relationships. Therefore, it is challenging to apply product based ranking and recommendation techniques directly to software. Moreover, the service providers create apps and publish service metadata (more broadly a service specification) that can be used to selection and recommendation of apps. However, the important questions for service consumers are*; (a) Does the service always delivers what is described in its specifications*? and *(b) Is the service selection mechanism capable of providing the consumers with details required to evaluate the trustworthiness of the service?*

5.1 Selection and Recommendation of Trust for Apps

During the matching phase, the service details available in the specification are matched with user requirements. The matching scheme depends on the structure of the specification ranging from simple attribute-based matching to more complex multi-level matching. Our previous work has performed experiments on how Multi-level Matching can be performed on selection software apps with varying operators at different levels [70], [71], [72]. Analysis of different schemes that match both functional and QoS features have been discussed in background section (Subsection 3.1) of this dissertation, which is related to our previous publications ([70], [71], [72]). Most prevalent service specifications do not include a comprehensive trust contract and, do not consider a trust contract and associated operators needed for matching. In contrast, the approach proposed in this dissertation mainly focuses on trust contracts in service selection and recommendations. In the next sections, we present our framework named TruSStReMark.

Service selection is an expensive and time consuming process, especially, when the search space is very large. Therefore, many prevalent methods use heuristic-based approaches to initially prune the search space. Service selection algorithms proposed in most prevalent approaches can be divided in to two phases; the off-line phase and the on-line phase (as described in Figure 5.1).

```
Prevalent Service Selection Algorithms
Input: Service query
Output: Ordered list of matching services.
//Off-Line Phase (periodically updated)
  -   Pre-Processing of a large collection of dataset
//On-line Phase (executed for each query)
  -   Matching service attributes against requirements
  -   Perform Selection and Recommendations
```

Figure 5.1 Structure of a service selection (and recommendation) algorithm with differentiation of on-line and off-line phases

The offline phase performs the pre-processing of the review datasets. This pre-processing phase is a time consuming process and involves periodic modifications to the whole collection of service specifications. It orders and categorizes existing apps. Different techniques such as ordering, indexing, clustering and heuristic based methods are used. In this phase, new knowledge about the apps (such as, how confirmed users have used this service already and what are the most important and most discussed features of the service) is discovered and subsequently represented and stored in a suitable format. This process makes the knowledge quickly accessible for the on-line matching phase. For example, within a service cluster, an ontology can be created to improve the accuracy of the semantics of matching of apps. In contrast, the matching phase quickly selects a subset of apps and matches the input requirements to find suitable apps. However, the two important questions (a), (b) from the previous section are left as afterthoughts during this phase, which we consider as a major drawback.

5.1.1 Structure of Selection and Recommendation Algorithms

One alternative solution is the option given to the users to include a numerical rating such as average star ratings (e.g., out of 5 stars). This method is now commonly available to service users in service marketplaces. However, the star rating system is also similar to having a single QoS attribute as the trust of the service. It is hard to associate a common number that encompasses all aspects of a service by a user who may or may not have used all the aspects of the service. Therefore, the star rating has limited use in answering the questions (a), and (b). Therefore, there exists a need for a new way to long-term aggregate and monitor the trust of a service.

The TruSStReMark framework answers the questions (a), and (b) using augmented trust-based specification and using augmented selection and recommendation algorithms. These modifications are highlighted in Figure 5.2. When compared with Figure 5.1, it is apparent that additional information helps to improve the quality of the results of service selection and recommendation algorithms. Updated service specifications include a trust contract that provides the capability to handle trust requirements of the service queries. These queries can request about the trust requirements associated with either one prominent attribute (such as the battery life) or multiple attributes.

The Off-line phase of the service selection process periodically collects the evidence of trust attributes published by the service specification. The trust aggregation can be performed hourly or daily, as new evidences arrives or as new versions arrives. For example, let us consider a partial service specification as shown in Figure 5.3.

Modified Service Selection Algorithm (modifications*)
Input: Service query (Including trust requirements*)
Output: Ordered list of matching services.
-Optional ordering according to the trustworthiness*

//Off-Line Phase (periodically updated)

- Pre-Processing of a large collection of dataset
- Periodically aggregate trust attributes *

//On-line Phase (executed for each query)

- Trust related matching semantics and operators *
- Matching service attributes against requirements

Figure 5.2 Modified structure of a service selection algorithm with provisions for aggregation of trust attributes and trust related matching semantics [87]

Modified_Service_Specification*
//Level L_0 -L_4 (Skip Display)
//*New Level (L_5): Trust Attributes (Partial Table)

Version 1.1	**Version 1.2**
License – GPL (ref L_0) IV (1.0,0) | XV (1,0,0) Battery Life (2hr) (ref L_4) IV (1.0,0) | XV (0.5,0.4,0.1)	License – GPL (ref L_0) IV (1.0,0) | EV (1,0,0) Battery Life (2hr) (ref L_4) IV (0.9,0,0.1) | XV (0.8,0.1,0.1)

Figure 5.3 Trust contract with aggregated contract versioning [87]

Figure 5.3 indicates the trust representation and aggregation with service versioning and dynamic updates (i.e., including changes proposed to the service specification as indicated in Figure 5.2). Here, the timeframe of trust aggregation is based on the version of the service. For simplicity, only two parameters of the trust contract are shown; License

(From General Level) and Battery Life (Non-functional Level). Version 1.1 of the license attribute indicates the same (B, D, U) tuple for both views (Internal View-IV and External View-EV). This indicates that the evidences (e.g., developer documentation) presented for both the developers and external users' views are similar (i.e., no conflicting opinions exists). However, there exists a significant mismatch between the evidence aggregated for the battery life attribute, and thus shows two conflicting opinions. As indicated earlier, trust needs to be monitored over a period of time and requires storage different evidences of the history associated with a service's attributes. By investigating the trust attributes for versions 1.1 and 1.2, the user can track the trend of trustworthiness (i.e., in agreement, in convergence, or in divergence) [87].

For example, in Figure 5.3 the developer's view of battery life in the version 1.2 has reduced as per its trust tuple. At the same time, the reviewers have improved their view as per the higher value of their aggregated trust tuple. Both the IV and XV can include optional notes for each other, in this case it is noted 'fixed existing energy related issues and 90% of the test-cases succeed' and 'noted battery life improvements' [87]. Having these trust related aggregations collected over a period of time as part of a trust contract provides additional knowledge, which can be used during service selection. Using this modified specification of the trust contract the TruSStReMark framework provides *improvement to the algorithm that allow an efficient and trusted software service search, ranking of selection and recommendation in the context of a typical software marketplace*. The goal is to improve the confidence associated with the service outcomes, obtained from software marketplaces, for any user query. Such confidence will eliminate the 'trial-and-error' situations that many users have to go through, before they finalize a particular software service for their needs.

The calculation of confidence is based on a comprehensive analysis of external evidences available as textual reviews of software apps. These reviews are aggregated on a day-by-day basis from service marketplaces. The volume of these reviews for a given service, increases as the number of service users increases, necessitating an automatic big data analysis. In order to reduce the associated computational overhead, TruSStReMark processes these reviews offline (day-by-day in our current setup or it can be hour-by-hour if resources are available) within a given timeframe using Natural Language Processing (NLP) techniques such as the Sentiment Analysis (SA), and Theory of Belief techniques such as Subjective Logic (SL). The resulting numerical values are found to be sufficient for the future calculations.

Hence, when new reviews arrive, there is no need to go back and process them again, because existing operators available in SL enable us to merge new and old results as required. For example, SA calculates general sentiments of a document for all the named entities. To calculate the confidence of selected set of named entities, SA needs different documents, which only include parts of the document containing relevant information. If the requirement is to calculate confidence about a QoS expressed by the users from the textual reviews (e.g., item descriptions and user reviews), then the SA needs to maintain documents about each interested named entity (e.g., QoS). Furthermore, SA adds new information as it arrives and recalculate sentiments. However, unlike SL, SA does not provide a technique to calculate confidence based on conjunction of opinions or consensus of opinions.

5.1.2 TruSStReMark framework applied to marketplaces

In the context of a software marketplace, the search and selection of a software service is always performed using a given set of requirements (especially, focusing on the QoS features). Therefore, the underlying assumption in this process is that apps are reusable or can always be replaced after their installation when users' requirements change. The open culture of these marketplaces allows many alternatives for a given type of service with the hope that one instance could be replaced by its possibly superior alternative, when such an alternative becomes available in the near future. For example, if a better music playlist manager becomes available in the future then users tend to replace the existing playlist application with this new one while keeping their profile intact (e.g., in the Android Marketplace, users now consider *Spotify* as an alternative to *Pandora*).

While considering such replaceability requirements, users are interested in knowing and comparing different features and the stability of the QoS of various alternatives. In future software marketplaces, it is fair to assume the existence of the following two types of scenarios: i) users who use or replace the existing apps as individual entities (i.e., they do not attempt to relate or compose apps together, and ii) users who select apps with the purpose of integrating them with other apps to create complex systems. The first scenario is similar to searching for physical and tangible products from a marketplace such as Amazon. However, as indicated earlier, the dimensions of a typical physical product and the dimensions of a software service are distinctly different. For example, most product descriptions consider physical features (such as height, weight, length, etc.), along with their measurements. The confidence associated with these measurements is certain and immutable over time. In contrast, software apps consist of more complex features such as different QoS values, and the confidence associated with these measurements are a function of execution the environment and time.

In the second scenario, the selection of a software service happens because that service needs to operate as a piece of a larger, and perhaps, distributed system. For example, a user who has obtained word processing software, such as MS Word, from the Windows Desktop Application Store can benefit from a recommendation related to suitable add-ons. Microsoft lists over 150 apps and add-ons for MS Word, and it is a challenge for such a user to discover the best add-ons with her particular version of Microsoft Word. If apps are to interact with other apps, a simple user-service matrix, incorporated by prevalent approaches to suggest recommendations will fall short, because by using only information about the user-service numerical relationship, the algorithm could not infer only from the user ratings of that service the version compatibility or confidence of the users about a specific QoS of that service.

In such cases, in addition to the replaceability, the concept of compatibility needs to be included during the search and recommendation process. To provide better recommendations, in both of these cases, additional levels of information about the trust of a software service, especially in the context of its dynamic nature or the QoS features, needs to be analysed. TruSStReMark performs this extra analysis using users' reviews from the marketplaces. The TruSStReMark framework augments and enhances the two prevalent approaches (i.e., Content-based Filtering (CBF) and Collaborative Filtering (CLF) which are described in CHAPTER 3).

5.1.3 Conversion of SA to SL

As indicated earlier, the TruSStReMark framework improves confidence while selecting and recommending software apps by aggregating external evidences available as textual reviews. SL provides the necessary operators to aggregate different opinions about a service QoS from different users depending on the situation. As described in the previous Chapter, Table 4-3 indicates the definitions of conjunction, consensus and ordering operators from SL that are used in TruSStReMark. These evidences are aggregated by passing the identified sentences which describe a particular QoS of a service from user reviews. The aggregation and quantification uses SA techniques related to opinion mining strategies. As a part of SA, opinion mining refers to the use of NLP techniques to analyze textual content and use computational linguistics criteria to identify and extract subjective and polarity information. For example, the sentiment of a textual sentence is expressed by SA using a numerical pair of polarization (i.e., the confidence about an identified named entity) with a value between -1 and +1 and subjectivity (i.e., the weight of the expressed polarization) with a value between 0 and +1. In the analysis of textual reviews, we have used a publically available text processing tool called the TextBlob [78] for the NLP-based sentiment analysis. In our approach, we pass textual reviews from users to calculate the sentiment expressed about a particular QoS property of a software service.

Table 5-1 presents four sample sentiments calculated from associated review sentences. The TruSStReMark identifies a set of named entities for users from storing their queries (i.e., which QoS features are important to them). These named entities for apps can be calculated from the keywords, their description, and from user reviews. Given that, and using TextBlob library [78], the TruSStReMark calculates the sentiments of user reviews.

Table 5-1 General and QoS sentiments and relative scores from reviews [91] [92]

Sample Review Sentences	Named Entity	Sentiment Analysis	
		Polarity	Subjectivity
Nice, Good Work.	General	+0.700	0.875
Very bad.	General	-0.350	0.600
I love this apps' UI with large text and figures.	User Interface	+0.357	0.514
I hate this app, it drains the battery so fast and slows down my device.	Resources Usage	-0.252	0.596

The positive polarity of SA is a measure of belief in the textual evidence. Similarly, the negative polarity is a measure of disbelief. Then subjectivity is the indication of certainty when high and uncertainty when low. Therefore, we adapt the following technique for conversion from SA to SL.

Table 5-2 Conversion from Sentiments (Subjectivity, Polarity) tuples to < Belief, Disbelief, Uncertainty > tuples [92]

(P,S)	<B,D,U>		(P,S)	<B,D,U>
(+1,1)	<1,0,0>		(0,0)	<undefined>
(+0.75,0.25)	<0.75,0,0.25>		(-0.25,0.25)	<0,0.25,0.75>
(+0.5,0.5)	<0.5,0,0.5>		(-0.5,0.5)	<0,0.5,0.5>
(+0.25,0.25)	<0.25,0,0.75>		(-0.75,0.25)	<0,0.75,0.25>
(0,0)	<undefined>		(-1,1)	<0,1,0>

The following discussion describes the steps followed for converting the identified QoS using textual sentiments (i.e., 2 dimensions, [polarity [-1,1],subjectivity[0,1]) to Subjective Logic-based B,D,U tuples (i.e., 3 dimensions (Belief[0,1], Disbelief[0,1], Uncertainty[0,1]). The boundary cases in this conversion process are computed as listed in Table 5-2. The in-between values are equally weighted and divided among the relevant entries. The following boundary cases and the remaining cases are calculated by averaging and interpolation using a linear model as described below.

Let A be a matrix in which each row is a case of P, S combination, and the first column is P, and the second column is S. For example, from the table, A should be represented in matrix format as indicated in Table 5-3.

Table 5-3 Matrix representation of the SSA to SL conversion

A = [1, 1	Y = [1, 0, 0
0.75, 0.25	0.75, 0, 0.25
0.5, 0.5	0.5, 0, 0.5
0.25, 0.25	0.25, 0, 0.75
-0.25, 0.25	0, 0.25, 0.75
-0.5, 0.5	0, 0.5, 0.5
-0.75, 0.25	0, 0.75, 0.25
-1, 1]	0, 1, 0]

Let Y be a matrix in which each row is a corresponding case of B, D, U combination, and the 1st column is B, 2nd column is D and the 3rd column is U. From Table 5-3, Y should be (we just need B and D in Y since B, D, U are linearly dependent such that B+D=1-U). Then, the problem is defined as to convert from A to B. Assume the conversion follows the following linear form:

A * X = Y,

where X is the conversion matrix that the solutions is looking for. We formulate the solution as a multivariate linear regression [91] [92]. Next, the calculation of X is obtained by solving the following optimization problem:

min_X || A * X - Y ||^2_F

s.t., A * X >= 0

Once X is calculated, then we can use it to convert a new (P, S) combination (i.e., a), via the following equation:

BDU values = a * X (that is, y = a * X is the converted BDU values)

Since this result is not necessarily non-negative (as b,d,u values are probabilities) , a normalization method needed to be applied and the TruSStReMark framework uses the following normalization:

y = y / sum(y) (i.e., y normalize y by dividing y by the sum of its values)

This was chosen to normalize the dominance of any single value and get the ranges within the value ranges of b,d, and u [92]. Therefore, the results should be the normalized y representing the BDU probabilities.

Starting from the identified boundary samples, which are listed in Table 5-3, all the other sentiment values computed by the framework are converted to subjective logic tuples using the model described above.

5.1.4 Initial case-study on Trust-based Recommendation

TruSStReMark is initially used to experiment with the initial versions of the trust-based recommendation algorithms [93] (i.e., NP-Tr-Rec and P-Tr-Rec as indicated in Figure 5.4 and Figure 5.5). We have developed two different techniques. They are: 1) one technique does not take user profile information of the reviewer into the calculations (NP-Tr-Rec); and 2) Other techniques does use the reviewers' profiles for the trust-based service recommendations (P-Tr-Rec).

Figure 5.4 presents the basic structure of the first technique (i.e., algorithm which is not based on reviwers' profiles NP-Tr-Rec) and Figure 5.5 presents the basic structure of the profile-based algorithm (P-Tr-Rec). Since, it's profiled, the (P-Tr-Rec) uses the cosine similarity to calculate correlations to compare both the traditional star rating-based recommendations and the trust-based recommendations. The first method is analyzed by ordering the output of NP-Tr-Rec algorithm by ranks of the apps and the second method is analyzed using percentage error by ‘leave-one-out’ predictions of similarity matrices resulted by P-Tr-Rec algorithm.

```
NP-Tr-Rec-Algorithm
Input: Reviews of services (or Apps) (Star Ratings and Textual Narratives)
For Each SERVICE ($R_s$ as Rating, S[] as Feature Sentiments Array){
    For Each REVIEW (R as Star Rating, T as Text)(1 to N)   {
        $R_s = R_s + R$ // Add Values; $N_c = N_c + 1$ // Review Count
        For each FEATURE in S[] {
            Calculate SENTIMENT of T (Polarity, Subjectivity)
            Calculate W - TIME-SENSITIVITY of T (Range 0 to 1)
                ( ('ReviewDate'-'ServiceFirstAvailableDate') /
                 ('Today'- 'ServiceFirstAvailableDate') )
            Update S[i]($P_i$,$S_i$)  (with + or – Adjustments)
                Based on current P, S subject to weights of W and $N_c$
    Calculate Average Star Rating ($R_A = R_s / N$)
Output:  $R_A$ (Range (0 to 5)), S[] (Each Entry Range (-1 to +1))
```

Figure 5.4 Non-Profiled Trust-based Recommendation (NP-Tr-Rec) [91]

```
P-Tr-Rec -Algorithm
Input: Reviews of services (or Apps)
 (Star Ratings and Textual Narratives)

For Every REVIWER (i and j)
  Calculate Two COSINE Similarity Matrices ($SM_1$ and $SM_2$)
      Each based on the vales from
       - (Star Rating) REVIEWER-APP vectors for i and j
       - (Sentiment of App UI) REVIEWER-APP vectors for i and j
If two users are A and B :   For each common rating (r)
The COSINE Similarity = $( (\Sigma\ A_r * B_r) / (\| A \| * | |B \|) )$
  Where $\| A \| = ( \Sigma ( A_r )^2 )^{0.5}$
```

Figure 5.5 Profiled Trust-based recommendation (P-Tr-Rec) [91]

As a preliminary study, we evaluated the review data of a subset of Android marketplace based apps (i.e., most commonly known as apps) in relation to apps domain. The randomly selected dataset includes 79648 apps (which were considered as individual software apps) and 2389446 reviews. In this dataset, the most common categories of apps are Notifications (Weather and Traffic), Reservations (Flight and Rental Car), Navigation (Maps and Directions), Entertainment (Games and Music) and News (Papers and Magazines). These external textual reviews about these apps are considered for the trust-based recommendations against the traditional star rating-based recommendations (which we also calculated to validate from already available data in the Marketplace). The automatic spelling correction functionality of the TextBlob is used before parsing available reviews. Both the abovementioned (described in Figure 5.4 and Figure 5.5) approaches were evaluated by passing the dataset assuming '*the need for a weather service*' as the explicit query. The execution environment was made up of Java and Python running on a machine containing Windows 8.0 (64-bitOS), 8 GB RAM, and Intel[9] 3.10 Gz processor.

For these experiments a subset of weather forecasting apps was chosen, which includes top five weather apps (i.e., AccuWeather, WeatherBug, AndroidWeather, YahooWeather, and Weather.com). The traditional way is to rank these apps by average star rating with manual inspection and optional readings of textual narratives. Following sub-sections describe the analysis of the results. For the sake of preserving the privacy of the apps and the reviewers; we have anonymized details during the following discussion.

(1) Trust-based Recommendation (Non-profiled) without timing constraints: Table 5-4 indicates the results of executing the **NP-Tr-Rec** algorithm on apps related to the weather forecasting domain.

Table 5-4 NP-Tr-Rec results for top-5 weather apps [91]

	Tradition Ranking based on R_A(0 to 5)		**Trust-based Ranking based on Sentiment Scores of S[] between (-1 to +1)**			
Service	**Star Rating (R_AValue | Rank)**		**User Interface (S[1]Value | Rank)**		**Battery Life (S[2]Value | Rank)**	
App 1	4.3	4	+0.75	1	+0.36	2
App 2	4.4	3	+0.68	2	+0.42	1
App 3	4.4	2	+0.46	4	+0.32	3
App 4	4.5	1	+0.51	3	+0.30	4
App 5	4.2	5	+0.27	5	+0.27	5

In the Table 5-4, the apps are ranked according to both R_A and S [User Interface, Battery Life] (from Figure 5.4), which correspond to traditional and trust-based recommendations. These results are calculated by parsing reviews and each of these service has around 300,000 star ratings and 50,000 textual reviews. However, this is not surprising as everybody who rates an service does not publish comments and we observed that the ratio of comments/ratings is around 1/5.

The ranks obtained via the star rating and via our algorithm are different for each Service except for the Service5. The low ranking of the Service5, correctly computed by both the rating schemes, is attributed to the presence of many negative reviews. The Sentiment Scores (i.e., output values S[]) for all the apps are positive, which indicates that a majority of the reviews are positive about the UI and battery life of these apps. If we consider only the average star ratings of all these five apps, as a product-based recommender would do, we find that there is hardly any difference (of 0.3 where 4.5 being the highest rating and 4.2 being the lowest rating) between these five apps. A similar pattern is observed with other categories of apps which clearly highlight the fact that an average star rating does not accurately reflect the trust about an service. Thus, require a better recommendation technique. **NP-Tr-Rec** address this deficiency by providing a rank based on various features of Apps as reflected in the narratives of the reviewers. Considering the S[] values in the Table 5-4, it is clear that Service1 and Service2 are ranked higher when compared with other apps. However, these two apps are ranked relatively lower according to the average star rating-based rank. Therefore, our approach (**NP-Tr-Rec**) provides a better ranking that will help the users while selecting apps to their preferences from available choices in a marketplace.

(2) Trust-based Recommendation (Non-profiled) with timing constraints: Results presented in Table 5-5 are obtained by enforcing a pre-defined and arbitrarily selected cutoff time (60 sec) in the NP-Tr-Rec technique while processing the reviews. The cutoff time selects only the recent reviews and hence, indicates how the new users react to the apps. Both the star ratings and the S [] values are affected by this time limit, but we were not able to draw significant insights by comparing these results with the results shown in Table 5-4. We concluded that further investigations are needed to evaluate and identify an appropriate time interval that can be used to limit such computations.

Table 5-5 Results from Table 3 with timing constraints [91]

Service	**Tradition Ranking (R_A)**	**Trust-based Ranking (S[])**	
	Star Rating	**User Interface**	**Battery Life**
App 1	4.6	+0.52	+0.64
App 2	4.1	+0.35	+0.22
App 3	4.8	+0.28	+0.40
App 4	4.5	+0.74	+0.16
App 5	4.0	-0.16	-0.24

Although the marketplace encourages the reviewers to show their identity, which is linked to their email accounts, some reviewers choose not to reveal their identity. However, as we observed, the percentage of number of anonymous reviews is less than 10% of the reviews. This leads to an additional investigation wherein the ratings of the reviewers are also considered as a parameter while ranking apps. From the selected review dataset of the weather domain, we identified the top 1000 reviewers by counting the number of different reviews published by them. The ranking analysis was then limited to the reviews from these reviewers. A highly sparse matrix (as a data table) is populated with the identified reviewers on one side and the apps that they have reviewed on the other side. Each data point corresponding to a cell in this sparse matrix contains information about reviewer's star rating and corresponding trust scores (i.e., sentiments of the UI feature) calculated from their textual reviews. These calculations are based on the approach described by Figure 5.5. As a preliminary effort, in this experiment, we considered only the sentiments about the UI of the service. Due to the privacy obligations, again, the identities of the reviewers are masked in the Table 5-6.

Table 5-6 Percentage error calculations of P-Tr-Rec results [91]

Reviewer (Service)	**Traditional Star Rating**			**Trust-based Sentiment of UI**		
	Prediced	**Actual**	**%E**	**Predicted**	**Actual**	**%E**
R1 (App 1)	3.91	4.00	3%	+0.76	+0.82	7%
R2 (App 2)	4.86	5.00	2%	+0.66	+0.71	7%
R3 (App 5)	2.10	2.50	35%	-0.58	-0.46	30%

Table 5-6 indicates the partial results of executing the **P-Tr-Rec** approach on the selected dataset. We indicate only 3 reviewers and 3 apps in the above table. In this technique, we compute the cosine similarity measure between the reviewers and use that in the ranking process. To assess the validity of the technique, we performed leave-one-out predictions – i.e., selecting a top reviewer who has review about a particular service already (that is the actual value in Table 5-6) and predict the values of star rating and UI sentiment that this reviewer may create based on the correlation matrices with other similar reviewers.

To predict the values that the “left out reviewer” may assign to an service, we use the ratings from the top-20 most correlated reviewers who have rated the same service. Once the prediction for the “left-out” reviewer is computed we then compare, by calculating the percentage error (%E) the prediction with the actual rating given by that reviewer. As indicated in the Table 5-6, we are able to predict, fairly accurately, for the first two reviewers. However, in the case of the last reviewer, our predictions deviate a lot from the actual ratings. Further manual analysis of this case revealed that, when the sentiments are negative, the percentage error values are high compared to positive sentiments. We think a possible reason for this is the typical nature of the reviews for top ranked apps (i.e., the small size of the available negative reviews (~40000) compared to the positive reviews (~300000)). Again, this suggests further extensive investigations are needed to consider this particular technique.

The experience gained from this case study gave us the further empirical validation of the TruSStReMark framework and it provided means for next steps of improvements.

5.2 TruSStReMark for Selection and Recommendation

This subsection contains a discussion of our previous work presented in [92]. As indicated earlier, trust is a time dependent feature and hence, its variation over time needs to be considered in the selection process. However, most prevalent service S&R schemes do not include a trust establishment based on evaluations between the service vendors and consumers. Table 5-7 includes two phases where trust establishments happen between apps and consumers in marketplaces. These service trust evaluation schemes operate only based on quality of service agreements such as, static service level agreements (SLAs). Online apps are affected by the change of the environment conditions, which result in the fluctuations of the quality of service offered.

Table 5-7 Trust establishment phsases of online apps and cosumers

Active Phase	Initiates the trust establishments. (Optional agreements on future fluctuations in trust establishments)
Passive Phase	Periodic communications after the initiation of trust establishments. Updates trust establishments by consumers.

Therefore, we propose a trust evaluation based on the trust between vendors and consumers at the time of service S&R. These establishments are based on the aggregated views of the trust of the apps and their future rules of engagements to address the fluctuations. For example, trust aspects of the evaluation should include information about which evidences to consider in quantifying the effects of a service outage. We have investigated on how the existing algorithms can be modified based on two phases namely; Active Phase and Passive Phase (as described in Table 5-7). An active phase of trust evaluation is initiated at the beginning of the service selection. This evaluation is based on the current views of the trust of that service which establishes a mutual agreement for current and future variations of provided QoS. After the selection (or the recommendation) process is completed, as new evidences are available (such as new feature updates and releases), the passive phase periodically monitors this trust agreement. If any violations (i.e., QoS violations or service outages) are noted then the phase switches back to the active and should initiate the re-establishment of trust between entities.

The main feature of the algorithms that perform service S&R is that they match available service specifications (consist of different contracts) with user requirements. The TruSStReMark framework considers trust attributes of the available apps targeting their

QoS features. Also, service selection is an expensive and time consuming process, especially when the search space is very large. As mentioned in the previous chapter, proposed service S&R algorithms are divided in to two phases; one is the off-line phase and other is the on-line phase (as described in Figure 5.2). Once the service specifications including additionally calculated trust contracts are available for various apps, these contracts can be used to improve the S&R process. To perform service S&R, additionally trust requirements of a query are matched against available service.

5.2.1 Trust-based Service Selection

The main two parts of service selection are to perform a search operation to select a candidate set and to rank them according to the requirements. The most common approach for search and rank is called content-based filtering (CBF). It uses the product description and, optionally a profile of the user’s preference. These details are expressed using keywords or other types of information such as previous items that the user had liked in past, and the user’s location. If such personalized history is not available, then a general popularity of items (based on other users’ behavior) is used in cold start situations. A set of candidate items are selected and scored based on a criteria (e.g., the number of keywords applicable) to rank the final results. A widely used algorithm to identify a set of keywords that reflects the summary of given textual corpus is the TF-IDF [86] (Term Frequency and Inverse Document Frequency) representation. The CBF algorithm uses a content-based profile of items and users based on a weighted vector (e.g., TF-IDF scores of each keyword) of item features. The weights denote the importance of each feature to the user or for the item in general.

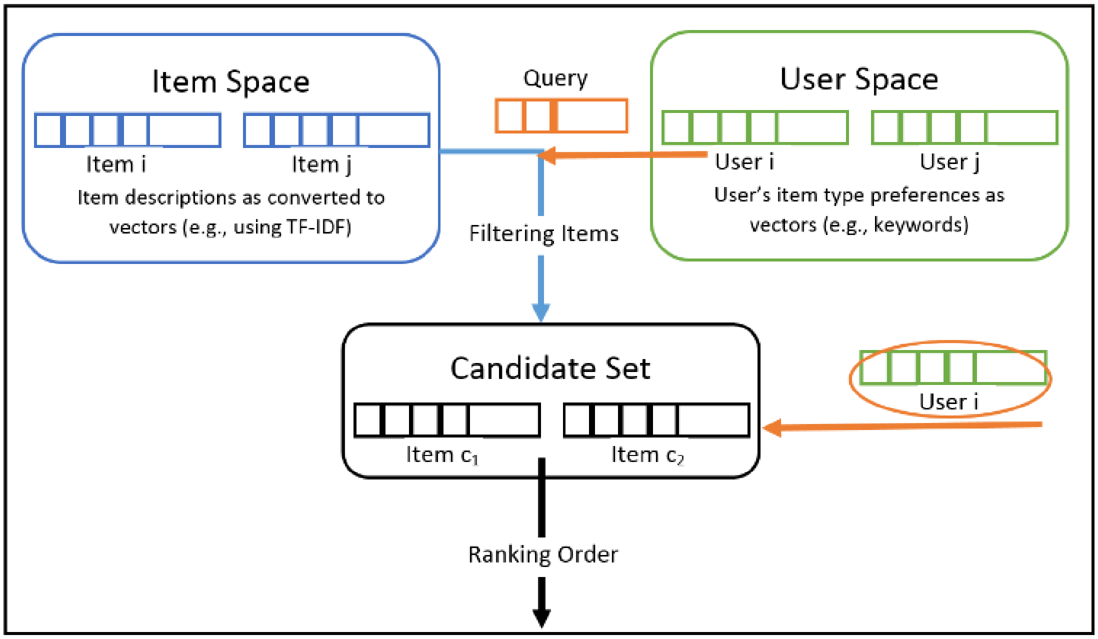

Item Space
Item i
Item j
Item descriptions as converted to vectors (e.g., using TF-IDF)
Query
User Space
User i
User j
User's item type preferences as vectors (e.g., keywords)
Filtering Items
Candidate Set
Item c$_1$
Item c$_2$
User i
Ranking Order

Figure 5.6 Content-based filtering approach (CBF-Rank) [92]

Given a set of keywords as a user query, the algorithm initially filters the items based on the keywords (i.e., types of item) and then ranks items in their importance according to user preference. Figure 5.6 summarizes this approach. A key issue with this approach is how effectively the system can apply a user's feedback to other items for future predictions. The popularity of the CBF algorithm is mainly due to its simplicity and low resource consumption. Today, given the availability of large computing power with resources, an algorithm such as CBF with limited content analysis and limited degree of flexibility for updates should be re-evaluated. Given a dataset from a marketplace which includes service descriptions, user-service information, and set of reviews by the users, TruSStReMark performs improved content-based search and ranking algorithm as follows: for each service, the basic content-based algorithm calculates a vector (of size 'k' keywords) by parsing the service descriptions using a NLP technique such as the TF-IDF. This result is given in a sorted order based on the score produced by the TF-IDF algorithm.

```
Tr-CBF-Rank-Algorithm

Part (1)
Input: R Reviews of services and D Service Description
For Each SERVICE (S[] as Service Vector)
   For All REVIEWS (T as Combined Text Document of R,D)
        Calculate TF-IDF and select top k
Output:  S[k] Service Vector (Sorted from TF-IDF score)

Part (2)
Input: R Reviews of Users and Q Query information
For Each USER (U[] as User Vector)
   For All REVIEWS (T as Combined Text Document of R,Q)
        Calculate TF-IDF and select top k
Output:  U[k] User Vector (Sorted from TF-IDF score)

Part (3)
Input: R Reviews of services
For Each SERVICE (S[] as QoS <B,D,U> tuples){
   For Each REVIEW (T as Text)(1 to N)   {
     For each QoS in S[] {
       Calculate SENTIMENT of T (Polarity, Subjectivity)
        Convert (P,S) to <B,D,U>
       Calculate W - TIME-SENSITIVITY of T (Range 0 to 1)
          ( ('ReviewDate'-'ServiceFirstAvailableDate') /
           ('Today'- 'ServiceFirstAvailableDate') )
       Update S[i] <B,D,U> (using Consensus Operator)
          Based on current <B,D,U> subject to weights of W
Output:  Updated S[]

Part (4)
Input: R Reviews of services
For Each USER (U[] as QoS <B,D,U> tuples){
   For Each REVIEW (T as Text)(1 to N)   {
     For each QoS in U[] {
       Calculate SENTIMENT of T (Polarity, Subjectivity)
        Convert (P,S) to <B,D,U>
       Calculate W - TIME-SENSITIVITY of T (Range 0 to 1)
          ( ('ReviewDate'-'ServiceFirstAvailableDate') /
           ('Today'- 'ServiceFirstAvailableDate') )
       Update U[i] <B,D,U> (using Conjunction Operator)
          Based on current <B,D,U> subject to weights of W
Output:  Updated U[]
```

Figure 5.7 Outline of Trust-based search and ranking (Tr-CBF-Rank) [92]

Figure 5.6 indicates the places where TruSStReMark is augmenting the process of content-based search and ranking algorithm and Figure 5.7 indicates the associated pseudo code. Service descriptions may not include all interesting feature keywords for the users and TruSStReMark updates this vector by dynamically parsing the reviews for a particular service from all the users who have included a review (Part 1 in Figure 5.7).

Similarly, TruSStReMark uses these reviews to update the vector of user preferences (Part 2 in Figure 5.7). The execution frequency of Parts 1 and 2 can be decided based on given criteria, such as, the predefined number of new reviews or the time between updating of apps. From these service vectors and user vectors (each with keywords and their TF-IDF scores), an important subset of QoS parameters for each service can be identified based on a knowledge base (KB).

The domain experts manually create these KBs (i.e., both machine and human readable of <target domain, set of keywords> tuples) using their domain knowledge about a specific domain (e.g., the related entries for a weather related service domain of this KB may include: forecast, battery life, user interface, accuracy, cost, reliability etc.).As the next step, TruSStReMark keeps an additional vector of <B,D,U> tuples for both of these corresponding apps and user preferred QoS lists. These tuple values get periodically updated by Part 3 and Part 4 of the algorithm (Figure 5.7). The framework collects the evidences about the QoS attributes from reviews. The timeframe of the aggregation is more frequent than Part 1, which can be hourly or daily depending on the available resources and capacity of the infrastructure which the algorithm executes on.

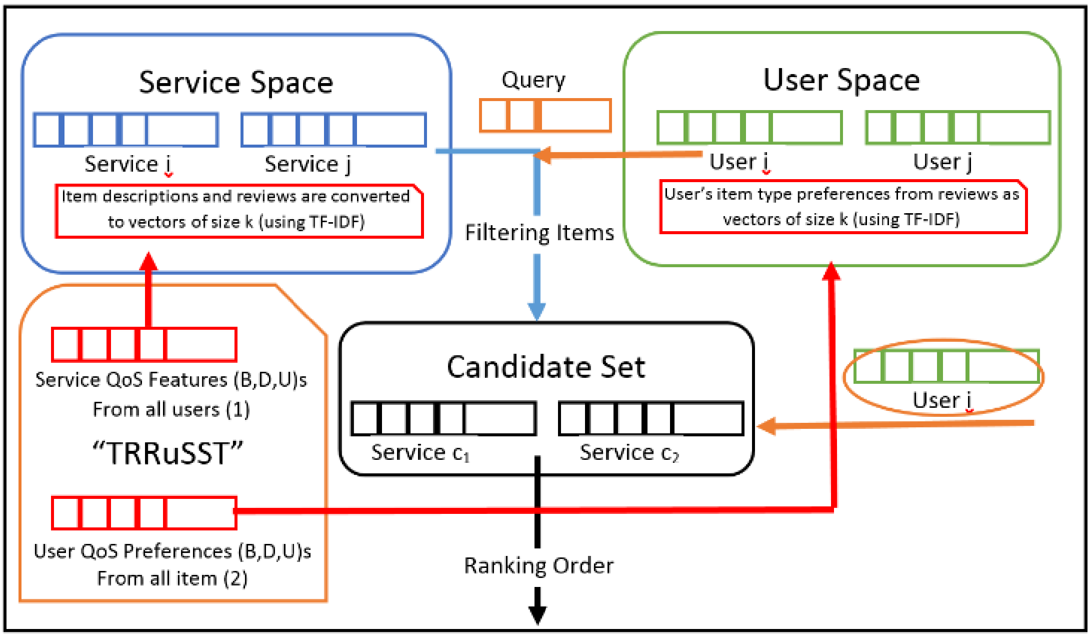

Service Space
Service i
Service j
Item descriptions and reviews are converted to vectors of size k (using TF-IDF)
Query
Filtering Items
User Space
User i
User j
User's item type preferences from reviews as vectors of size k (using TF-IDF)
Service QoS Features (B,D,U)s From all users (1)
"TRRuSST"
User QoS Preferences (B,D,U)s From all item (2)
Candidate Set
Service c1
Service c2
User i
Ranking Order

Figure 5.8 Trust-based search and ranking (Tr-CBF-Rank) [92]

The service vector <B,D,U> tuples are updated based on review sentiments of the QoS keyword present in the review, which is based on the calculated sentiments (i.e., based on SA to SL conversion presented in Subsection 5.1.3). These service vector tuples are updated using the consensus operator of subjective logic, as it is used to combine many independent user opinions about the same proposition (i.e., QoS) into a single opinion. Also, this adjustment is biased with a weight of the freshness of the review as mentioned in Figure 5.7.

Similarly, the user vector <B,D,U> tuples are updated based on review sentiments and the conjunction operator of subjective logic is necessary here, since it is used to aggregate two opinions of the same QoS expressed for different apps by the same user to reflect the conjunctive truth of both. The main goal of TruSStReMark is to record fluctuations associated with the QoS to predict the behavior of the service over an identified set of QoS values. We chose to represent and track each QoS value of a software service using these Subjective Logic based <B,D,U> tuples from the evidences gathered from user reviews.

This approach works without parsing the reviews many times. It can be configured to include different vector sizes of apps and user vectors as well as the number of QoS additionally tracked based on the domain of the service.

All the calculations based on trust are carried out during an off-line phase of the service search and rank algorithm, however, the querying happens in real-time. When a service search query (with QoS values) is presented to TruSStReMark, such as “request: weather forecasting service, QoSs: large text and good battery life”. the initial filtering to generate a set of candidate apps is done by looking at service vector scores (weather and forecasting) for matching keywords and at the same time comparing the <B,D,U> values of service QoS features (battery life and large text) which are inferred by the query. Two <B,D,U> values are compared by giving precedence to B first, U next, and D last, such that with respect to their QoSs beliefs; highly believable apps appear first, then highly uncertain (i.e., neither B or D is prominent) apps appear in the middle and highly disbelievable apps appear last (referring to the subjective logic-based ordering operator from Table 4-3).

Next, the experiments are carried out to show the improvement in quality of the results of service selection when TruSStReMark based selection algorithms are used. Those results are then compared with the prevalent CBF approach. The details about these experiments and analysis are presented in Subsection 5.3.

5.2.2 Trust-based Service Recommendation

As mentioned earlier, the inherent features associated with the software domain such as frequent updates, execution environment changes, and associated dynamic QoS concerns, make it more challenging to apply these existing product-based recommendation techniques to software marketplaces without any modifications. The following section presents the work on the TruSStReMark framework to augment and enhance the prevalent collaborative filtering based approaches by using the quantification of trust when searching, ranking, and recommending software apps.

Collaborative filtering (CLF) is based on the assumption that item preferences (item-item) or user preferences (user-user) which are similar in the past will be similar also in the future. These systems compare collected data (i.e., user-item relationship matrix which contains either binary or integer values, and these values represent that the user purchased this item or shows rating given by the user) to calculate a list of candidate items. This is a sparse matrix and each row or column vectors are compared to calculate their similarity (e.g., using Cosine or Pearson correlation) with each other, in order to calculate user-user or item-item similarity matrix (for each entity, the results are prune to include top-k similar entities).

For an item-type or a user preference (depending on its past p entries), a candidate set of items (of size p*k) is generated. Finally, a ranking score is calculated for each candidate as the sum of cosine similarities of that item with a user's past p entries to generate a sorted list of recommendations. Figure 5.9 summarizes this approach. A key issue with such collaborative filtering approaches is that their results often suffer from stability (i.e., takes more time to update already stable user preferences or item groups) and hence, there could be trust-based concerns about the results, especially when considering apps in an online marketplace. Frequent events in a marketplace related to users and apps make the basic ranking and recommendation techniques fall short if they do not include the effect of these events using trust based augmentations to their basic recommendation scores.

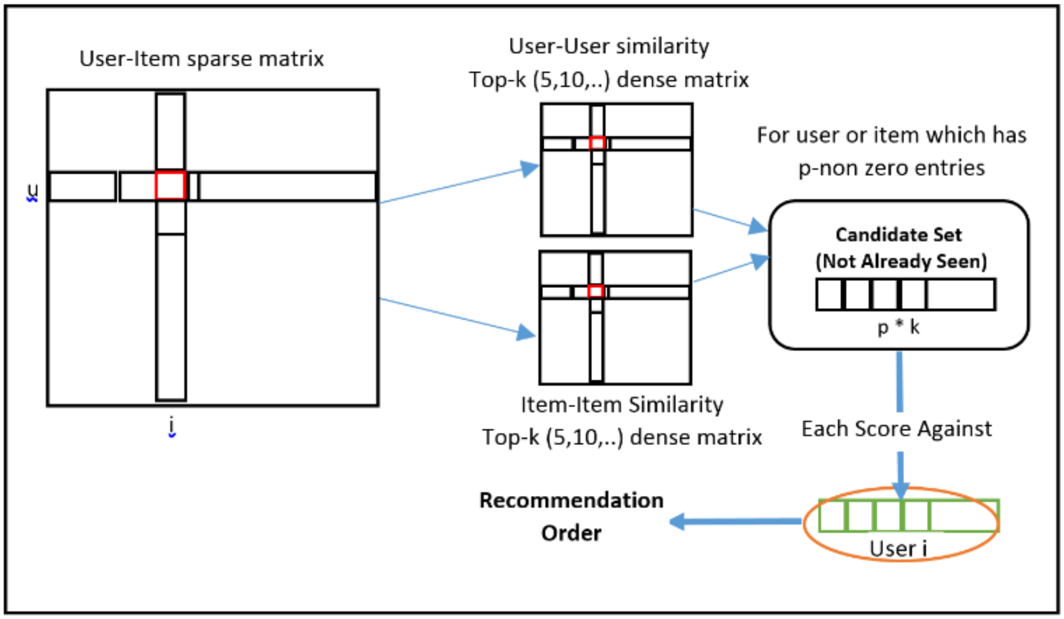


Figure 5.9 Collaborative filtering approach (CLF-Rec) [92]

The prevalent product-based recommendation systems usually do not consider the narrative portion of reviews. TruSStReMark uses these reviews about different QoS features of the service, which contain important information about trust (e.g., subjective opinion and confidence). It uses these text reviews for trust-based calculations as a part of the proposed service recommendation techniques. These techniques are described in Figure 5.9 and Figure 5.10 as compared to the prevalent collaborative filtering recommendations.

We follow the approach described in Figure 5.11 to generate data for preprocessing. The algorithm now has identified the service and user QoS lists which are then stored and their <B,D,U> values periodically updated to keep the system up-to-date. As mentioned in the previous subsection, the TruSStReMark framework isolates feature-based sentences (e.g., corresponding to UI and battery life) from these reviews and then calculate associated sentiments to convert them to <B,D,U> tuples and aggregate them using subjective logic based operators. The main updates happen when the candidate set is selected based on the collaborative filtering approach as indicated in Figure 5.10 and Figure 5.11.

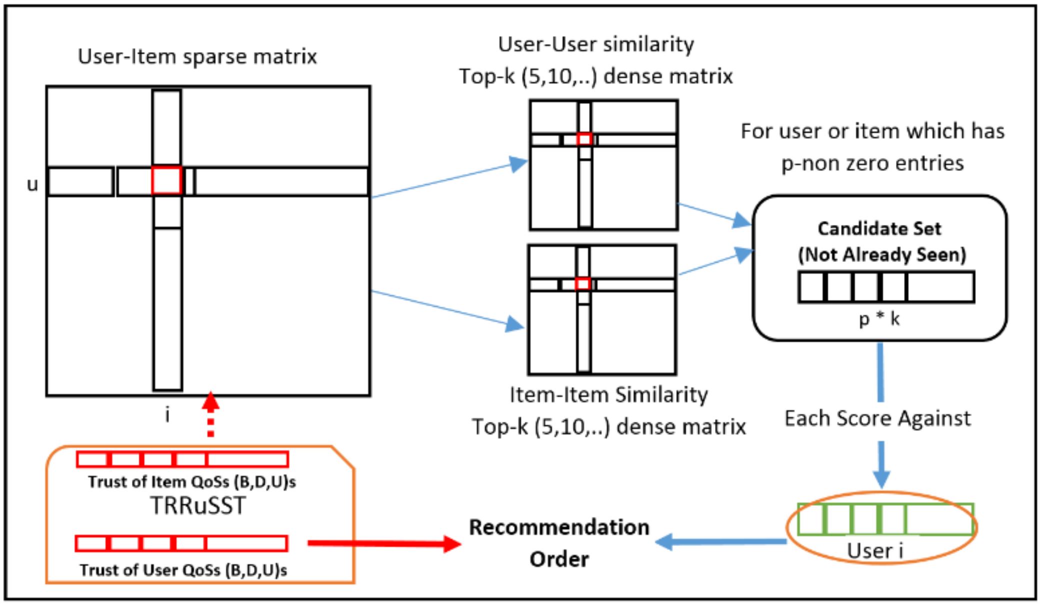
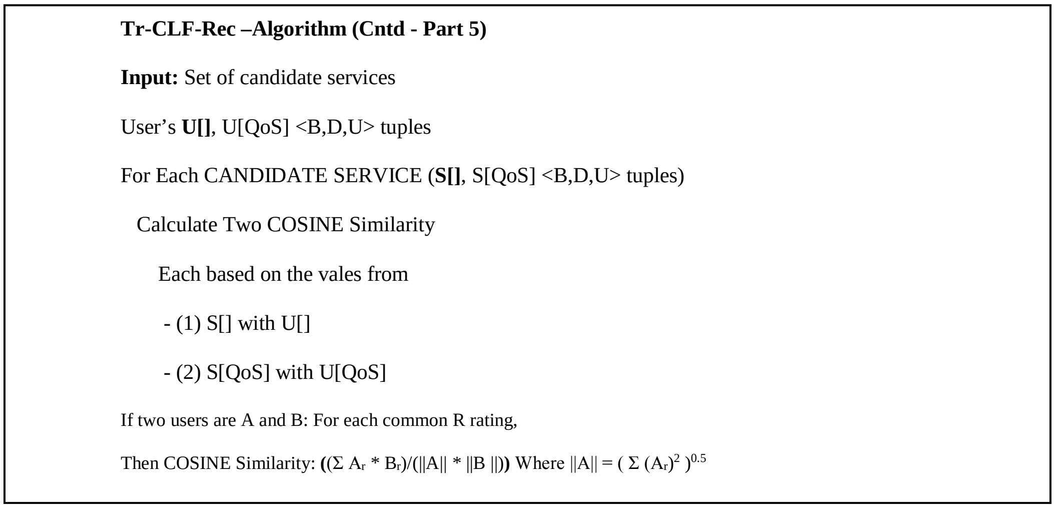


Figure 5.10 Trust-based Recommendation (Tr-CLF-Rec) [92]



```
Tr-CLF-Rec –Algorithm (Cntd - Part 5)
Input: Set of candidate services
User's U[], U[QoS] <B,D,U> tuples
For Each CANDIDATE SERVICE (S[], S[QoS] <B,D,U> tuples)
  Calculate Two COSINE Similarity
     Each based on the vales from
      - (1) S[] with U[]
      - (2) S[QoS] with U[QoS]
If two users are A and B: For each common R rating,
Then COSINE Similarity: $((\Sigma A_r * B_r)/(||A|| * ||B ||))$ Where $||A|| = ( \Sigma (A_r)^2 )^{0.5}$
```

Figure 5.11 Outline of Trust-based Recommendation (Tr-CLF-Rec) [92]

For each candidate service, we calculate the cosine similarity using the B,D,U tuples to generate the updated and sorted list of recommendations. Next, the experiments are carried out to show the improvement in quality of the results of service recommendation when TruSStReMark based algorithms are applied. Those results are compared with the prevalent CLF approach and the experiments and results are presented in Subsection 5.3.

Until now, this dissertation has defined the trust with respect to a software service, proposed a trust model for apps' lifecycle, and proposed a suitable trust contract for apps needed to perform trust-based service operations. It also focused on the long term monitoring aspects of trust contract and the benefits associated to service selection and recommendation. The focus of the following discussion is to present the required algorithmic modifications in service selection and recommendations to get the benefits associated with long term monitoring of trust contracts and to experiment with the simulated TruSStReMark framework using datasets from Android and Amazon marketplaces.

As mentioned earlier, TruSStReMark defines the trust of a service in a marketplace, quantifies the trust by monitoring and aggregating various external evidences, represents the trust in the service specification and improves the service selection and recommendation algorithms. The simulation environment of TruSStReMark was developed by using JAVA (as the main programming language), Python (as the secondary programming language), TextBlob (as the library for NLP processing textual data), Apache Darby (as the relational meta-database engine), and Apache Tomcat (as the JSP and Servlet Container for basic front end views). The parallelization of algorithms was done by porting the initial version of the framework to the Hadoop and Spark ecosystems. This environment is developed by mainly using Apache HDFS (as the storage for evidences in text format), Apache Hadoop (as the batch processing framework), Apache Hive (as the data warehouse and storage for caching tables), and Apache Spark (as the streaming processing framework).

This prototype was experimented with service datasets from the Android and Amazon online marketplaces. Publically available software apps from the Android and Amazon marketplaces are used as datasets to empirically evaluate this framework. Next subsection describe the experimentations of TruSStReMark andanalyzest the results [90], [89].

5.3 Analysis of Trust-based Selection and Recommendation Algorithms

This section provides details of experimentation and analysis of results of the improved algorithm of trust-based selection (i.e., search and ranking algorithms of TruSStReMark), as compared to the prevalent content-based filtering. There is an additional computational cost spent on parsing the reviews and which is justified by the quality improvements of the results. We present an empirical study that focuses only on the external views (i.e., reviews available in the marketplace) of apps to provide a trust-based search, rank and recommendation for a set of queries. To evaluate the approach, we use the Android marketplace Review dataset [11] (2389446 reviews of 70000 mobile apps and apps). To achieve fair comparisons, the Android marketplace dataset was pruned to only include the weather domain, which was then comparable (around 90766 reviews and 3895 apps) to the Amazon Dataset (which we use in Section 4.2). The mean number of words for both of these datasets is more than 30 words per review. Hence, for the replaceability study, we have used the Android marketplace dataset and for the compatibility study we have used the Amazon software marketplace dataset [16] as described in Section 4.2.

Compared to the prevalent approach of content-based filtering, this candidate set is optimized for the query and reflects the narratives about the certain QoS attributes expressed by the users. Also, this approach captures the fluctuations of QoS in certain time frames which is an advantage compared to the prevalent fixed approach. Finally, the candidate set is ranked first according to user preference score value against the service vector scores and then readjusted with service QoS beliefs against user QoS beliefs based on the criteria mentioned above (if there exist common QoSs in both the user and the service intersection set of QoS vectors). For example, during our experiments we have used 3,5,10,15 as our user and feature vector sizes and 3 and 5 as user and service QoS additional vector sizes. Example keywords for the Android marketplace dataset of weather apps show weather, forecast, 5 day, 10 day for service and user feature vector and Battery, UI, accuracy as service and user QoS vectors.

Using the Android marketplace dataset, the collaborative filtering application is not possible, because of the dataset limitations. For example, because the majority of reviews are anonymous, the user related data could not be extracted. Available software from the Amazon dataset and available apps from the marketplace are assumed to behave as apps within our prototypical (TruSStReMark) environment. The goals of the experiments are to compare the TruSStReMark based approaches for ranking with the prevalent approach (Tr-CBF-Rank against prevalent content-based filtering). The automatic spelling correction functionality of the TextBlob Library [84] is used before parsing available reviews. Randomly ~10% of the data is selected as the test dataset and reset is selected as the training dataset. This was performed by picking 10% of the users from the dataset to remove a service from each of the selected users (who wrote a positive review about an identified QoS in their review) and adding those users' apps and corresponding reviews to the test set. We then parsed the dataset through both the prevalent and TruSStReMark based algorithms to generate top-N search and recommendation result sets for each user. If the test service is found in the set of search or recommended apps of the relevant user, it is considered as a Hit. Hit Rate (HR) and Average Reciprocal Hit-Rank (ARHR) are indications of how well the algorithm performed considering the quality and relative position of the results. This approach was evaluated by parsing the dataset together with the algorithm execution to measure the performance. In these experiments, the TruSStReMark execution environment was made up of Java and Python running on an environment containing Amazon EC2 (Amazon Elastic Computing Cloud) free tier Linux t2.micro instance with 64-bit platform support.

Definitions of HR and ARHR measurements: HR is a measure of the number of the test apps that the algorithm included in the results of selection and recommendation. If N is the total apps of the test set, then HR = (Number of Hits)/N. Compared to HR, the ARHR is capable of evaluating the relative importance of the position in the results of selection and recommendation. Hits which are present earlier are weighted higher than the Hits that are present later in the result ordering. If Pi is the position of the service of the size N result set then, ARHR = (1\N) SUM (1\Pi). The experimentation results include comparisons of the result-sets in terms of HR and ARHR metrics for each of the dataset. The following subsection explains the analysis of the results with performance comparisons.

5.3.1 Results Analysis of Trust-base Selection

Using the Android marketplace dataset, the collaborative filtering application is not possible, because of the dataset limitations. For example, because the majority of reviews are anonymous, the user related data could not be extracted. Available software from the

Amazon dataset and available apps from the marketplace are assumed to behave as apps within our prototypical (TruSStReMark) environment.

Table 5-8 Improvements of HR and ARHR on Android Marketplace [92]

Technique / Result Set Size	Top 3	Top 5	Top 10	Top 15
CBF (HR)	11.67	21.34	22.6	24.8
Tr-CBF (HR)	12.22	25.69	30.81	32.62
CBF (ARHR)	5.3	9.92	10.76	12.4
Tr-CBF (ARHR)	6.78	14.27	17.11	17.16

The goals of the experiments are to compare the TruSStReMark based approaches for ranking with the prevalent approach (Tr-CBF-Rank against prevalent content-based filtering). The automatic spelling correction functionality of the TextBlob Library [84] is used before parsing available reviews. Randomly ~10% of the data is selected as the test dataset and reset is selected as the training dataset.

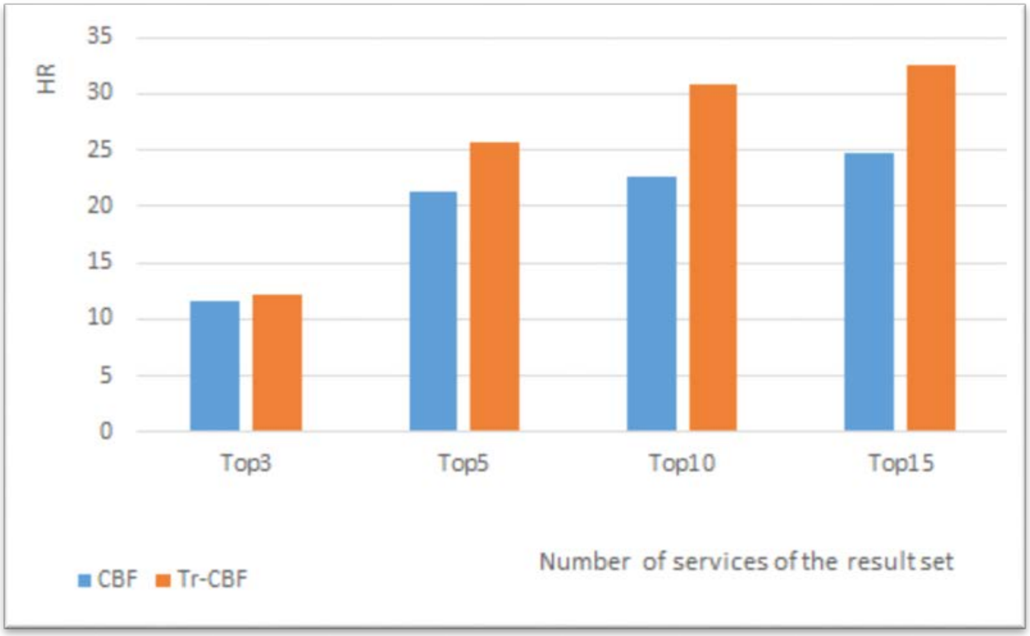


Figure 5.12 HR comparison of Android Marketplace dataset [92]

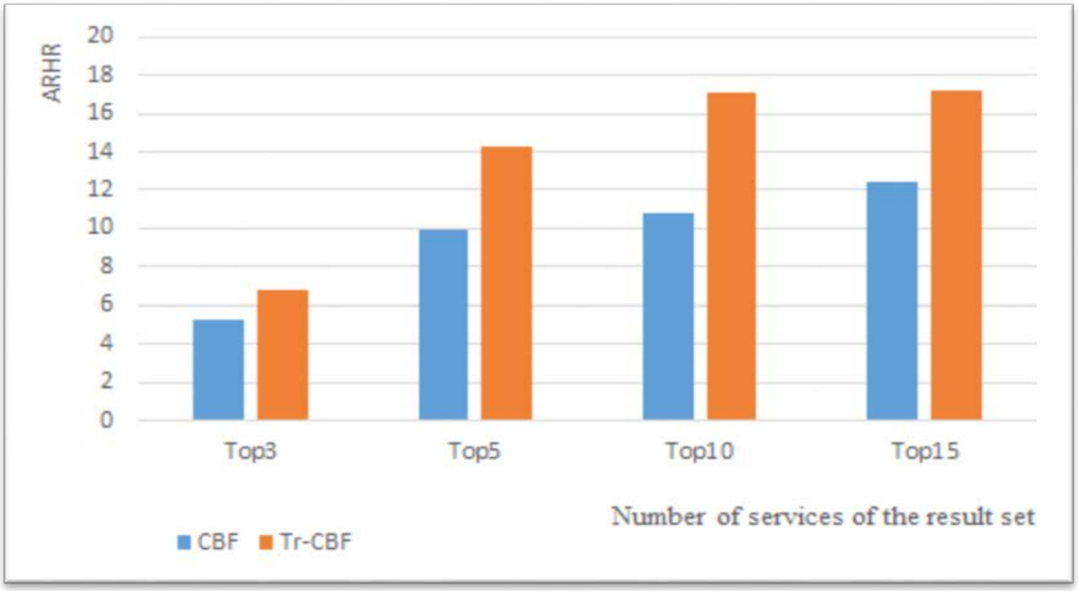


Figure 5.13 ARHR comparison of Android Marketplace dataset [92]

This approach was evaluated by parsing the dataset together with the algorithm execution to measure the performance. The TruSStReMark execution environment was made up of Java and Python running on an environment containing Amazon EC2 (Amazon Elastic Computing Cloud) free tier Linux t2.micro instance with 64-bit platform support.

Table 5-8 indicates the percentage improvements obtained by the Tr-CBF-Rank as compared to the content-based filtering for the Android marketplace-based dataset. It is evident from the Figure 5.12 that the proposed trust-based search and ranking algorithm outperforms the prevalent content-based filtering approach in terms of HR. When comparing the two approaches, the percentage increase of HR is greater in our approach due to association of the QoS of apps. The increase of ARHR in Figure 5.13 shows that the proposed trust-based search and ranking algorithm can also provide better ranking of results as compared to the prevalent content-based filtering approach.

5.3.2 Results Analysis of Trust-based Recommendations

TruSStReMark uses these text reviews for trust-based calculations as a part of the proposed service recommendation technique as compared to the prevalent collaborative filtering recommendations. We follow a similar approach as indicated in the previous section to generate data for preprocessing. The algorithm has now identified the service and user QoSs lists which are then stored and their <B,D,U> values periodically updated to keep the system up-to-date. As mentioned above, in the TruSStReMark (in Section 4.1), we isolate feature-based sentences (e.g., corresponding to UI and battery life) from these reviews and then calculate associated sentiments to convert them to <B,D,U> tuples and aggregate them using subjective logic based operators.

Table 5-9 Improvements of HR and ARHR on Amazon Marketplace [92]

Technique / Result Set Size	**Top 3**	**Top 5**	**Top 10**	**Top 15**
CLF (HR)	19.17	21.8	24.26	26.72
Tr-CLF	21.08	23.82	26.65	29.18
CLF (ARHR)	9.58	10.38	11.55	13.03
Tr-CLF (ARHR)	10.64	12.53	13.66	15.36

The main updates happen when the candidate set is selected based on the collaborative filtering approach Basically, for each candidate service we calculate the cosine similarity additionally using the B,D,U tuples to generate the updated and sorted list of recommendations.

Similar to previous subsection, we randomly picked approximately 10% of the users from the dataset to remove one service from each of the selected users. Then we pass each of the dataset though both prevalent and TruSStReMark based algorithms to generate top-N search and recommendation result sets for each user. If the test service is found in the set of search or recommended apps of the relevant user, it is considered as a Hit. Hit Rate (HR) and Average Reciprocal Hit-Rank (ARHR) are indications of how well the algorithm performed considering the quality and relative position of the results (Subsection 4.1.2).

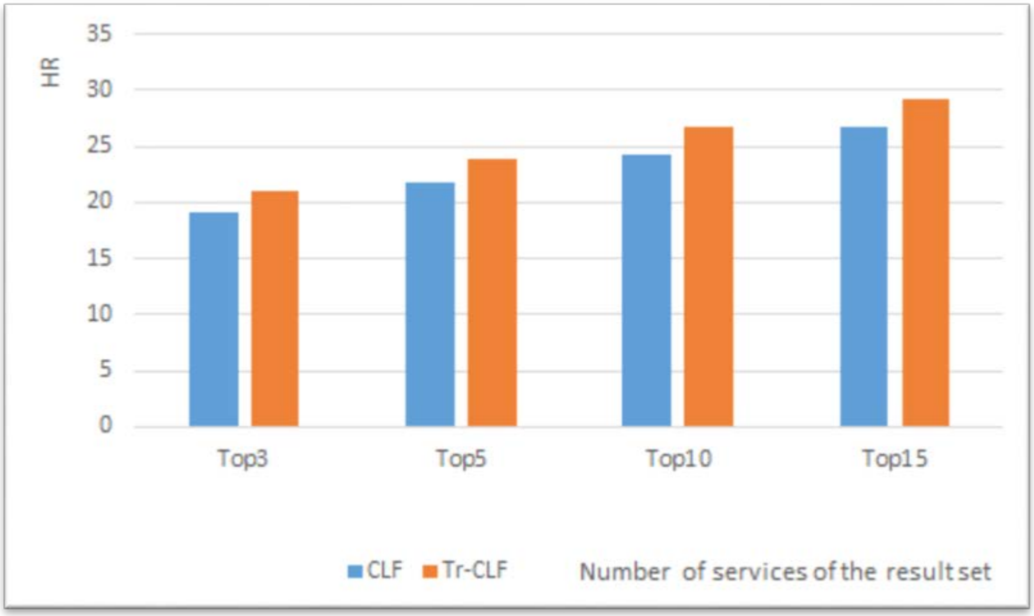


Figure 5.14 HR comparison of Amazon Marketplace dataset [92]

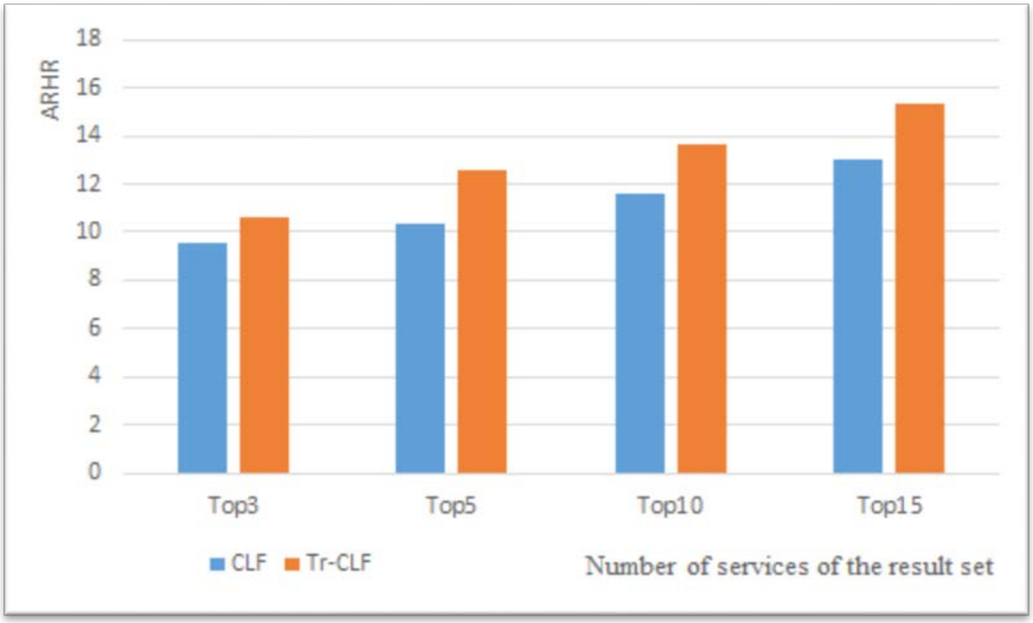


Figure 5.15 ARHR comparison of Amazon Marketplace dataset [92]

Table 5-9 indicates the percentage improvements of the Tr-CLF-Rec when compared with the traditional collaborative filtering recommendations using the Amazon marketplace dataset. It is evident from Figure 5.14 that the proposed trust-based recommendation algorithm outperforms the prevalent collaborative filtering approach in terms of HR.

The percentage increase in HR is lower in this approach (Tr-CLF-Rec) when compared to the Tr-CBF-Rank. However, the Tr-CLF-Rec could be improved and that is one of our future directions. Similar to the Tr-CBF, the increase of ARHR in Figure 5.15shows that the proposed trust-based recommendation algorithm can also provide a better relative position of recommendations in its results as compared to the prevalent collaborative filtering approach. In summary, by analyzing the results, the percentage improvement of ARHR is close to half than that of HR improvements. The percentage improvement of ARHR is higher than HR improvement in Tr-CLF-Rec when compared to CLF. This shows the importance of QoS consideration in our trust-based approach.

When this experiments were performed, in our simulated marketplace environment, the performance became an issue when new data was generated by our marketplace crawler. We compared the runtime variation of the algorithm with the increasing number of reviews in the datasets. As the number of reviews increases, both the number of users and number of apps also increases. From the comparisons in Figure 5.16, it is clear that both prevalent and trust-based algorithms take more time to perform their operations. However the trust-based selection and recommendation algorithms increase at a much larger rate, due to the sentiment analysis and the subjective logic-based calculations of the selected set of QoS-related sentences. We think that this increase in time can be reduced significantly by parallelizing and running the algorithms. The natural choice was the MapReduce based Hadoop ecosystem and experimenting it on Amazon EMR (Elastic MapReduce) cluster.

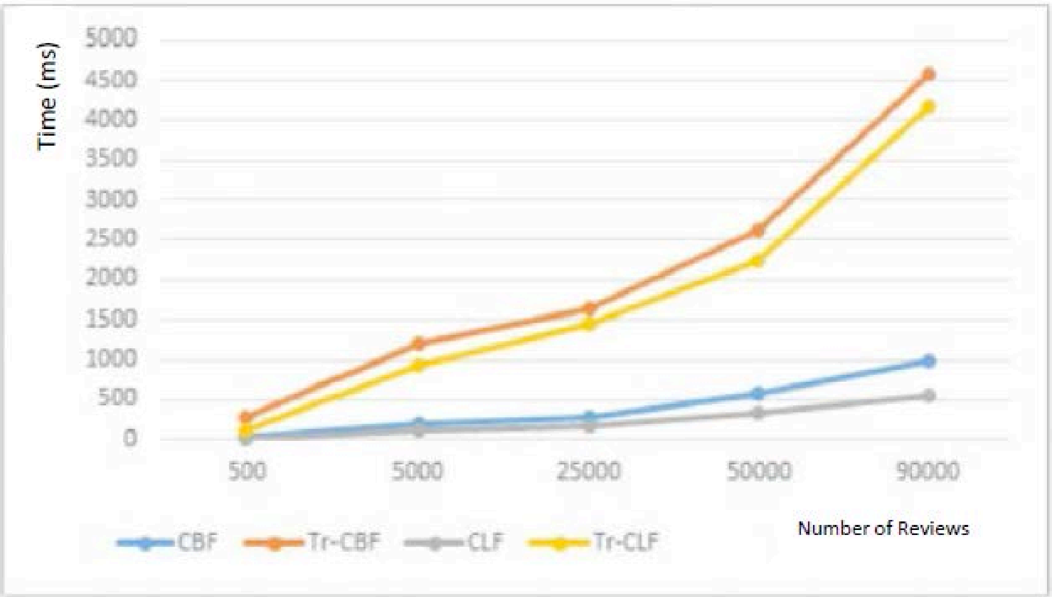


Figure 5.16 Aggregated average runtimes of the non-parallel algorithms [92]

Hence, the TruSStReMark framework parallelizes the online and offline parts of this algorithm. When a new set of evidence arrives (i.e., as in terms of new reviews which explains different service QoS values), the TruSStReMark framework calculates the trustworthiness of that current QoS values using the above approach. The details of how the framework uses the Hadoop Ecosystem to process these new evidences in parallel and how the framework uses the Spark Ecosystem to ingest these new data as a streaming solution to improve the performance of the system is discussed in next subsections and experimental results and analyses are presented in Chapter 6.

CHAPTER 6. PARALLELIZATION OF TRUSSTREMARK ALGORITHMS

Previous Chapter (i.e., CHAPTER 5) described the approach taken by TruSStReMark to perform service selection and recommendation by suggesting algorithmic modifications. Although, these two approaches performed better with respect to the quality of the results, in empirical evaluations (i.e., as indicated in Subsection 5.3), when used with the larger datasets from Amazon and Android marketplaces, they could not offer real-time or near-real-time performance. This chapter discusses the parallelization of these two approaches using the map-reduce paradigm. We note that the processing of reviews, due to their independent natures, can occur in parallel and thus achieve better performance. These parallelized versions are empirically evaluated using extended versions of publically available review data from Amazon Snap dataset and Android marketplace dataset.

First, we performed and compared the runtime variation of the serialize versions of the algorithms (i.e., Tr-CBF-Rank presented in Subsection 5.2.1 and Tr-CLF-Rec presented in Subsection 5.2.2) with the increasing number of reviews in the datasets. As the number of reviews increases, both the number of users and number of apps also increases. Due to this exercise, it was clear that both prevalent and trust-based algorithms take more time to perform their operations. However the trust-based algorithms increase at a much larger rate, due to the sentiment analysis operations on the reviews, the conversions of aggregated sentiments to subjective logic based tuples, and the subjective logic-based calculations of the selected set of QoS-related sentences.

Therefore, we proposed that this increase in time can be reduced significantly by parallelizing and running the algorithms (named pEbRanknRec) using a framework such as MapReduce based Hadoop environment using an Amazon EMR (Elastic MapReduce) cluster. Additionally, pEbRanknRec is augmenting the process by passing textual reviews from users to calculate the sentiment expressed about each user's targeted QoS properties of apps. These QoSs for apps can be calculated from the keywords, from service description and from user reviews. Using TextBlob library [84] inside each sentence, the algorithm calculates the sentiments of user reviews and cached them. SL provides the necessary operators to aggregate different opinions about a service QoS from different users depending on the situation. The conjunction, consensus and ordering operators (which are listed in Table 4-3) are used in pEbRanknRec algorithms. The positive polarity of SA is a measure of belief in the textual evidence and similarly, the negative polarity is a measure of disbelief. Naturally then, subjectivity is the indication of certainty when high and uncertainty when low. Therefore, without loss of generality, we adapt the linear conversion from SA to SL as expressed in Subsection 5.1.3 which describes part of the calculations done inside pEbRanknRec algorithms.

This framework uses the Hortonworks Data Platform (Figure 6.1), which is an open-source Hadoop distribution with pre-configured packages and tools such as, YARN (i.e., the resource and job management engine), HDFS (i.e., Hadoop Distributed File System) and HIVE (i.e., the data warehouse infrastructure).

Periodically during the offline phase, the custom review scrapper loads the most recent batch of the reviews from the marketplace to the HDFS. This ETL processing (i.e., Extract, Transform and Load) part of the offline phase can be configured to run either hourly, daily or off-peak load times. For example, a MapReduce job runs offline periodically to calculate vectors (of size k keywords) by parsing a new service description (i.e., identified using unique item id from the reviews) using a NLP technique such as TF-IDF (all data structures are cached and reused if no updates detected).

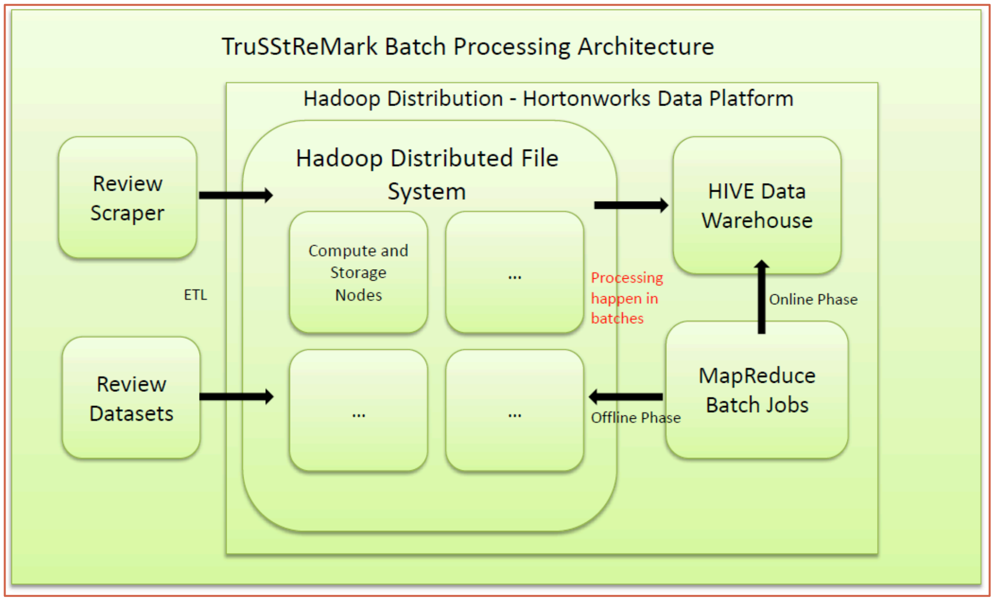

TruSStReMark Batch Processing Architecture
Hadoop Distribution - Hortonworks Data Platform
Review Scraper
ETL
Review Datasets
Hadoop Distributed File System
Compute and Storage Nodes
...
...
...
Processing happen in batches
HIVE Data Warehouse
Online Phase
MapReduce Batch Jobs
Offline Phase

Figure 6.1 TruSStReMark Batch Processing Architecture

Given a review dataset from a marketplace which includes service descriptions, user-service relations, and set of reviews by the users, the batch processing version of the algorithms of pEbRanknRec performs dataset transformation to partition (i.e., possible due to the review independence property) such that HDFS could handle the correct service spaces, user spaces and user-service spaces for each mapper. Also during the offline phase MapReduce batch jobs are submitted to extract sentiments (as described in Figure 5.7) and load to a structured format in the Hive warehouse, and then convert those sentiments to subjective logic based tuples to the appropriate location in the Hive warehouse. Finally, when the new data is available in the hive warehouse, other batch jobs use subjective login operators to update trust vectors in the user space and the service space with a fresh set of subjective logic tuples, which are organized by QoSs.

6.1 pEb-CBF-Rank and pEb-CLF-Rec

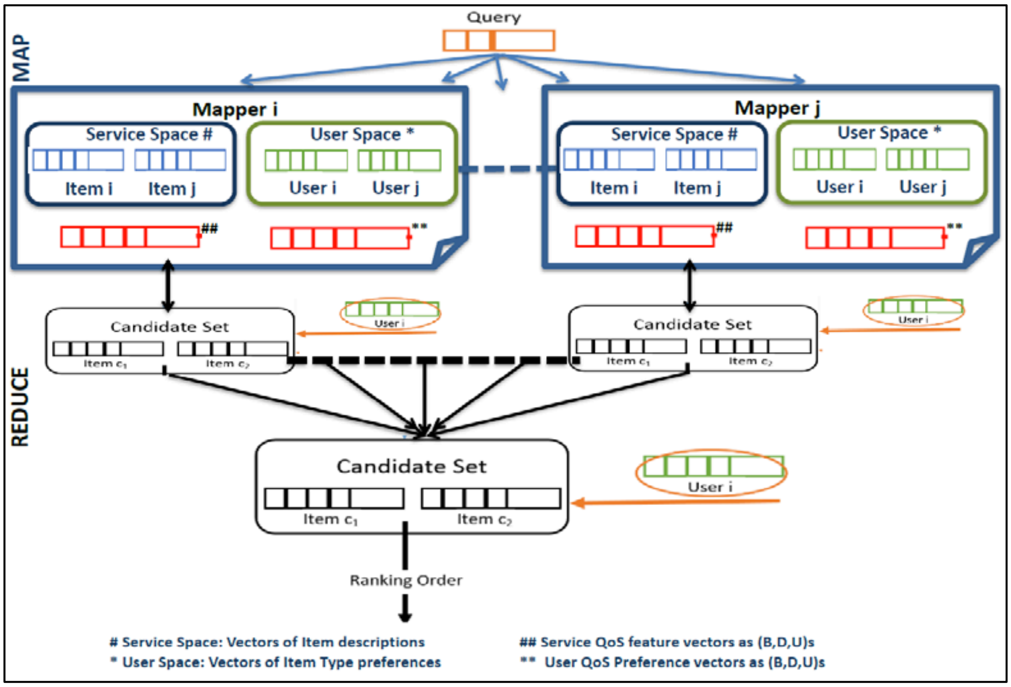


Figure 6.2 Online Phase of the evidence-based ranking (pEb-CBF-Rank)

The goal of the online phase of the parallelized TruSStReMark framework (i.e., pEbRanknRec approach as indicated in Figure 6.2 and Figure 6.3) is to improve performance and confidence while searching and recommending software apps by parallelizing the previous algorithms of aggregating external evidences available as textual reviews. First, it has a list of named entities for each user (i.e., a profile of important service QoS feature keywords). It performs the improved evidence-based search and recommendation algorithms (i.e., as indicated in Figure 6.4 and Figure 6.5) in parallel as follows: each user query for items/apps/apps is tagged with evidences of these feature vectors are used (i.e., which are calculated from users reviews during the offline phase of the algorithm). Since, structured data (i.e., user spaces, item spaces and their corresponding trust scores) are available in the data warehouse each mapper can now handle the correct service spaces, user spaces and user-service spaces as indicated in Figure 6.4 and Figure 6.5.

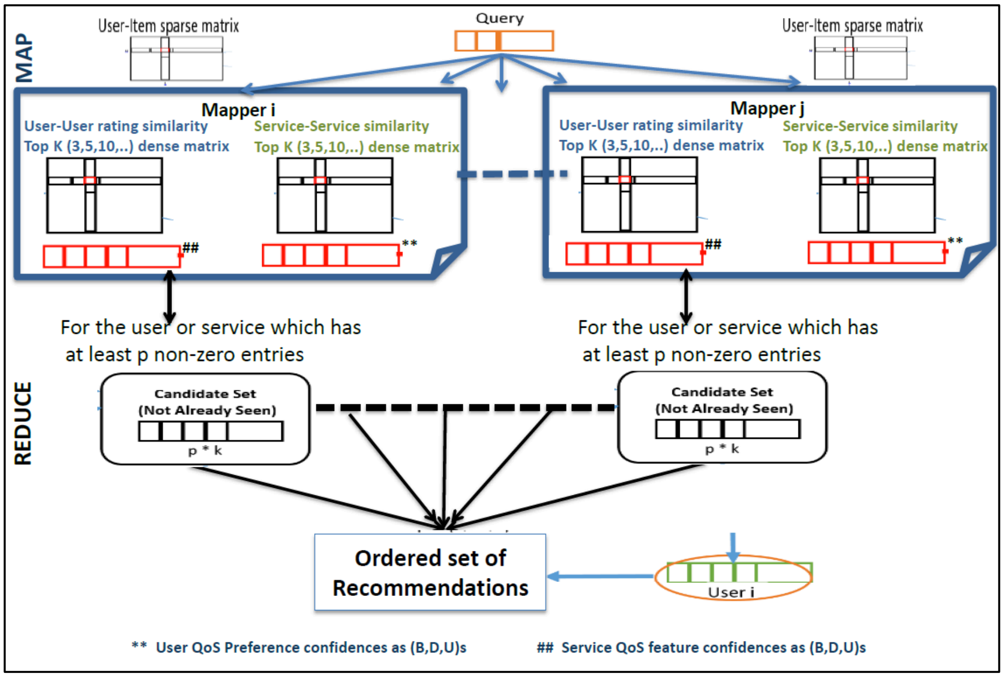

MAP
User-Item sparse matrix
Query
User-Item sparse matrix
Mapper i
User-User rating similarity
Top K (3,5,10,..) dense matrix
Service-Service similarity
Top K (3,5,10,..) dense matrix
##
**
Mapper j
User-User rating similarity
Top K (3,5,10,..) dense matrix
Service-Service similarity
Top K (3,5,10,..) dense matrix
##
**
For the user or service which has at least p non-zero entries
For the user or service which has at least p non-zero entries
REDUCE
Candidate Set (Not Already Seen)
p * k
Candidate Set (Not Already Seen)
p * k
Ordered set of Recommendations
User i
** User QoS Preference confidences as (B,D,U)s
## Service QoS feature confidences as (B,D,U)s

Figure 6.3 Online phase of the evidence-based recommendations (pEb-CLF-Rec)

For each service inside each mapper, the basic content-based algorithm the following parts are executed. During the offline phase, we perform the preprocessing parsing stage after partitioning and loading data to the HDFS and another set of jobs will generate the metadata and load them to the HIVE tables. When a query is forwarded to the system, during the online phase of the algorithms, each mapper uses part of the meta-data by executing the algorithm to measure accuracy and performance matrixes. Then we randomly pick approximately 5% of the users from each dataset to remove one service from each of the selected users and we parse each of the dataset though both prevalent and evidence-based algorithms to generate top-N search and recommendation (S&R) result sets for each user. If the test service is found in the set (at any particular rank i) of resulting apps list to the relevant user then it is considered as a Hit. We use a custom range petitioner to sort the candidate set directed to each reducer to produce a globally sorted candidate service set based on their confidence scores.

```
pEb-CBF-Rank: Service Reviews, Descriptions and Query
For Each Selected Interval (0)(BY Hour, Day, OR Review Size)
   Push new/updated review data to HDFS
Each Mapper in Parallel (1) (only for new/updated reviews):
  For Each SERVICE (S[] as Service Vector)
    For All REVIEWS (T as Combined Text Document of R,D)
       Calculate TF-IDF and select top k and cache
  Cached:  S[k] Service Vector (Sorted from TF-IDF score)
Each Mapper in Parallel (2): (only for new/updated reviews):
  For Each USER (U[] as User Vector)
      For All REVIEWS (T as Combined Text Document of R,Q)
       Calculate TF-IDF and select top k and cache
  Cached:  U[k] User Vector (Sorted from TF-IDF score)
Each Mapper: with S[] (3)  and  with U[] (4)
  For Each SERVICE (S[]/U[] as QoS <B,D,U> tuples){
   For Each REVIEW (T as Text)(1 to N)   {
     For each QoS in S[]/U[] {
       In Parallel Calculate SENTIMENT of T (Polarity, Subjectivity)
       In Parallel Convert (P,S) to <B,D,U>
         Calculate W - TIME-SENSITIVITY of T (Range 0 to 1)
          [('ReviewDate'-'ServiceFirstAvailableDate') /
           ('Today'- 'ServiceFirstAvailableDate')]
       In Parallel Update S[i]/U[i] <B,D,U> (Consensus OR Conjunction)
           Based on current <B,D,U> : subject to weights of W
  Cached:  Produce S*[], U*[] (with evidence <B,D,U>s)
  // User and Service Space Metadata generation Complete!
Each Mapper in Parallel: (With respective to Q)
  For each size k and perform CBF on S[]/U[]
Each Reducer in Parallel:
  Produce local candidate set C[k]
    Locally sorted: C[k] based on search ranking score
Use custom range practitioner (sort globally) produce C*[k]
```

Figure 6.4 Evidence-based search and ranking (pEb-CBF-Rank)

When all the results are obtained, we produce another set randomly to cross validate and generate results iteratively to get the average of all matrices. To compare the sequential algorithm with the parallel algorithm we use average end-to-end time calculations. In the parallel version of the algorithm, the end-to-end time is measured by combining the average mappers time and reducer times in online phase with offline phase.

```
pEb-CLF-Rec: Service Reviews, Descriptions and Query
Execute Steps (0)-(3) Cntd Addition to Figure 3
Each Mapper in Parallel: (only for new/updated reviews):
  Pass all reviews and augment or update
  Cached: produce local U-I [] rating vector
Each Mapper in Parallel: User's U[], U[QoS] <B,D,U> tuples
 For Each CANDIDATE SERVICE (S[], S[QoS] <B,D,U> tuples)
   Calculate Two COSINE Similarity
       Each based on the vales from
        - (1) S[] with U[]
        - (2) S[QoS] with U[QoS] [For Each (B,D,U)]
    If two users are A and B: For each common R rating,
    Then COSINE Similarity: ((Σ Ar * Br)/(||A|| * ||B ||))
     Where ||A|| = ( Σ (Ar)2 )0.5
// User-Service Space Metadata generation Complete!
Each Mapper in Parallel: (With respective to Q)
  For each size p,k and perform CLF on S-I[]
Each Reducer in Parallel:
  Produce local candidate set C[p*k]
    Locally sorted: C[p*k] based on search ranking score
Use custom range practitioner (sort globally) produce C*[p*k]
```

Figure 6.5 Evidence-based Recommendation (pEb-CLF-Rec)

We evaluate each of the result-sets for both pEb-CBF-Rank and pEb-CLF-Rec using HR (Hit Ratio) and ARHR (Average Reciprocal Hit-Rank) measurements that are commonly used in S&R evaluations. The HR is a measure of the number of test apps that each algorithm included in the result-set of S&R. If n is total #apps of the result-set, then HR = (Number of Hits)/n. The ARHR is for evaluating the relative significance of the position (p_i) in the result-set of S&R. Hits which appear earlier are scored higher than the Hits that appear later in the result-set ordering. If service is positioned at pi in the result-set of size n, then ARHR = $(1\backslash n) \sum(1\backslash p_i)$. The globally sorted final result-sets are evaluated using the rank (i.e., number of 3,5,10,15 or 20 set of apps) by producing 100 times to get the average. Next, the experiments are carried out to show the improvement in quality of the results of service selection and recommendation when parallelized TruSStReMark based algorithms are applied. Then those results are compared with the prevalent CLF approach which are presented in the next Chapter.

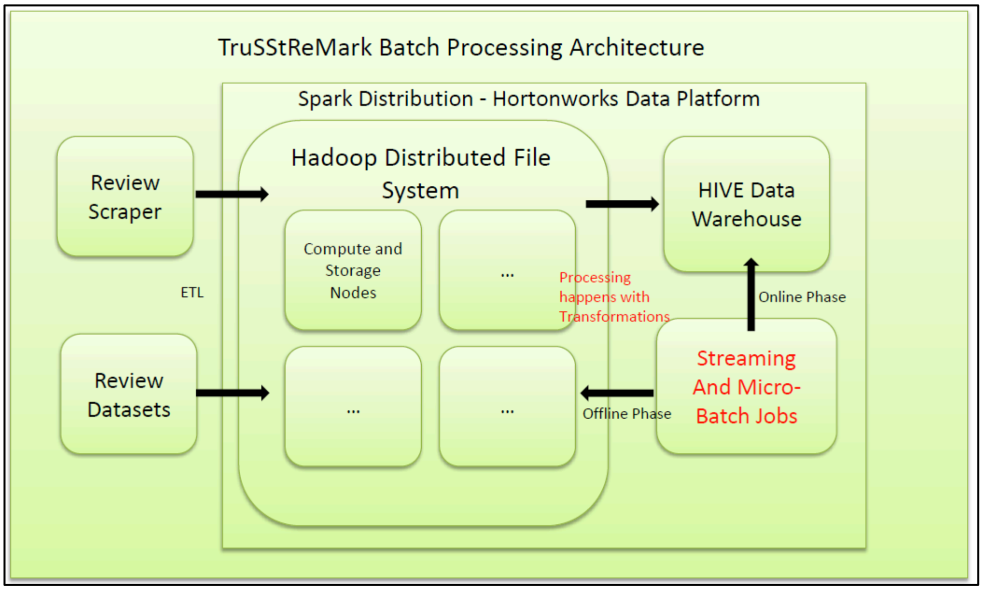

TruSStReMark Batch Processing Architecture
Spark Distribution - Hortonworks Data Platform
Review Scraper
ETL
Review Datasets
Hadoop Distributed File System
Compute and Storage Nodes
...
...
...
Processing happens with Transformations
HIVE Data Warehouse
Online Phase
Streaming And Micro-Batch Jobs
Offline Phase

Figure 6.6 TruSStReMark Stream Processing Architecture

This section describes TruSStReMark adaptation to the stream processing concepts. Since there were improvements with parallelized versions of the algorithms (i.e,. pEb-CBF-Rank and pEb-CLF-Rec), we also realized that MapReduce based batch processing is also limiting the performance of these algorithms. Therefore, we thought that this increase in time can be reduced further by running the algorithms using a data streaming framework such as Apache Spark based Sparks Streaming oriented processing to replace the Apache Hadoop based batch processing framework. Since Spark Streaming can directly use HDFS, the overall architecture remains the same (as indicated in Figure 6.6). The advantage that stream processing brings is its ability to perform micro-batch based transformations, without flushing the data to the disk. For example, earlier, during the batch processing, TruSStReMark executes multiple jobs in batches (e.g., to extract related sentiments from QoS of apps using the reviews, to convert sentiments to subjective logic based tuples and augment apps' trust vectors using subjective logic based operators). The end-to-end time is measured as indicated earlier to perform each streaming step and iteratively to measure the average runtime. The streaming version performs faster which is expected due to not having the overhead associated with the batch oriented processing associated with MapReduce jobs. For example, each time a batch job executes the results are needed to be written to distributed disk such that the next batch jobs have a consistent view of the data to preserve the states. However the streaming version of this does not need to write data and preserved them in the distributed version of the memory to perform sequence of jobs by having a consistent view of the states. This improves the performance of our algorithms while performing the bath oriented jobs to calculate the QoS oriented sentiments, to convert them to the associated subjective logic based tuples, and then update the service trust scores using subjective logic based operators. The experiments are performed with this improved stream processing architecture using Amazon EMR (Apache Spark with Spark Streaming) clusters. The results and analysis are discussed in next subsections.

6.2 Results and Analysis of pEb-CBF-Rank and pEb-CLF-Rec

To empirically evaluate this parallelized approach, we increase the size of data to 10 times the size of both the sequential experiments datasets (i.e., when algorithms perform inside a single instance). We then apply Search/Ranking (pEb-CBF-Rank) to the Android marketplace Review dataset [10] (which now includes 34,169,077 reviews of mobile 2,702,594 apps and apps) and apply Recommendation techniques (pEb-CLF-Rec) to the

Amazon Marketplace Reviews dataset [9] (which now includes Reviews 34,686,770 reviews of products 2,441,053) respectively. Both datasets had around 6 million users, users > 50 reviews more than 50 thousand, 80 median words per review and timespan Jan 1995 – October 2016. Our experimental setup was made up of Java and Python running on an environment containing Amazon EC2 (Amazon Elastic Computing Cloud) free tier Linux t2.micro instances with 64-bit platform support, 15 node EMR (Elastic MapReduce) Cluster running version 2.4 Hadoop.

Also, we present the results from the experiments conducted with a parallel version of the trust-based selection algorithm (i.e., pEb-CBF-Rank). For the search/ranking study we used the Android Marketplace dataset. The experiments compare the evidence-based approach for search/ranking with the prevalent approaches of Content-based filtering (i.e., CBF and sequential sEb-CBF-Rank, with parallel pEb-CBF-Rank). Table 6-1 indicates the HR and ARHR percentage improvements obtained by these approaches against Android marketplace-based dataset. It is evident from Figure 6.7 that the proposed parallel evidence-based search and ranking algorithm performs better than the prevalent content-based filtering (CBF) algorithm. This is equally true with its sequential approach in terms of HR and ARHR given the advantage of generating the results in real-time. We perfumed the algorithms iteratively 100 times to get the average of quality matrices including the average runtime. Since, we randomly take a portion of the dataset out to cross validate the results at each step there is a slight difference between the sequential and parallel version of the results. However they perform nearly within the same range of values when we average the results.

Table 6-1 Percentage improvements of pEb-CBF-Rank

Technique / Result Set Size	**Top 3**	**Top 5**	**Top 10**	**Top 15**	**Top 20**
CBF (HR)	11.67	21.34	22.67	24.81	26.72
sEb-CBF-Rank (HR)	12.22	25.69	30.81	32.62	33.9
pEb-CBF-Rank (HR)	11.76	25.32	30.14	32.22	33.65
CBF (ARHR)	5.3	9.92	10.76	12.4	12.86
sEb-CBF-Rank (ARHR)	6.78	14.27	17.11	17.38	17.92
pEb-CBF-Rank (ARHR)	5.42	13.76	16.87	17.16	17.56

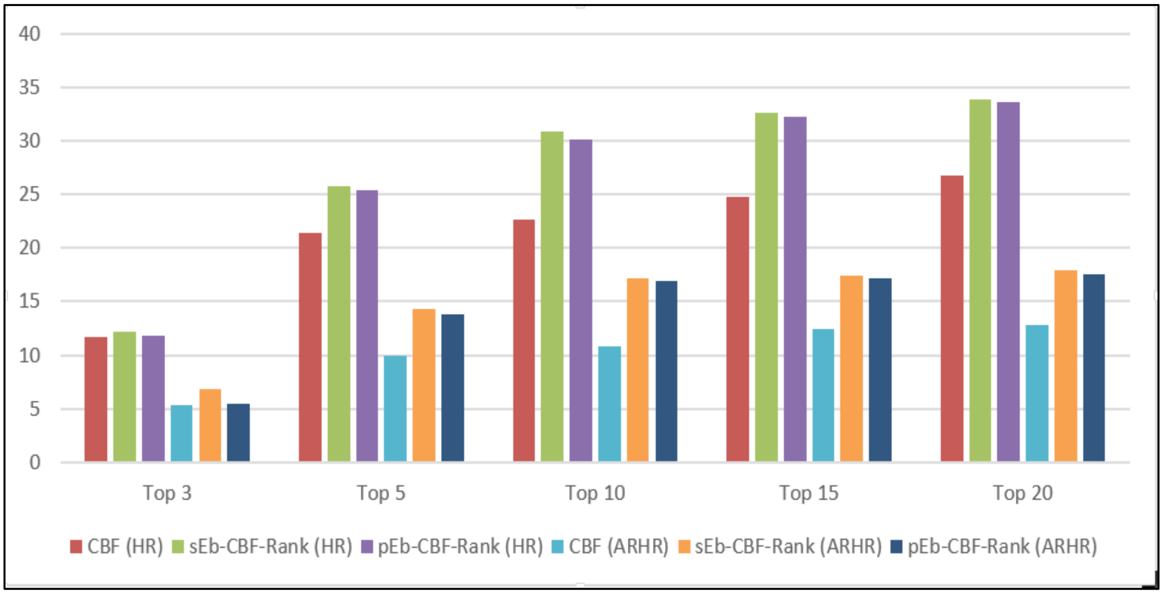


Figure 6.7 HR and ARHR comparison of pEb-CBF-Rank

Next, we present the results from the experiments conducted with parallel version of the trust-based recommendation algorithm (i.e., pEb-CLF-Rec). For this recommendation study we used the Amazon Marketplace dataset. The experiments compare the evidence-based approach for search/ranking with the prevalent approaches of Collaborative filtering (i.e., CLF, sequential version (sEb-CLF-Rank), and parallel version (pEb-CLF-Rec) of the algorithms).

Table 6-2 Percentage improvements of HR and ARHR of pEb-CLF-Rec

Technique / Result Set Size	Top 3	Top 5	Top 10	Top 15	Top 20
CLF (HR)	19.17	21.8	24.26	26.72	30.72
sEb-CLF-Rec (HR)	21.08	23.82	26.65	29.18	32.18
pEb-CLF- Rec (HR)	20.64	23.13	25.95	28.58	31.38
CLF (ARHR)	9.58	10.38	11.55	13.03	15.03
sEb-CLF- Rec (ARHR)	10.64	12.53	13.66	15.36	18.36
pEb-CLF- Rec (ARHR)	9.93	12.02	13.21	14.89	17.52

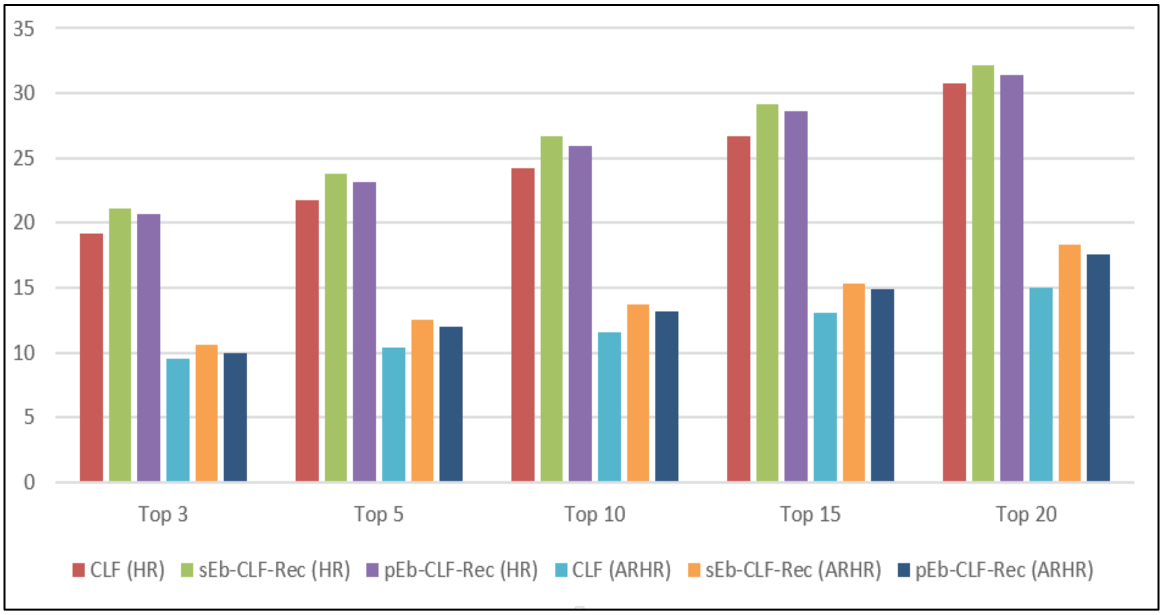


Figure 6.8 HR and ARHR comparison of pEb-CLF-Rec

Table 6-2 indicates the HR and ARHR percentage improvements obtained by these approaches against the Amazon marketplace-based dataset. It is evident from the Figure 6.8 that the proposed parallel evidence-based search and ranking algorithm perform better than the prevalent collaborative filtering approach (CLF) and equally with its sequential approach in terms of HR and ARHR given the advance of it performs in real-time. In summary, by analyzing the results, the percentage improvement of HR and ARHR is equal to prevalent approaches given the results were generated in near real-time. Again, as mentioned previously, we randomly take a portion of the dataset out to cross validate the results at each step there is a slight difference between the sequential and parallel version of the results. However they perform nearly during the same range of values when we average the results. This shows the importance of QoS consideration in our evidence-based approach in real-time, and this validates our approach.

As an additional step to verify the behavior of sequential and parallel version of the algorithms, we removed the random portion of the dataset partitioning. At each iteration of the experiment, both the sequential and parallel algorithms now receive the same test and training data without random selection between the approaches by manually overriding the data in HDFS. The experiment was performed 100 times to get the average values of both HR and ARHR matrices and Figure 6.9 indicates results of this additional experiment. As expected the results indicates that both sequential and parallel algorithms were able to produce the same HR and ARHR values, since they both received the same set of data sets.

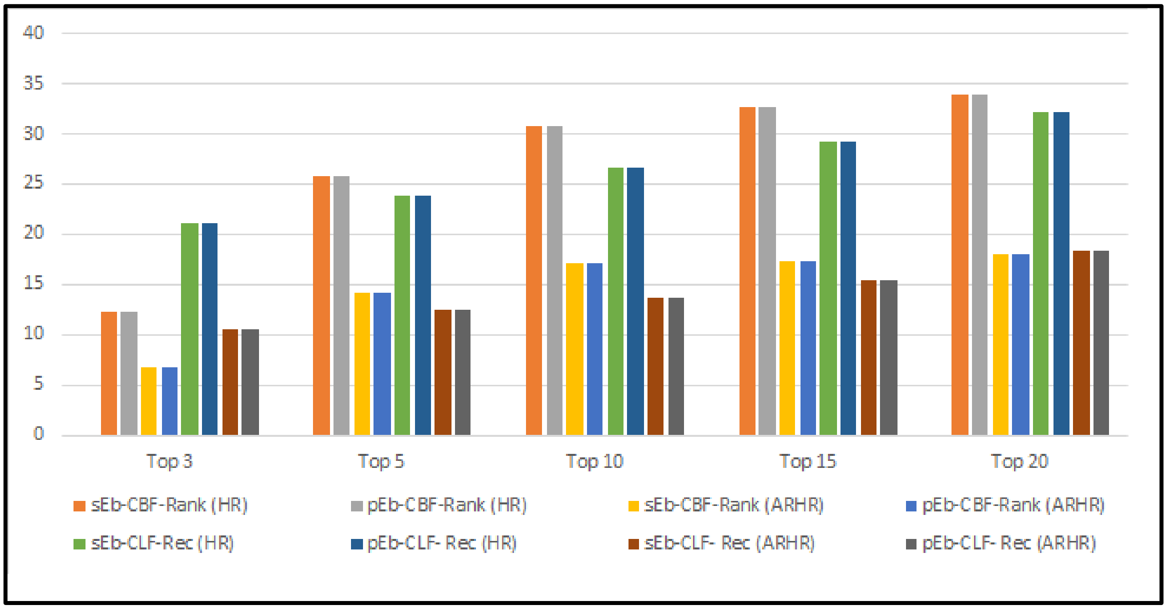


Figure 6.9 Results comparison of pEb-CBF-Rank without randomization

6.3 Results from converting batch processing to stream processing

Table 6-3 indicates the HR and ARHR percentage improvements by comparing the batch oriented version of the pEb-CBF-Rank (i.e., bt-pEb-CBF-Rank) to stream oriented version of the pEb-CBF-Rank (ie., st-pEb-CBF-Rank). As indicated earlier the slight differences are due to the random requirement of data partitioning to training and test datasets which is mandatory to calculate the HR and ARHR measures.

Table 6-3 HR and ARHR on batch and streaming experiments of pEb-CBF-Rank

Technique / Result Set Size	Top 3	Top 5	Top 10	Top 15	Top 20
bt-pEb-CBF-Rank (HR)	11.76	25.32	30.14	32.22	33.65
st-pEb-CBF-Rank (HR)	12.34	24.05	31.64	30.60	31.96
bt-pEb-CBF-Rank (ARHR)	5.42	13.76	16.87	17.16	17.56
st-pEb-CBF-Rank (ARHR)	5.69	13.07	17.713	16.30	16.68

Figure 6.10 indicates these results in graphical notation. As expected we concluded that apart from the slight differentiation of the quality measures due to the random requirements in calculating HR and ARHR, the batch and streaming version of the trust-based selection algorithms performs equally well in quality matrices.

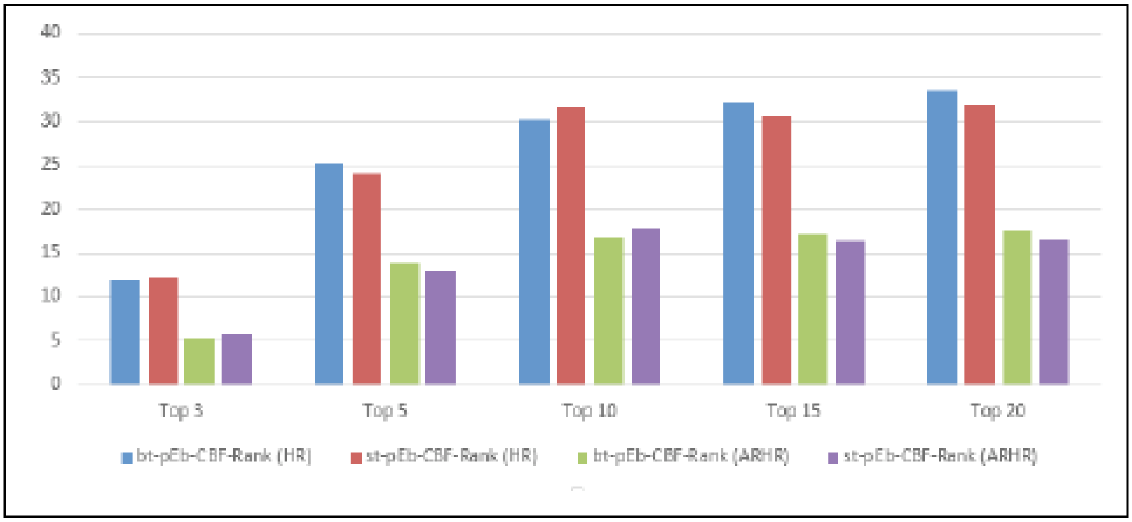


Figure 6.10 Quality comparison of bt-pEb-CBF-Rank and st-pEb-CBF-Rank

Similarly, Table 6-4 indicates the HR and ARHR percentage improvements by comparing the batch oriented version of the pEb-CLF-Rac (i.e., bt- pEb-CLF-Rac) to stream oriented version of the pEb-CLF-Rac (ie., st- pEb-CLF-Rac). As indicated earlier the variations of the values are due to the randomness of the data partition while calculating the quality matrices.

Table 6-4 HR and ARHR on batch and streaming experiments of pEb-CLF-Rac

Technique / Result Set Size	**Top 3**	**Top 5**	**Top 10**	**Top 15**	**Top 20**
bt-pEb-CLF-Rec (HR)	20.64	23.13	25.95	28.58	31.38
st-pEb-CLF-Rec (HR)	19.60	24.28	27.24	27.15	29.81
bt-pEb-CLF-Rec (ARHR)	9.93	12.02	13.21	14.89	17.52
st-pEb-CLF-Rec (ARHR)	9.43	12.621	13.87	14.14	16.64

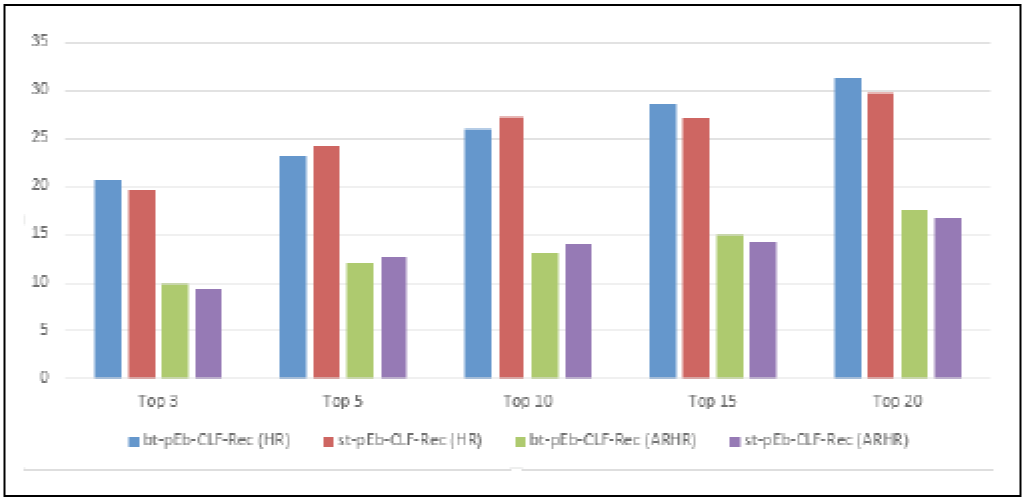


Figure 6.11 Quality comparison of bt-pEb-CLF-Rec and st-pEb-CLF-Rec

Figure 6.11 indicates the trust-based recommendation algorithms versions of the batch and streaming results in graphical notation. As expected we concluded that HR and ARHR matrices of the batch and streaming version of the trust-based recommendation algorithms performs equally well.

6.4 Performance and runtime analysis of the algorithms

We evaluated the performance of the pEb-CBF-Rank and the pEb-CLF-Rec against prevalent approaches. Since the main concern is performance of the algorithms, we compared the runtime comparison of algorithms (prevalent CBF/CLF with both sequential sEb-CBF/CLF-Rank/Rec, and parallel pEb-CBF/CLF-Rank/Rec) with the increasing number of reviews in the datasets. As the number of reviews increases, both the number of users and number of apps also increase.

To compare the sequential algorithm with the parallel algorithm we use average end-to-end time calculations. In the parallel version of the algorithm, the end-to-end time is measured by combining the average mappers time and reducer times in the online phase with the offline phase.

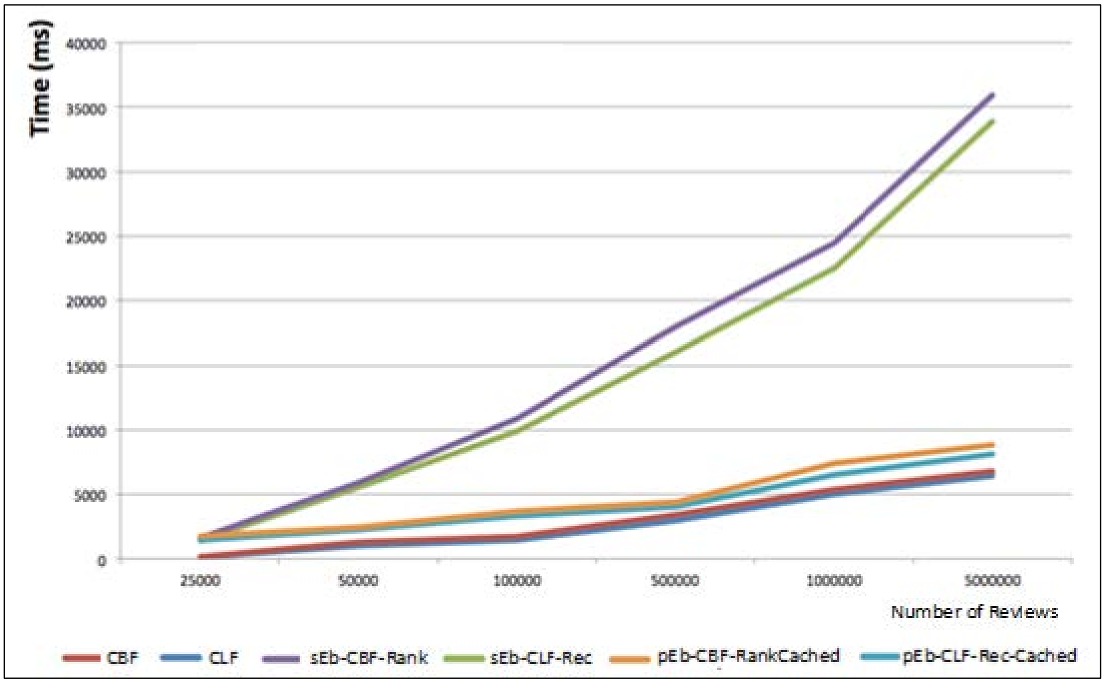


Figure 6.12 Aggregated average runtimes of the parallel algorithms

Also, caching experiments are performed by directly using the HIVE meta-data store. From the comparisons in Figure 6.12, it is clear that both prevalent and trust-based algorithms take more time to perform their operations. Since, the sequential evidence-based S&R algorithms increase at a much higher rate, due to its additional calculations. Our parallelized algorithms using MapReduce/Hadoop environment perform equally when the dataset size is much larger. Each data set inside the local node of the cluster is small hence, it was able to efficiently perform the algorithm locally in order to produce global results.

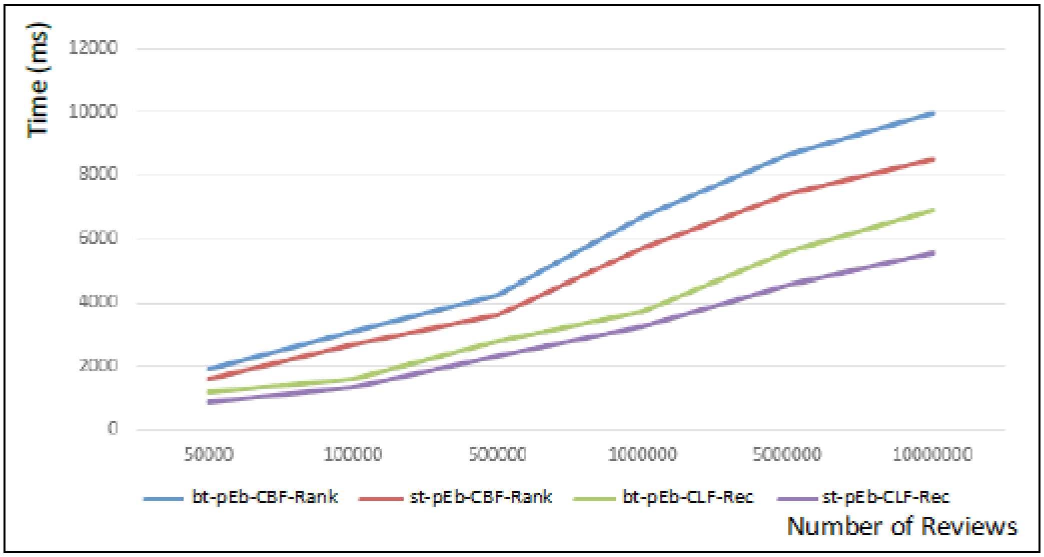


Figure 6.13 Aggregated average runtime of batch vs streaming approaches

Figure 6.13 indicates analysis about streaming versions of the algorithms in terms of average runtime compared to average runtime of the batch processing versions of the algorithms. The streaming version performs faster which is expected due to not having the overhead associated with the batch oriented processing that MapReduce jobs require. Each time a batch job executes the results need to be written to a disks which are distributed such that the next batch jobs have a consistent view of the data in order to preserve the states. We also noted that the gap is being widen when the number of reviews are increased. This is also expected, as when the load increases the stream processing performs much better than batch oriented processing version of the algorithms.

In Summary, using the TruSStReMark framework we suggested algorithmic modifications necessary to parallelize and perform evidence-based search and recommendation (S&R) of apps. The framework uses datasets from Amazon and Android marketplaces. The two parallel algorithms (i.e., pEb-CLF-Rec and pEb-CLF-Rec) in the pEbRanknRec algorithms were executed using an EMR environment in AWS using Hadoop echo-system. The main algorithmic calculations are based on sentiments from the large volumes of textual reviews, which are then numerically converted to subjective logic based BDU tuples to apply aggregation operators). Hive warehouse is used as the caching meta-store, where algorithms (i.e., st-pEb-CLF-Rec and st-pEb-CLF-Rec) quickly find the aggregated trust scores of QoS related to apps. When compared to prevalent approaches the results indicate that parallelized algorithms improve the average performance of the trust-based algorithms and were able to generate better or equally well with both datasets in terms of HR and ARHR. Next, the batch processing part of the Hadoop jobs are streamlined using Sparks streaming techniques. When compared to the batch oriented technique, the stream oriented algorithms (i.e., st-pEb-CLF-Rec and st-pEb-CLF-Rec) were able to perform equally well with the advantage of reduced aggregated average runtimes. The advantage of our approach is that it is based on heterogeneous and dynamic software features (QoSs) of apps that enable better temporal comparisons between software apps. The results of this study indicate that the evidence-based approaches provide better selections and recommendations both in terms of quality and relative ranking of apps.

CHAPTER 7. CONCLUSION AND FUTURE WORKS

This dissertation has described techniques to model, quantify, specify, and monitor the trust of software apps. It also provides methods to analyze, and aggregate, external reviews of software apps and uses them to perform trust-based service selection, and recommendation. We present four contributions, which extend the current state of the art with respective to the trust of software apps in online marketplaces. The list of contributions are:

- Surveying prevalent trust-related views and associated models to define and quantify the trust of a service.
- Extension of multi-level software specifications to represent trust-based attributes.
- TruSStReMark framework to perform trust-based monitoring, selection, and recommendation.
- Performance improvement to achieve real-time query processing.

Below we present the summary of our achievements in each of these four contributions.

- Surveying existing trust models for software apps:

We identified and analyzed different interpretations of trust in distributed software systems spanning over 25 years. We also categorized differences and commonalities of various prevalent trust views and models. These categories show that the community has accepted inherent attributes of trust such as subjectivity and uncertainty.

It also shows that important factors such as reputation of both truster and trustee, truster’s willingness to be vulnerable, trustee’s willingness to acquire benefits from truster, and functional and QoS capabilities of both truster and trustee are overwhelmingly accepted by the community. The result of this survey indicated that the presence of many models having their own view of trust (e.g., application specific view, and targeting a specific phase of the software life cycle), makes it a significant challenge for any two trust models to exchange information.

Therefore, we have suggested, in this dissertation, a set of principles that are necessary to understand and evaluate trust in software-intensive systems throughout their entire life-cycle. These eight trust principles are requirements, representation, associated knowledge base, specification, comparison and selection, countermeasures, composition, and evaluation associated with the life-cycle of software apps. As a case study to show the application of the principles, we have applied them to a set of apps that form a distributed tracking system.

- Extension of multi-level specifications to represent trust-based attributes

Our framework, TruSStReMark, represents trust of software apps. It uses the concepts of theory of evidence, subjective logic, sentiment analysis, and multi-level specifications to define a trust contract for software apps. The proposed trust contract extends the multi-level contract techniques to include the trust as a critical factor while developing apps. Furthermore, it presents how to quantify trust related attributes to improve trustworthy service selection and recommendation.

The TruSStReMark framework focuses only on using external evidences available from the user reviews to quantify trust of apps in a marketplace, mainly due to the lack of publically available internal evidences associated with apps. TruSStReMark processes available reviews of each service and captures the expressed sentiments about the important non-functional attributes and quantifies them using subjective logic-based <B,D,U> tuples (i.e., <Belief, Disbelief, Uncertainty> notation). This approach uses sentiments of the sentences, which describe non-functional qualities of each service and converts them from sentiment values (+/-polarity, subjectivity) to subjective logic tuples. Then, it aggregates these quantified <B, D, U> tuples using suitable operators available from subjective logic (e.g., consensus and conjunction operators) to create a single tuple for each service. This <B,D,U> tuples form the basis of the trust contract of a service.

This trust contract extends the concepts of multi-level specifications with an additional level denoting the trust values. The reason for that is, it resides after the negotiable QoS attributes of the service specification. The quantified and aggregated values of identified trust attributes which are presented in this trust contract (as subjective logic based tuples), contain references for other related attributes from existing levels (e.g., the trustworthiness of a service's QoS values). Since this contract enables long term monitoring and aggregation of trust associated with software apps, it can improve the performance of service selection and recommendations.

- Experimenting with the TruSStReMark framework

The TruSStReMark framework aggregates the above mentioned evidences into a trust contracts, then it performs trust-based service selection and recommendation using our proposed algorithms, which make efficient use of these trust contracts. As a part of our TruSStReMark framework, we suggest modifications and enhancements of the two prevalent algorithms (CBF and CLF) to include trustworthy service selection and recommendation principles. Our algorithms enable updating trust values to cater dynamic nature of apps and enable continuous monitoring of the trustworthiness of apps.

Our trust-based selection and recommendation approach has two main advantages. It broadens the spectrum from the single-dimensional analysis of apps (i.e., star rating of apps) to a multi-dimensional analysis (i.e., aggregating of trust-based attributes of apps), which may lead to further interesting discoveries, and it also enables comparisons between software apps based on different QoS features of apps. The experimental results indicate that the trust-based approaches provide better selections and recommendations of software apps both in terms of quality, relevance and relative ranking of the result-sets. Specifically, our approach performs better than prevalent approaches when used with two publically available datasets using root mean square error, precision, and recall, Hit-Rate (HR), and Average Reciprocal Hit-Rate (ARHR) metrics.

- Performance improvements of the TruSStReMark framework

Publically available apps from the Android and Amazon marketplaces are used as the input datasets during the experimentation of the proposed framework. As the number of apps and number of parallel queries increases, both online and offline phases of the trust-based algorithms need to be enhanced with parallelization techniques to produce results in real-time. The performance constraints of increased time required by the trust-based algorithms are addressed by parallelizing both during offline phase (i.e., ETL -Extract, Transform and Load, categorization and organization of evidences) and online phase (i.e., execution of the algorithms to analyze evidences in real-time). The online phase uses the data warehouses to generate and update metadata related to trust contracts and to perform analysis (i.e., execution of the algorithms to analyze evidences in real-time). The

TruSStReMark is parallelized using MapReduce concepts with the Apache Hadoop ecosystem (i.e., experiments are performed in using Amazon EC2 with Amazon EMR at AWS). Also, the external evidences are in Hadoop Distributed File System (HDFS) and executing MapReduce jobs regularly to update the trust-based scores and service metadata used by the real-time queries. This enables processing of parallel queries from many different users by analyzing a large number of apps which improves the performance to produce results in near real-time.

Further analyses of these algorithms revealed the batch-oriented operations in Hadoop is a bottleneck and hinders the overall performance. Hence, we reduced the Hadoop-based batch-oriented processing with the adaptation of Spark and Spark Streaming techniques. This improves the efficiency of both online and offline phases.

There are many possible future directions of the techniques presented in this dissertation. For example, a possible future direction related to the proposed trust model is to apply and evaluate this model to other complex software systems. Future explorations in the area of the proposed trust contract include, experimenting with different trust representations, investigating contracts that are incomplete and/or are missing critical information, analyzing trust aspects on specifications in different domains such as economics contracts, and legal contracts. Other possible future directions include improving the degree of certainty in predicting trust value of a composed systems, incorporating domain knowledge, and learning compatibility of apps dynamically, explore other similarity measurement techniques of apps, and investigate group-based selection and recommendation techniques.

LIST OF REFERENCES

LIST OF REFERENCES

[1] SAS (Statistical Analysis System) Institute, *The Internet of Things, Opportunities and Applications across Industries, Accessed on 08/15/16, URL: http://www.sas.com/en_us/offers/sem/iia-internet-of-things-108110.html*, 2015.

[2] Dave Evans (Cisco Inc.), *The Internet of Things, How the Next Evolution of the Internet is Changing Everything, Accessed on 08/15/16, URL: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf*, 2011.

[3] Amazon Inc., *Amazon Appstore, Accessed on 08/15/16, URL:www.amazon.com/appstore,* 2016.

[4] Canonical Inc., *Unbuntu Appstore, Accessed on 08/15/16, URL:https://apps.ubuntu.com/cat/,* 2016.

[5] Google Inc, *Google's Play Apps Marketplace, Accessed on 08/25/16, URL: https://play.google.com/store/apps,* 2016.

[6] Apple Inc., *Apple Appstore , Accessed on 08/15/16, URL:www.apple.com/appstore,* 2016.

[7] BlackBerry Inc., *BlackBerry World, Accessed on 08/15/16, URL:https://appworld.blackberry.com/webstore,* 2016.

[8] Microsoft Inc., *Microsoft Phone Store, Accessed on 08/15/16, URL:www.microsoft.com/en-us/store/apps/windows-phone,* 2016.

[9] Microsoft Inc., *Microsoft Appstore, Accessed on 08/15/16, URL:https://www.microsoftstore.com/store/msusa/en_US/home,* 2016.

[10] Amazon Inc, *Amazon's Software Marketplace, Accessed on 08/10/16, URL: https://aws.amazon.com/marketplace,* 2016.

[11] Google Inc, *Google's Web Application Marketplace, Accessed on 08/25/16, URL: https://apps.google.com/marketplace/,* 2016.

[12] L. S. Gallege, D. U. Gamage, J. H. Hill and R. R. Raje, "Towards a Comprehensive Method for Integrating Trust into Enterprise DRE Systems," in *Proceedings of Real-time Computing Systems and Applications (RTCSA)*, 2011.

[13] L. S. Gallege, D. U. Gamage, J. H. Hill and R. R. Raje, "Understanding the trust of software-intensive distributed systems," *Concurrency and Computation: Practice and Experience,* vol. 28, no. 1, pp. 114-143, 2016.

[14] W. Shouxin, Z. Li and W. Shuai, "A measurement approach of trust relation in web service," *Journal of Communication and Computer,* vol. 6, no. 8, 2009.

[15] Trusted Computing Consortium, *Trusted Computing Group, Accessed on 08/01/16, URL: www.trustedcomputinggroup.org,* 2003.

[16] Wikipedia Inc, *Collaborative Filtering, Accessed on 08/02/16, URL: https://en.wikipedia.org/wiki/Collaborative_filtering,* 2016.

[17] Wikipedia Inc, *Content Based Filtering, Accessed on 08/02/16, URL: https://en.wikipedia.org/wiki/Recommender_system#Content-based_filtering,* 2016.

[18] Statistra Inc, *Statistra (The Statistics Portal), Accessed on 08/01/16, URL: http://www.statista.com/,* 2016.

[19] Google Inc, *Google's Android Platform, Accessed on 08/25/16, URL: https://www.android.com,* 2016.

[20] Yahoo Inc., *Yahoo Weather Service, Accessed on 08/01/16, URL: https://mobile.yahoo.com/weather/,* 2016.

[21] Weather.com, *Weather.com Weather Service, Accessed on 08/01/16, URL: https://weather.com/apps,* 2003.

[22] G. Shafer, "A Mathematical Theory of Evidence," *Whitepaper, Princeton University Press,* 1976.

[23] A. Jøsang, "Artificial Reasoning with Subjective Logic," in *In Proceedings of the Second Australian Workshop on Commonsense Reasoning, Perth, Australia,* 1997.

[24] N. Indurkhya and F. J. Damerau, Handbook of Natural Language Processing, 2nd ed., Chapman and Hall Inc., 2010.

[25] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Foundations and Trends in Information Retrieval,* vol. 2, 2008.

[26] T. White, Hadoop: The Definitive Guide, O'Reilly Media, Inc., 2012.

[27] Apache Inc, *Hadoop Distributed File System (HDFS), Accessed on 01/30/16, URL: https://en.wikipedia.org/wiki/Apache_Hadoop#HDFS,* 2016.

[28] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design and Implementation*, Berkeley, CA, USA, 2004.

[29] H. Karau, A. Konwinski, P. Wendell and M. Zaharia, Learning Spark: Lightning-Fast Big Data Analytics, 1st ed., O'Reilly Media, Inc., 2015.

[30] Apache Inc, *Spark Streaming, Accessed on 08/01/16, URL: https://en.wikipedia.org/wiki/Apache_Spark#Spark_Streaming,* 2016.

[31] D. Gambetta, "Can We Trust Trust?," *Trust in Making and Breaking Cooperative Relations,* pp. 213-237, 1988.

[32] L. Xiong and L. Liu, "PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities," *Transactions on Knowledge and Data Engineering,* vol. 16, pp. 843-857, 2004.

[33] K. O'Hara, H. Alani, Y. Kalfoglou and N. Shadbolt, "Trust Strategies for the Semantic Web," in *Proceedings of the 2004 International Conference on Trust, Security, and Reputation on the Semantic Web - Volume 127,* 2004.

[34] J. Huang and D. Nicol, "A calculus of trust and its application to PKI and identity management," in *Proceedings of 8th Symposium on Identity and Trust on the Internet,* New York, USA, 2009.

[35] S. P. Marsh, "Formalising trust as a computational concept," Technical Report - Department of Computing Science and Mathematics, University of Stirling, 1994.

[36] C. Jouvray, M. Sall and A. Kung, "Enforcing trust in embedded systems using models," in *Proceedings of International Workshop on Security and Dependability for Resource Constrained Embedded Systems*, 2010.

[37] A. Jøsang, "An Algebra for Assessing Trust in Certification Chains," in *Proceedings of Network and Distributed Systems Security Symposium (NDSS'99). The Internet Society*, 1999.

[38] G. Theodorakopoulos and J. S. Baras, "On trust models and trust evaluation metrics for ad hoc networks," *Journal on Selected Areas in Communications*, vol. 24, pp. 318-328, 2006.

[39] P. Herrmann and H. Krumm, "Trust-Based Monitoring of Component-Structured Software," *Praxis der Informationsverarbeitung und Kommunikation,* 2004.

[40] V. Alagar and M. Mohammad, "A component model for trustworthy real-time reactive systems development," in *Proceedings of FACS Sophia-Antipolis*, 2007.

[41] Z. Yan, "A comprehensive trust model for component software," in *Proceedings of 4th international workshop on Security, privacy and trust in pervasive and ubiquitous computing*, New York, NY, USA, 2008.

[42] A. Abdul-Rahman and S. Halles, "A Distributed Trust Model," in *Proceedings of New security paradigm workshop*, New York, NY, USA, 1997.

[43] S. D. Kamvar, M. T. Schlosser and H. Garcia-Molina, "The Eigentrust algorithm for reputation management in P2P networks," in *Proceedings of 12th international conference on World Wide Web*, New York, NY, USA, 2003.

[44] N. Stakhanova, S. Ferrero, J. S. Wong and Y. Cai, "A Reputation-based Trust Management in Peer-to-Peer Network Systems.," in *Proceedings of Parallel and Distributed Computing Systems*, 2004.

[45] R. Chen, X. Zhao, L. Tang, J. bin Hu and Z. Chen, "CuboidTrust: A Global Reputation-Based Trust Model in Peer-to-Peer Networks.," in *Proceedings of ATC*, 2007.

[46] B. Meyer, C. Mingins and H. Schmidt, "Providing Trusted Components to the Industry," *IEEE Computer,* vol. 31, pp. 104-105, May 1998.

[47] T. Grandison and M. Sloman, "Specifying and Analysing Trust for Internet Applications," in *Proceedings of 2nd IFIP Conference on E-Commerce, E-Business, E-Government (I3E)*, 2002.

[48] OASIS Web Service Secure Exchange, *OASIS Web Service Trust Specification 1.3, Accessed on 08/15/16, http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.html,* 2007.

[49] M. G. Uddin, "Development and Automatic Monitoring of Trust-Aware Service-Based Software," Queen’s University, Kingston Ontario, Canada, 2008.

[50] C. V. D. Weth and K. Böhm, "A Unifying Framework for Behavior-based Trust Models," *On the Move to Meaningful Internet Systems,* vol. 4275, pp. 444-461, 2006.

[51] R. Sherwood, S. Lee and B. Bhattacharjee, "Cooperative peer groups in NICE," *Computer Networks,* vol. 50, pp. 523-544, March 2006.

[52] A. J. J. P. N. Beugnard and D. Watkins, "Making Components Contract Aware," *IEEE Computer,* vol. 32, no. 7, pp. 38-45, July 1999.

[53] W. Liu, "Trustworthy service selection and composition - reducing the entropy of service-oriented Web," in *INDIN '05. 2005 3rd IEEE International Conference on Industrial Informatics, 2005.*, 2005.

[54] N. Limam and R. Boutaba, "Assessing Software Service Quality and Trustworthiness at Selection Time," *IEEE Transactions on Software Engineering,* vol. 36, no. 4, pp. 559-574, july 2010.

[55] C. W. Hang and M. P. Singh, "Trustworthy Service Selection and Composition," *ACM Transactions Autonomous Adaptive Systems,* vol. 6, pp. 927-930, 2011.

[56] S. Yan, X. Zheng and D. Chen, "A User-Centric Trust and Reputation Method for Service Selection," in *Proceedings of International Symposium on Intelligence Information Processing and Trusted Computing (IPTC),* 2010.

[57] J. Li, X. Zheng, D. Chen and W. W. Song, "Trust Based Service Selection in Service Oriented Environment," *International Journal of Web Services Research,* vol. 9, no. 3, pp. 23-42, 2012.

[58] X. Su, M. Zhang and Y. Mu, "Trust-based group services selection in web-based service-oriented environments," *The World Wide Web: Internet and Web Information Systems* , vol. 19, no. 5, pp. 807-832, 2016.

[59] L. S. Gallege, D. U. Gamage, J. H. Hill and R. R. Raje, "Towards Trust-based Recommender Systems for Online Software Services," in *Proceedings of 9th Annual Cyber and Information Security Research Conference*, New York, NY, USA, 2014.

[60] 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,* New York, NY, USA, 2007.

[61] X. Zhoua, S. Wua, C. Chena, G. Chena and S. Yingb, "Real-time recommendation for microblogs," *Journal of Information Sciences (IJIS),* vol. 279, no. 20, pp. 301-325, 2014.

[62] J. Zhang, P. Votava, T. J. Lee, S. Adhikarla, I. Kulkumjon, M. Schlau, D. Natesan and R. Nemani, "A Technique of Analyzing Trust Relationships to Facilitate Scientific Service Discovery and Recommendation," in *Proceedings of the 2013 IEEE International Conference on Services Computing*, Washington, DC, USA, 2013.

[63] M. Tang, Y. Xu, J. Liu, Z. Zheng and X. F. Liu, "Trust-Aware Service Recommendation via Exploiting Social Networks," in *2013 IEEE International Conference on Services Computing (SCC),* 2013.

[64] S. Y. Lin, Y. C. Yang, C. C. Lo and K. M. Chao, "A Social Trust Based Recommendation Mechanism for Web Service Dynamic Collaboration," in *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, 2013.

[65] P. Gupta, V. Satuluri, A. Grewal, S. Gurumurthy, V. Zhabiuk, Q. Li and J. Lin, "Real-time Twitter Recommendation: Online Motif Detection in Large Dynamic Graphs," *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1379-1380, 2014.

[66] W. Zhang, H. Sun, X. Liu and X. Guo, "Temporal QoS-aware Web Service Recommendation via Non-negative Tensor Factorization," in *Proceedings of the 23rd International Conference on World Wide Web*, New York, NY, USA, 2014.

[67] L. Yao, Q. Z. Sheng, A. Segev and J. Yu, "Recommending Web Services via Combining Collaborative Filtering with Content-Based Features," in *Web Services (ICWS), 2013 IEEE 20th International Conference on*, 2013.

[68] X. Chen, Z. Zheng, Q. Yu and M. R. Lyu, "Web Service Recommendation via Exploiting Location and QoS Information," *IEEE Transactions on Parallel and Distributem Systems*, vol. 25, no. 7, pp. 1913-1924, #jul# 2014.

[69] L. S. Gallege, Design, Development and Experimentation of a Discovery Service with Multi-Level Matching, Purdue University, 2012.

[70] R. A. M. B. B. O. A. Raje and C. Burt, "A Unified Approach for the Integration of Distributed Heterogeneous Software Components," in *Proceedings of the 2001 Monterey Workshop on Engineering Automation for Software Intensive System Integration*, 2001.

[71] P. Mysore, "An Experimental Evaluation of the UniFrame Resource Discovery System," {2005}.

[72] L. S. Gallege, K. P. Pradhan and R. R. Raje, "Experiments with a Multi-level Discovery System," in *Proceedings of the series International Conference in Computing (ICC 2010)*, New Delhi, India, 2010.

[73] L. S. Gallege, A. Phadke, M. Babbar-Sebens and R. R. Raje, "Cloud Service Selection for Earth Science Domain," in *the 2nd International Conference on Recent Trends in Information Technology and Computer Science (ICRTITCS 2012)*, International Journal of Computer Applications (IJCA), 2012.

[74] Apache jUDDI, *UDDI Reference Implementation for Java (Apache jUDDI), Accessed on 08/01/16, URL: http://ws.apache.org/juddi/index.htm*, 2000.

[75] E. Al-Masri and Q. H. Mahmoud, *The QWS Dataset, Accessed on 08/01/16, URL: http://www.uoguelph.ca/~qmahmoud/qws/index.html*, 2000.

[76] A. M. Olson, R. R. Raje, B. Devaraju and L. S. Gallege, "Learning Improves Service Discovery," *Concurr. Comput. : Pract. Exper.*, 2015.

[77] R. R. Raje, S. Mukhopadhyay, S. Phatak, R. Shastri and L. S. Gallege, "Software Service Selection by Multi-level Matching and Reinforcement Learning," in *Bio-Inspired Models of Network, Information, and Computing Systems: 5th International ICST Conference, BIONETICS 2010, Boston, USA, December 1-3, 2010, Revised Selected Papers*, J. Suzuki and T. Nakano, Eds., Berlin, Heidelberg, Springer Berlin Heidelberg, 2012, pp. 310-324.

[78] Wikipedia Inc, *Dempster–Shafer theory, Accessed on 09/01/16, URL: https://en.wikipedia.org/wiki/Dempster–Shafer_theory,* 2016.

[79] J. Kohlas and P. A. Monney, "A Mathematical Theory of Hints An Approach to the Dempster-Shafer Theory of Evidence," *Lecture Notes in Economics and Mathematical Systems,* vol. 425, 1995.

[80] Wikipedia Inc, *The Concepts of Sentiment Analysis, Accessed on 09/01/16, URL: https://en.wikipedia.org/wiki/Sentiment_analysis,* 2016.

[81] B. Pang, L. Lee and S. Vaithyanathan, "Thumbs Up?: Sentiment Classification Using Machine Learning Techniques," in *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10,* Stroudsburg, PA, USA, 2002.

[82] S. Loria, *TextBlob - Python based Text Processing Tool Suit, Accessed on 08/15/16, URL:https://textblob.readthedocs.io/en/dev/,* 2016.

[83] Wikipedia Inc, *Recommender Systems, Accessed on 09/01/16, URL: https://en.wikipedia.org/wiki/Recommender_system,* 2016.

[84] Wikipedia Inc, *Term Frequency - Inverse Document Frequency, Accessed on 09/01/16, URL: https://en.wikipedia.org/wiki/Tf-idf,* 2016.

[85] Wikipedia Inc, *Big Data - Terms and Concepts, Accessed on 09/01/16, URL: https://en.wikipedia.org/wiki/Big_data,* 2016.

[86] Wikipedia Inc, *Lambda architecture is a data-processing, Accessed on 09/01/16, URL: https://en.wikipedia.org/wiki/Lambda_architecture*, 2016.

[87] Wikipedia Inc, *Data Stream Processing, Accessed on 09/01/16, URL: https://en.wikipedia.org/wiki/Stream_processing,* 2016.

[88] L. S. Gallege, "TruSSCom: Proposal for Trustworthy Service Representation, Selection and Negotiation for Integrating Software Systems," in *Proceedings of the 2013 International Conference on Systems, Programming, and Applications: Software for Humanity (SPLASH),* New York, NY, USA, 2013.

[89] L. S. Gallege, D. U. Gamage, J. H. Hill and R. R. Raje, "Trust contract of a service and its role in service selection for distributed software systems.," in *Proceedings of 8th Annual Cyber and Information Security Research Conference,* New York, NY, USA, 2013.

[90] L. S. Gallege, D. U. Gamage, J. H. Hill and R. R. Raje, "Trustworthy Service Selection Using Long-Term Monitoring of Trust Contracts," in *Proceedings of 17th IEEE International Enterprise Computing (EDOC)*, 2013.

[91] Open Comunity, *Carrot2 - an Open Source Search Results Clustering Engine, Accessed on 08/15/16, URL:http://project.carrot2.org/,* 2016.

[92] L. S. Gallege and R. R. Raje, "Towards Selecting and Recommending Online Software Services by Evaluating External Attributes," in *Proceedings of the 11th Annual Cyber and Information Security Research Conference,* New York, NY, USA, 2016.

[93] Wikipedia Inc., *Linear Regression, Accessed on 08/15/16, URL:https://en.wikipedia.org/wiki/Linear_regression,* 2016.

[94] Wikipedia Inc., *Miltivariate Linear Regression, Accessed on 08/15/16, URL:https://en.wikipedia.org/w/index.php?title=Multivariate_linear_regression,* 2016.

[95] Z. Yan and C. Prehofer, "Autonomic Trust Management for a Component-Based Software System," *IEEE Transactions on Dependable and Secure Computing,* vol. 8, no. 6, pp. 810-823, 2011.

[96] D. C. Schmidt, "Research and Development Advances in Middleware for Distributed, Real-time, and Embedded Systems," *Communications of ACM, Special Issue on Middleware,* vol. 45, pp. 43-48, 2002.

[97] S. Pearson, Trusted Computing Platforms: TCPA Technology in Context, Prentice Hall PTR, 2002.

[98] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan and K. Wallnau, *Ultra-Large-Scale Systems - The Software Challenge of the Future,* W. Pollak, Ed., 2006.

[99] M. Montaner, B. L\'{o}pez and J. L. de la Rosa, "Developing Trust in Recommender Agents," in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, New York, NY, USA, 2002.

[100] Microsoft Inc., *Microsoft Bot Marketplace, Accessed on 08/01/16, URL: https://bots.botframework.com/,* 2003.

[101] M. Mehdi, N. Bouguila and J. Bentahar, "Trustworthy Web Service Selection Using Probabilistic Models," in *2012 IEEE 19th International Conference on Web Services (ICWS)*, 2012.

[102] J. A. Konstan, "Introduction to Recommender Systems," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2008.

[103] Y. Kim and K.-G. Doh, "Quantitative Trust Management to Support QoS-Aware Service Selection in Service-Oriented Environments," in *Proceedings of ICPADS*, 2013.

[104] J. Huang and M. S. Fox, "An ontology of trust: formal semantics and transitivity," in *Proceedings of 8th international conference on electronic commerce,* New York, NY, USA, 2006.

[105] Y. Hijikata, Y. Kai and S. Nishida, "The Relation Between User Intervention and User Satisfaction for Information Recommendation," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, New York, NY, USA, 2012.

[106] R. Guha, R. Kumar, P. Raghavan and A. Tomkins, "Propagation of Trust and Distrust," in *Proceedings of 13th international conference on World Wide Web,* 2004.

[107] Google Inc., *Google Play, Accessed on 08/15/16, URL:https://play.google.com/store*, 2016.

[108] D. U. Gamage, L. S. Gallege, J. H. Hill and R. R. Raje, "A Compositional Trust Model for Predicting the Trust Value of Software System QoS Properties," in *Proceedings of CSE*, 2012.

[109] B. W. Boehm, Software Engineering Economics, Prentice Hall, 1981.

[110] Amazon Inc., *Amazon Snap Review dataset, Accessed on 08/15/16, URL:https://snap.stanford.edu/data/web-Amazon.html*, 2016.

[111] Z. M. Aljazzaf, M. Perry and M. A. M. Capretz, "Towards a unified trust framework for trust establishment and trust based service selection," in *Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference on*, 2011.

APPENDICES

Appendix A Trust in other domains [13]

This Appendix contains a discussion of trust in other domain from our survey paper [13]. Trust an important concept that has taken on different views over time. Any definition of trust is typically biased towards the problem under investigation and the context of the application domain. For example, the following brief summary contains trust perceptions in other domains, namely legal, sociology, psychology, economics and philosophy.

In the legal domain of Law and Justice, the trust is a relationship between three parties where this property is transferred by one party to be held by another party for the benefit of a third party. Hence, a trust is established by a settlor, who transfers some or all of his property to a trustee, who holds that trust property (or trust corpus) for the benefit of the beneficiaries. In the legal domain of sociology the trust is related with the position and role of entities in social systems. In the domain of psychology, trust is believing that the person whom is trusted will do what is expected. It starts small with one entity and grows to others which results in feelings of security and trust, while failure results insecurity and mistrust. In the domain of economics, the trust is perceived as the difference between actual human behavior and the behavior expected by the individual. However, in philosophy many claim that trust is more complex than a human association and they also distinguish trust and dependence by indicating that trust can be betrayed and reliance. Constructive type of trusts is not depend on the expressed intentions of all parties involved and it is employed by courts to prevent inequality or injustice.

Appendix B Physical Products Vs Software Apps [91]

The below table presents a brief comparison between physical products and two popular online software apps, which was presented in [91]. As seen from the table, it is evident that updates are frequent for apps and maybe hidden from the users. In contrast, if there are any updates to physical products, they are usually visible to the users. Also, in case of apps, the payment method is usually based on subscriptions and includes an explicit agreement between the provider and the user. Such an agreement usually transfers the responsibility is to the user ("caveat emptor"). These two aspects alone provide a basis to negatively answer the first question raised in Section 1 and make a case for exploring the second question in more details.

Comparison between physical products and online software apps.

Criteria (Example)	Mobile App (Yahoo Weather1)	Software Service (LifeLock2)	Movie DVD (RoboCop3)	Consumer Product (Rolex4)
Purpose	Weather Forecast	Identify Theft Protection	Entertainment	Wearable Timepiece
Payment Method	Free / Monthly Subscription	Monthly Subscription	One Time	Onetime
Include Software	Yes	Yes	No	No
Change over time	Periodical Updates	Periodical Updates	Then yield a sequel	Yield new product

This initial negative reaction to the first question is strengthened by following additional observations: i) inherent features of apps, many of which are time and execution-environment dependent, do not lend themselves to fit the "fixed-attribute-based descriptions" of typical consumer products – for example, most physical products have time-invariant features such as, the weight and dimensions; ii) unlike physical products, most apps are described using a programmatic interface (which could be multi-level) along with a set of quality of service (QoS) parameters. These QoS parameters are typically time and execution environment sensitive, thus, making apps to be quite different from physical products; iii) the selection of a physical product usually happens in isolation – unlike a service that may be selected and combined with other compatible apps, based on the need of the application, to generate a complex distributed software system; and iv) physical products have a well-defined lifetime (indicated by their sell by date, expiration date and warranty period), while a service is assumed to run and evolve from its initial release with continuous updates. Hence, the specification and associated opinions available about a service need to be evaluated in a time sensitive manner if any recommendations are to be provided to a user.

As a consequence of these inherent complexities, any good service selection or recommender system for the software apps domain must tackle all these complexities effectively, which the prevalent product-oriented recommendation systems do not address. These abovementioned challenges, hence, suggest a slightly different approach while recommending software apps – one which considers trust (defined in the next section) about a service to be the central feature. In this dissertation, we describe a framework called TruSStReMark which aims to provide a comprehensive service selection framework targeting online applications and apps.

VITA

VITA

Lahiru Sandakith Gallege

Full Name:	Lahiru Sandakith, Pileththuwasan Gallege
Place of Birth:	Ahangama, Galle, Sri Lanka.

Education:

December, 2016	Doctor of Philosophy, Computer Science, Purdue University (IUPUI), Indianapolis, Indiana, USA.
May, 2011	Master of Science, Computer Science, Purdue University (IUPUI), Indianapolis, Indiana, USA.
August, 2005	Bachelor of Science, Computer Science and Engineering University of Moratuwa, Sri Lanka.

Experience:

Sept 2015 - Present	Senior Data Scientist, Customer Analytics, KPMG LLP / Link Analytics LLC, Knoxville, TN, USA.
May 2014 - Aug 2014	Software Engineering Intern, Angie's List Inc, Indianapolis, USA.
May 2011 - Aug 2011	Rosen Center for Advanced Computing, Purdue University, West Lafayette, USA.
Aug 2008 - May 2015	Teaching/Research Assistant, Indiana University - Purdue University, Indianapolis (IUPUI), Indianapolis, USA.
May 2006 - Jul 2008	Senior Software Engineer, WSO2 Inc, Colombo, Sri Lanka.

Honors/Affiliations:

School of Science Teaching Assistant Award - Computer Science (IUPUI), 2015.

Gersting Award for an Outstanding Graduate Student - Computer Science (IUPUI), 2015.

Outstanding Computer Science Undergraduate Award, University of Moratuwa, Sri Lanka, 2005.

Open Source Project Contributor/Committer for the Eclipse Foundation - WTP (Web Tools Platform).

Open Source Project Committer/PMC for the Apache Software Foundation – Apache Axis2 Tools.