Senda Romdhani

Trust evaluation for stream data services based on data quality and service performance

N°d'ordre NNT : 2022LYSE3012

THÈSE de DOCTORAT DE L'UNIVERSITÉ DE LYON
Opérée au sein de
L'Université Jean Moulin Lyon 3

École Doctorale N° 512

ED en Informatique et Mathématiques (InfoMaths)

Discipline de doctorat : Informatique

Soutenue publiquement le 10/05/2022, par

Senda ROMDHANI

Trust evaluation for stream data services based on data quality and service performance

Devant le jury composé de :

Acknowledgement

This thesis is the fruit of years of hard work, dedication, consistency, and sacrifices. This journey has been very special for me as it was filled with so many challenges such as how to manage my time, how to think and research, and how to defend my ideas, etc.

Throughout this journey, my three supervisors helped me a lot by guiding me, advising me, challenging me, and much more. Together, we created a balance. For that, I would like to thank them: I thank Mrs. Chirine GHEDIRA-GUEGAN for her continuous support and for showing me how to stay focused and look at problems and challenges in a simple way. I thank Mrs. Genoveva VARGAS-SOLAR for challenging me to organize my schedule, for showing me how to research, and for redirecting me towards insightful research works. I thank Mrs. Nadia BENNANI for challenging me to enhance my ideas, and for showing me how to give attention to details that matter and how to argument my work. I also thank another member of the SUMMIT project namely Mr. Javier ESPINOSA-OVIEDO for supporting me at a technical level and guiding me through my experiments.

I would also like to thank the members of the jury who evaluated my work: Mrs. Daniela GRIGORI and Mr. Allela HADJALI for reviewing my dissertation, as well as Mr. Christophe CHASSOT, Mr. Belhassen ZOUARI, and Mrs. Sana SELLAMI for being part of my jury and their interest in my work.

I would like to thank SOC and DRIM team members for their insightful discussions, for making my journey inside the laboratory LIRIS better, and for learning me how to be part of a team.

Finally, much gratitude goes to my brothers, my parents, my best friend Khouloud, and my dear friend Lucas. They were always there for me and supported me through hard and good times even though I turn unbearable when I stress (and I do, a lot). Thank you for your understanding.

Abstract — In recent years, the number of stream data services that can capture real-time data from the physical world has been increasing tremendously. Therefore, selecting a data service that corresponds to users quality requirements and conditions given a request is challenging. Particularly because stream data services provide data under different conditions (data freshness, provenance, security, service performance etc.). Moreover, data that are accessed using these services are generally used for important (critical) decision making. Therefore, the selected services must be trustworthy. A trustworthy stream data service respects the QoS terms as promised by its provider and provides access to up-to-date data. However, services are deployed in different service environments under the black box model (black box data services). This black box model creates blind spots since services neither export (meta)-data about conditions in which they collect data, nor the quality of data they deliver.

This thesis proposes a solution for the trustworthy stream data service selection challenge. This solution consists of (1) evaluating the trustworthiness of stream data services using performance and data quality as trust factors and (2) ranking services according to their trust levels given a request. Our research focused on three issues, mainly (1) the definition of an evaluation model for data quality for stream data services, (2) considering the black box, the proposal of protocols and strategies for collecting the necessary evidence, and how they are used for this evaluation, and (3) the definition of a trust evaluation model for black box stream data services that combines performance and data quality.

To address these issues, this thesis contributes to two axes. **First**, we proposed a data quality evaluation model for stream data services focusing on data freshness. Data freshness is evaluated using two timeliness metrics including data timeliness and database timeliness. Data timeliness indicates the extent to which the captured data are up to date. Database timeliness indicates the extent to which the service's database is up-to-date. Then, we proposed $TUTOR$, a daTa qUaliTy Observability pRotocol for black box stream data services, that helps capture the necessary evidence using sampling techniques for the computation of the timeliness metrics and thus, data freshness level. As a result, services are tagged with an up-to-date data quality level. **Second**, we proposed a trust evaluation model for black box stream data services which is based simultaneously on the functional and non-functional aspects of the service. In other words, on the technical aspects of the service and the quality aspects of the delivered data. In order to define this trust evaluation model, we followed a series of steps: first, the definition of the QoS metrics used for service performance evaluation including availability, time efficiency, and task success ratio. Second, defining a way for composing data quality and service performance in order to compute data service trust. As a result, services are tagged with an up-to-date trust level. Finally, we provided a proof of concept of the proposals and validated them in the context of medical data services related to sleep apnea in the context of the SUMMIT project

Keywords: Trust, Stream data services, Performance, Data quality.

Résumé —

Ces dernières années ont été marquées par une croissance exponentielle de services de données en flux continu, issues du monde physique, ce qui a accru la difficulté de leur sélection en réponse à des requêtes complexes, en adéquation avec les attentes et les conditions qualitatives des consommateurs. En effet, lesdits services permettent l'accès et le recueil de données en temps réel souvent collectées dans différentes conditions (fraîcheur, provenance, sécurité, performance du service, etc.). En outre, les données obtenues grâce à ces services sont généralement utilisées pour prendre des décisions importantes voir critiques, exigeant de ce fait la sélection de services fiables (de confiance). Un service de flux de données est dit fiable dès lors qu'il est conforme aux conditions QoS promises par son fournisseur et donne accès à des données actualisées (à jour). Cependant, les services sont souvent fournis et déployés dans divers environnements en adoptant le modèle de la boîte noire. Ce dernier modèle crée des obstacles supplémentaires dans la mesure où ces services n'exposent ni exportent de (méta)-données sur les conditions dans lesquelles ils recueillent des données, ni sur la qualité des données fournies.

Partant de ce constat, l'objectif de la présente thèse est de proposer une solution au défi de la sélection de services de flux de données fiables. Plus précisément, étant donnée une requête utilisateur, cette solution doit permettre de (1) calculer la fiabilité des services de flux de données en utilisant leur performance et la qualité des données qu'ils fournissent et (2) classer les services en fonction de leur niveau de fiabilité.

A cette fin, nos travaux de recherche se sont focalisés sur trois problématiques complémentaires à savoir : (1) la définition d'un modèle d'évaluation de la qualité des données pour les services de flux de données, (2) compte tenu du caractère boîte noire, la proposition de protocoles et de stratégies pour la recueil des informations nécessaires pour l'évaluation, et la façon dont elles sont utilisées pour cette évaluation, et (3) la définition d'un modèle d'évaluation de la fiabilité pour les services de données de type "boîte noire" alliant performance de services et qualité des données.

En réponse à ces problématiques, nous avons proposé **dans un premier lieu** un modèle d'évaluation de la qualité des données pour les services de flux de données axé sur la fraîcheur des données. La fraîcheur des données est évaluée à l'aide de deux métriques d'actualité, y compris l'actualité des données et l'actualité de la base de données. L'actualité des données révèle à quel point les données recueillies sont à jour. L'actualité de la base de données révèle quant à elle à quel point la base de données du service est à jour. Ensuite, nous avons proposé *TUTOR*, un protocole d'observabilité de qualité de données pour les services de données boîtes noires, permettant de recueillir les preuves nécessaires à l'aide de techniques d'échantillonnage pour le calcul des métriques d'actualité et par conséquent, le niveau de fraîcheur des données. Les services sont ainsi étiquetés avec un niveau de qualité des données à jour. **Dans un second lieu**, nous avons proposé un modèle d'évaluation de la fiabilité pour les services de flux de données reposant simultanément sur les aspects fonctionnels et non fonctionnels du service. Autrement dit, sur les aspects techniques du service et les aspects qualité des données fournies. Une série d'étapes a été suivie en vue de définir ce modèle d'évaluation de la fiabilité: premièrement, la définition des métriques pour l'évaluation des performances des services, y compris la disponibilité, l'efficacité du temps et le taux de réussite des tâches. Deuxièmement, la définition d'une méthode alliant la qualité des données et la performance afin de calculer la fiabilité des services de données. Les services sont ainsi étiquetés avec un niveau de confiance à jour. Enfin, nous avons implémenté ces propositions et les avons validées en utilisant des services de données du domaine médicale portant sur l'apnée du sommeil dans le cadre du projet SUMMIT dans lequel s'inscrit la présente thèse, financé par la région Auvergne Rhône Alpes.

Mots clés : Fiabilité, services de flux de données, performance, qualité de données.

Contents

List of Figures

List of Tables

Table of acronyms

API	*Application Programming Interface*
DETECT	Data sErvice as a black box Trust Evaluation arChitecTure
IaaS	*Infrastructure as a Service*
IoT	*Internet of Things*
KPI	*Key Performance Indicator*
PaaS	*Platform as a Service*
QoD	*Quality of Data*
QoS	*Quality of Service*
REST	*REpresentational State Transfer*
SaaS	*Software as a Service*
SLA	*Service Level Agreement*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SUMMIT	*Multi-cloud multi-layered service*
TUTOR	daTa qUaliTy Observability pRotocol
UF	Update Frequency
WS	Web Service
WSN	Wireless Sensor Networks

Chapter 1

Introduction

Contents

One of the most common and significant challenges of data provisioning -retrieving and providing data to consumers- is selecting the right data according to specific requirements and quality constraints specified in users' requests. The emergence of multiple data providers available online that provide the same data has intensified this challenge.

Many data providers (e.g., the pandemic COVID-19, telemedicine, IoT, social networks) rely on tools and devices that can continuously collect data from the physical world under different conditions (e.g., production rate). These data frequently change according to their production rates [Per+04]. Thus, data have a limited validity duration. For instance, stock trading data changes in a matter of seconds, and data captured in the last minute or seconds can no longer be valid for a stock trader as he risks losing money over outdated information. These changes can happen with a static or dynamic frequency. For instance, temperature recordings are produced every hour. Still, observations about the number of steps of people do not have a predefined production rate (i.e., dynamic variable production). Moreover, data providers have developed solutions for exploring and exploiting data, namely, data services. In general, a service is a unit that gives access to hardware and software entities through API[1]s. For example, in the e-health context, services can provide data valuable for doctors to diagnose and monitor a patient's condition (e.g., physiological health indicators, apnea events during sleeping hours, daily glucose measures, dietary and training sessions telemetry).

Data services rely on back-ends and pipeline configurations to collect telemetry data, adopt storing strategies and propose interfaces to access data. They are deployed on service-based environments (web, cloud, etc.) and have associated QoS metrics (availability, response time, cost, etc.). Data services are designed using different parameters to maintain data,

[1] *Application Programming Interface*

for example, data insertion rate/frequency. The updates can occur according to a static or dynamic frequency [CGM03].

In this context, searching for the right data implies looking for data services that can potentially provide them according to specific conditions and quality requirements (freshness, availability, etc.). Generally, companies and individuals use data for (critical) decision-making in many areas, including traffic management, economy, health, and telemedicine. For example, they monitor the sensitive medical patients' conditions distantly using telemetry sent by smart devices and telemedicine services. Therefore, data services accessing these decision-making data must be trustworthy. The challenge is to evaluate the trust level of those services.

Indeed, various data services offer the same data under different quality conditions. Therefore, their trust level varies due to the conditions introduced by their deployment context. In a data service ecosystem consisting of services with different QoS and distinct data quality properties, trust is related to the degree of data quality the data source can ensure. In cases where QoS promised by a service provider is stated in Service Level Agreement (SLA). Trust is also related to the degree of SLA fulfilment service can guarantee (i.e., service performance) [Man15].

Service providers monitor services by computing QoS measures (e.g., service performance) used to make internal technical decisions related to, for example, resources allocation upon an increase of requests' rate to ensure a specific response time. However, for privacy and security reasons, service providers rarely share their service monitoring QoS measures or share highly aggregated information. Therefore, the provided SLAs are often shallow due to the lack of included information and warranties. Indeed, SLAs only provide KPIs for guiding consumers on how to measure service performance if desired. Still, consumers lack knowledge and expertise on using those KPIs for measuring performance. Indeed, research performed by *CloudQuadrants* (a collaboration between cloud experts and attorneys) on the maturity of the Cloud SLAs of twelve widely used Infrastructure-as-a-Service providers shows that SLAs are overall immature and shallow. As a result, organizations are sometimes disappointed when warranties in their SLAs turn out to be different from what they had expected[2]. Therefore, applications and consumers cannot use SLAs as the only criteria to select a service.

In addition, data services and providers rarely promise anything with regards to the quality of data (QoD) (if it is considered, how it is defined, what metrics are used to monitor it (if monitored), how they are measured in the background, etc.). Thus, it is not easy for consumers to compute factors related to data quality like freshness and completeness. Indeed, data services adopt a black-box model where services do not provide details on their data management back-end, including how data are collected updated and to which extent they are fresh. In this black-box model, data quality measures are usually calculated in the back-end or implicitly at the data collection and storage layers, if they are considered. Thus, data consumers (e.g., users and applications) do not know what to expect regarding QoD, and they have to trust the data service "blindly" (non-transparent decision making).

[2]`https://bit.ly/3ml2ie8`

Still, when selecting services, consumers must have evidence to determine whether they can trust services, and the evidence includes the current trust level of data services. Therefore, it is necessary to propose solutions for the trust evaluation of data services using QoS and QoD.

To illustrate these aspects, consider the following e-health scenario that highlights the importance of considering trust when selecting data services adopting a black-box model.

1.1 Motivation Scenario

The democratization of smart devices makes it possible to continuously collect physiological data through services and use applications to run statistics and maintain historical data about health under different conditions[3]. This trend has made it possible for physicians to monitor their patients' health through telemetry remotely. Health data sharing across the complex medical network has to be granted. Thereby, medical staff can access data through services-based applications with services hosted on service environments like the cloud, the Web, the fog or the edge.

In this context, consider Alice, a 50-year-old woman with breast cancer treated with chemotherapy. Fever chemotherapy-induced neutropenia (FN) is the most frequent, potentially lethal complication of chemotherapy in pediatric and adult patients with cancer. Therefore, cancer patients' must continuously monitor body temperature.

When Alice is at home, she uses medical devices (MD) provided by the hospital's telemedicine services to keep track of her temperature and other physiological measures (heartbeat ratio, body glucose level etc.). Data are produced at different frequencies (i.e., production rates) and sent by the MDs to be stored using telemedicine services in the hospitals' storage. Alice also uses personal devices (PD), including a connected and a regular thermometer. The frequency of the manual temperature measurements is variable, as it is determined by Alice's mood, memory to measure the temperature, whether she is alone and whether someone else comes by and encourages her to do a check-up.

With built-in software-as-a-service applications, she generates statistics and interprets her condition. According to a fixed or dynamic rate, these data are stored locally in her smartphone and on cloud storage service providers. For instance, Alice updates her captured temperature manually on her smartphone whenever she remembers or desires. However, her connected device daily updates, at the same time, the corresponding database with the captured temperature levels throughout the day.

As a result, several data services that are deployed on different environments give access to Alice's indicators under distinct quality conditions (e.g., availability, response time, security) (see Figure 1.1). Doctors can access her data at any moment, including complementary

[3]This scenario is inspired by the application context of the project SUMMIT, but given confidentiality agreements, we do not refer to the apnoea use cases addressed in the project.

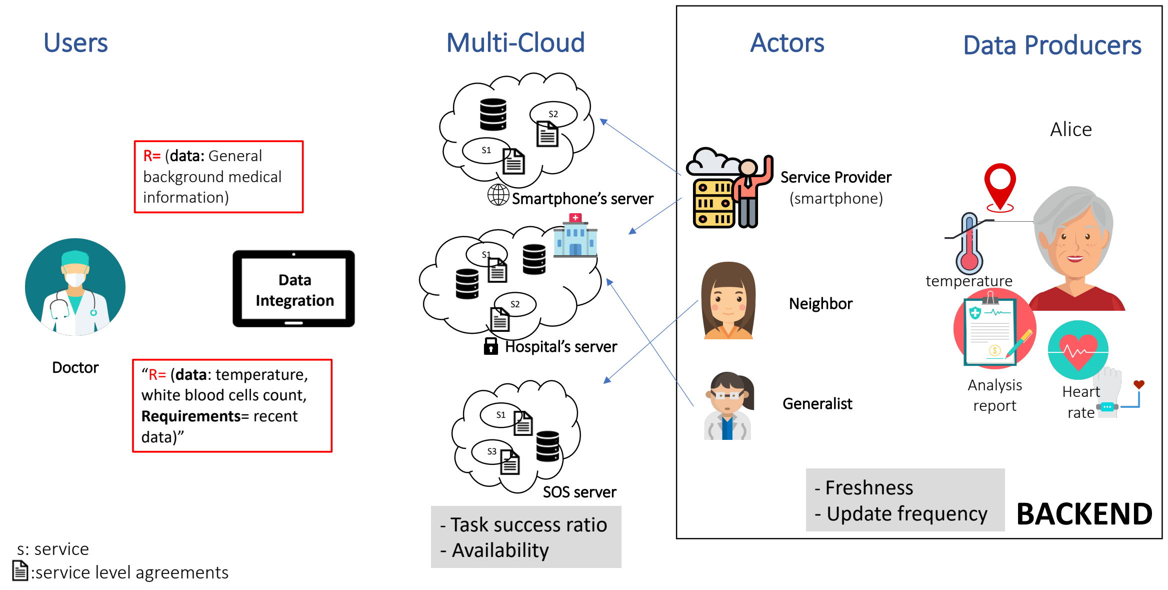


Figure 1.1: Alice data services

information such as her location, heartbeat ratio, emergency contacts, etc.

Selecting the right services to provide the required data with the best quality and conditions can be critical for doctors in an emergency. Moreover, doctors may be more interested in data quality than performance, equally interested or more interested in performance. For instance, the doctor requires instant access to a temperature reading and thus, the service with the shortest response time is the best to respond to the doctor's request. In another situation, the doctor looks up services that provide fresh observations about Alice's average heartbeat per minute, giving more importance to data quality. The doctor can, in contrast, looks up services that update temperature readings frequently and guarantee a response time in the order of milliseconds (equal importance given to data quality and performance).

Therefore, in ideal conditions, data services must be tagged with a trust level measure used as a selection criterion. The data service's trust level must be evaluated recurrently considering data quality focusing on data freshness [DBES09] and the performance of candidate data services, taking into consideration the user's quality requirement.

Monitoring and computing metrics to evaluate black-box services' trust calls for the proposal of suitable protocols and strategies to be deployed at a service broker level (i.e., an intermediary between a data service provider and consumers).

Indeed, how to measure the response time and the availability of services when the execution environment does not share details about them. It is more challenging to evaluate QoD when services do not share details about their back-end?

1.2 Problem Statement

The problem addressed in this thesis is then stated as follows:

Given a user request searching for services providing data with specific data quality and service performance requirements, how to develop a trust-based data services ranking approach in the context of services adopting a black box model?

We divided this research problem into two sub-problems.

RP1 - Evaluating data quality for black box data services.

The data quality evaluation process is highly context-dependent and can be achieved using different quality dimensions, including completeness, accuracy, freshness, etc. Our use cases' application requires the currency of data. Therefore, we are interested in evaluating data quality, focusing on data freshness. Evaluating *data freshness* can be achieved from the knowledge (i.e., evidence) about the time-related configuration of the data sources (e.g., production rate) and the conditions under which stream data services continuously collect and store data (e.g., insertion rate). Indeed, data are produced by different devices under different production rates that may be static or dynamic. For instance, a connected thermometer is programmed to capture temperature levels automatically every minute (static). In contrast, Alice's regular thermometer captures data when desired as she carries the action (dynamic). Moreover, These produced data are updated by different services in their corresponding databases under different insertion/update rates that may be static or dynamic. For instance, a service may only update its database when it has access to a WiFi connection (dynamic), while another service is programmed to update its database every $5m$ (static). However, data services are black boxes meaning they don't share such information about data configuration.

We identify two challenges for evaluating data quality provided by black-box services: (1) defining data freshness and (2) evaluating it due to the possible lack of the needed meta-data (e.g., data production rate, frequency of update of the storage of the service). To this end, we consider the following research questions:

- Which metrics to use for evaluating data freshness and how to compose them?
- What are meta-data (i.e., evidence) available to evaluate these metrics? What method to adopt to collect evidence? When unavailable, is it possible to infer/estimate evidence indirectly (e.g., database update frequency)?
- How to evaluate data freshness metrics using the collected evidence?
- Monitoring data quality by continuously collecting evidence and evaluating freshness metrics can consume resources and create overhead.

  What is the cost versus the added-value ratio of our solution?

RP2 - Evaluating the trust level of black box data services.

The explosion of stream data services introduced several challenges, including a service selection challenge. Indeed, given a request guided by quality requirements, a user has to choose among multiple candidate services offering access to the same data under different quality conditions with no guarantees about their reliability/trustworthiness. For instance, in our scenario, Alice's health indicators are produced using other devices and are accessed using several services related to her smartphone, the hospital, or even the device provider. These data services are different in terms of the quality of their accessed data (freshness etc.) and their performance (availability, time efficiency, etc.). Therefore, we propose to select services using trust as a selection criterion. The challenge is to evaluate the trust level of black box stream data services using both the technical behavior of the service (i.e., the way it deals with requests, including its QoS) and the conditions in which it provides data. The work presented in this thesis considered the following research questions:

- How to combine Qos and QoD to model the trust evaluation of black-box data services while considering the user requirements and needs?

1.3 Main Contribution & Publications

This work has been done in the context of the project SUMMIT[4] - a technology transfer project funded by the Auvergne Rhone-Alpes region - which addresses trust-based services composition on multi-cloud environments. The project focuses on data-centred requirements like data integration and service-based queries developed on multi-device environments in the medical context through services composition [Car+16]; [Ben+15]; [Car+15]; [Ben+14]. The *SUMMIT* project is based on *Rhone* [Car+16] a services composition algorithm that computes a services composition solution space guided by QoS criteria expressed by user requirements. One of the significant challenges is the selection of trustworthy data services adopting a black-box model that can be composed to suitably integrate data or respond to queries.

Therefore, we address this challenge by proposing a trust evaluation model for black box data services (see Figure 1.2). The general principle of the solution is to provide a trust-tagged registry of data services ranked according to their trust levels in response to a query. Therefore the main steps of the selection process are:

1. Upon requests, evaluate the trust level of candidate data services using their performance level and data quality level.
2. Update the service registry.
3. Rank those candidate data services (descendant) using their trust level.
4. Select the most trustworthy data service to respond to the request.

[4] *Multi-cloud multi-layered service*

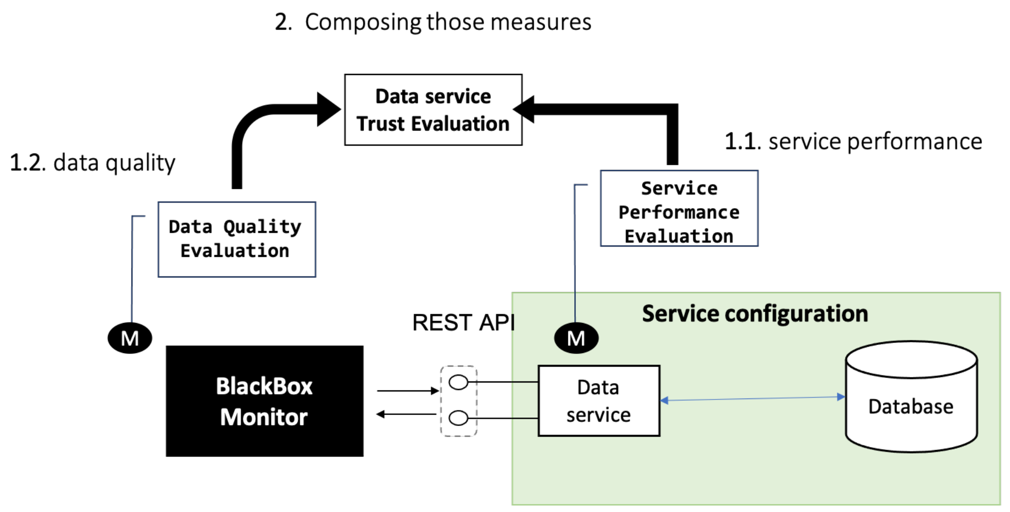


Figure 1.2: Main Contributions

1.3.1 Main Contribution

The main contribution of our work is twofold:

C1 - Data Freshness Evaluation Model & Protocol for Black Box Data Services

In the literature, most data quality evaluation models are based on quality metrics defined using available information (e.g., last modification date, number of updates). In our research context, this information is unknown due to the black-box model. Therefore, we define the trust evaluation model for data services using QoS and QoD. This contribution uses the QoD factor to define a data freshness evaluation model for black box data services (step 1.2 in figure 1.2). To compute data quality (i.e., data freshness), we propose the protocol TUTOR (daTa qUaliTy Observability pRotocol). Using TUTOR, we collect measures that serve to determine data freshness. We build knowledge using blind sampling and then apply our proposed data quality evaluation model. As a result, data services are periodically tagged with a data quality measure. Through experiments, we validate TUTOR and our data quality evaluation model and measure its performance in terms of computation time.

C2 - Trust Evaluation Model for Black Box Data Service

We propose a services trust evaluation approach that combines QoS and QoD that considers services' functional and non-functional properties. Our proposal combines technical aspects of services, including *service performance* and the quality aspects of the data represented by *data freshness*. To define this trust evaluation model, we first defined quality metrics for evaluating service performance. Second, we proposed an approach for combining the service performance and data quality factors. We validated the trust evaluation model designing the architecture DETECT (black-box Data sErvice Trust Evaluation arChitecTure) and measured its performance in terms of computation time. The originality of this approach comes from the strategy of combining QoS and QoD for evaluating services' trust. Indeed, to the extent of our knowledge, current work deals with service and data trust metrics independently to simplify the problem and abstract the specific characteristics of black-box data services.

1.3.2 Publications

These contributions are the basis for the following publications:

- S. Romdhani, N. Bennani, C. Ghedira-Guegan, and G. Vargas-Solar,"Trusted Data Integration in Service Environments: A Systematic Mapping," in Service-Oriented Computing. ICSOC 2019. Lecture Notes in Computer Science, vol 11895. Springer.
- S. Romdhani, "Towards Multi-level Trust-Driven Data Integration in Multi-cloud Environments," in Yangui S. et al. (eds) Service-Oriented Computing – ICSOC 2019 Workshops. ICSOC 2019. Lecture Notes in Computer Science, vol 12019. Springer, Cham
- S. Romdhani, G. Vargas-Solar, N. Bennani, and C. Ghedira-Guegan, "QoS-based Trust Evaluation for Data Services as a Black Box," in: International Conference on Web Services. ICWS 2021.

1.4 Outline of the thesis

This thesis is organized into four chapters (see figure 1.3).

Chapter 2 introduces background concepts related to services, service-oriented computing and service types, including data and black-box services. The chapter describes the characteristics and essential aspects of services' configuration and deployment in service-based environments. The chapter sets the concepts and vocabulary that serve as a knowledge background of both the state of the art and the proposed contributions.

Chapter 3 presents the state of the art of trust definitions and models addressing the functional perspective of information systems. It then reviews and compares existing trust-based data service selection solutions, including those evaluating data trust and service trust.

Chapter 4 presents a data freshness evaluation model for data services as a black box and an observability protocol for collecting the necessary information for this evaluation. It also validates this solution by running a set of experiments to demonstrate the effectiveness and the efficiency of our developed observability protocol and the proposed data quality evaluation model.

Chapter 5 presents the proposed trust evaluation model for black box data services combining service performance and data quality factors. The data quality factor is defined and evaluated as presented in chapter.4. The methods presented in this chapter are implemented thanks to *DETECT* (a *D*ata s*E*rvice as a black box *T*rust *E*valuation ar*C*hitecTure). *DETECT* is a prototype that provides a set of functionalities (monitoring performance, evaluating trust, ranking services according to their trust levels) to deliver trust evaluation for data services as a black-box.

Chapter 6 concludes the thesis. It summarises the contribution of the work. It assesses the results obtained through the validation of the contributions and experiments. Finally, it enumerates research perspectives addressing open issues.

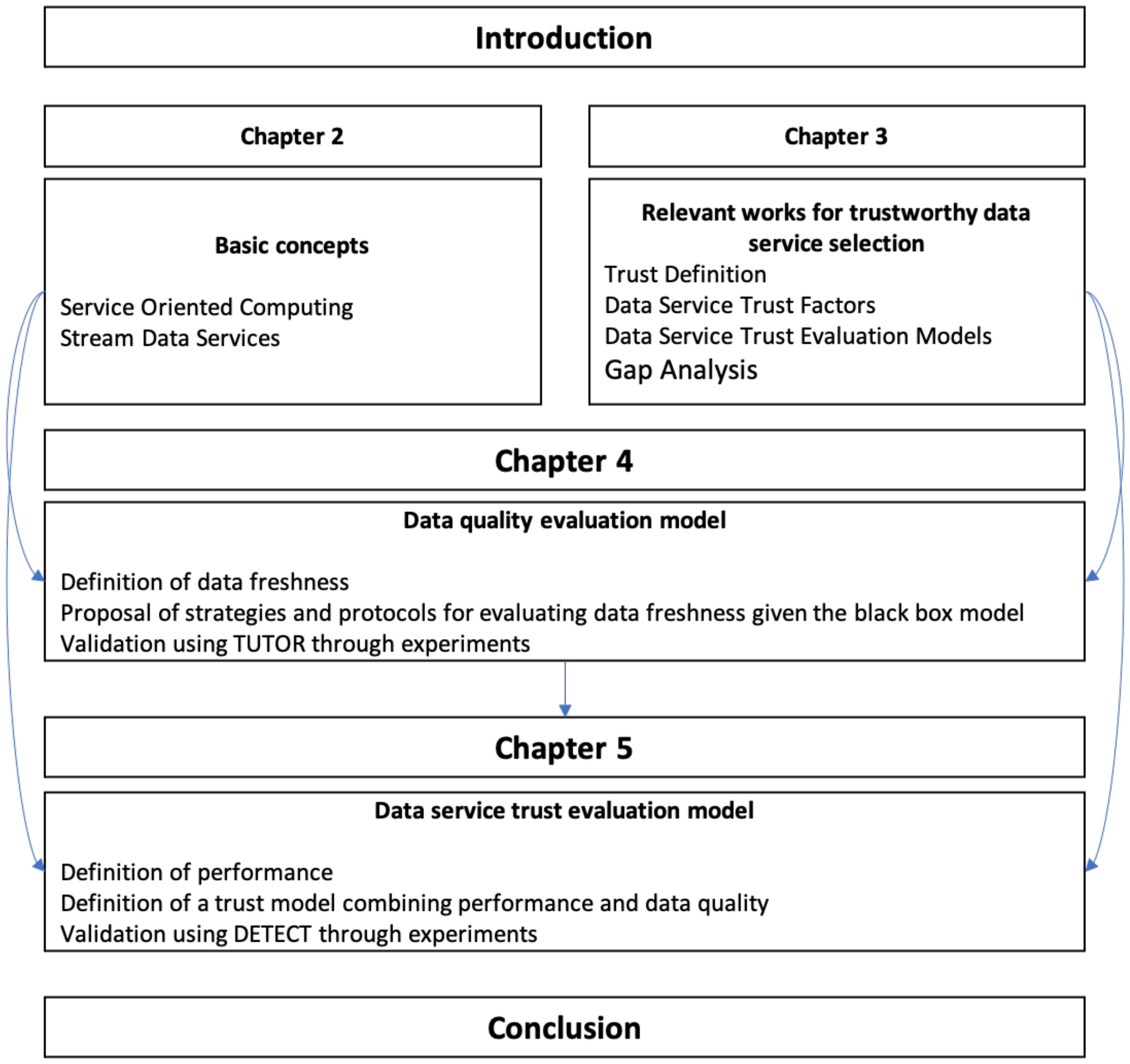

Introduction
Chapter 2
Chapter 3
Basic concepts
Service Oriented Computing
Stream Data Services
Relevant works for trustworthy data service selection
Trust Definition
Data Service Trust Factors
Data Service Trust Evaluation Models
Gap Analysis
Chapter 4
Data quality evaluation model
Definition of data freshness
Proposal of strategies and protocols for evaluating data freshness given the black box model
Validation using TUTOR through experiments
Chapter 5
Data service trust evaluation model
Definition of performance
Definition of a trust model combining performance and data quality
Validation using DETECT through experiments
Conclusion

Figure 1.3: Structure of the thesis

Chapter 2

Service based Systems: Basic Concepts

Contents

"Trust is the glue of life. It's the most essential ingredient in effective communication. It's the foundational principle that holds all relationships."

Stephen Covey

2.1 Introduction

Today, services-oriented computing provides tools for building information systems out of autonomous and self-contained units called services. The notion of service is an abstraction of a software or hardware entity that can be accessed on the network, and used as a building block for constructing more complex applications. This chapter introduces the background concepts about services-oriented computing used throughout this dissertation.

The chapter is organized as follows. we first define services in general and present an overview of service oriented computing. Second, we present stream data services and their related characteristics.

2.2 Service Oriented Computing

Service-oriented programming (SOP) is a programming paradigm that uses "services" as the unit of computer work to design and implement integrated software programs. An application consists of a user interface (UI) and a set of functionalities and components that enable consumers to perform specific tasks. It runs using an operating system OS and connects to hardware devices (e.g., memory, storage), including databases. In the past, applications (e.g., calculators, card games) were installed directly on the users' computers. With the advancement of technology, providers now have multiple possibilities to deliver applications to end-users via online solutions (e.g., web servers, web browsers). Users can access and use applications directly online. Moreover, by developing applications and systems in terms of distributed services that can be composed together (see figure 2.1), providers made applications and systems consumable not only by humans but also by other applications. In essence, services became the key element for building distributed applications, namely service-based applications. This is the major contribution given by Service-Oriented Computing (SOC) [Pap+08] to the realization of cloud computing. Indeed, SOC enables applications to be assembled with little effort into a world of cooperating services that can be loosely coupled to create flexible, dynamic business processes and agile applications. A service can be defined according to [BVS13]; [HS05] as follows.

Definition 2.1
A service is an abstraction representing a self-describing and platform-agnostic component that can perform anything from a simple function to a complex business process.

SOC introduces three main components (see figure 2.2): service provider, service consumer, and service registry -creates a link between the provider and the consumer by providing a single place where loosely coupled software components can be exposed and priced singularly, rather than entire applications. Many service architectures exist to deliver applications and enable the reuse of services (e.g., SOA, REST) that can be deployed on several service environments (cloud, web, etc.). This enables the delivery of complex processes and transactions using services while permitting the composition of applications on the fly and the reuse of services from anywhere and by anyone using several communication protocols (e.g. HTTP, SOAP). In essence, the service provider publishes a service in the service registry that the consumer can discover and use. Therefore, the registry contains the list of available services, what they offer using several description languages (e.g., WSDL) and contracts (e.g., WSLA, SLA), and how they can be accessed and used through their Application Programming Interface (API).

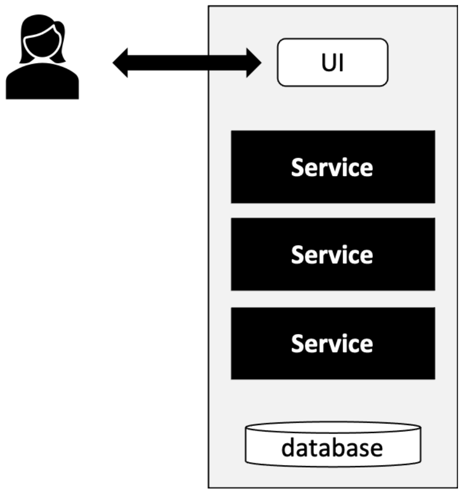


Figure 2.1: Service-based application: illustration

For instance, a service deployed using SOA on the web (WS) can be called using SOAP over HTML.

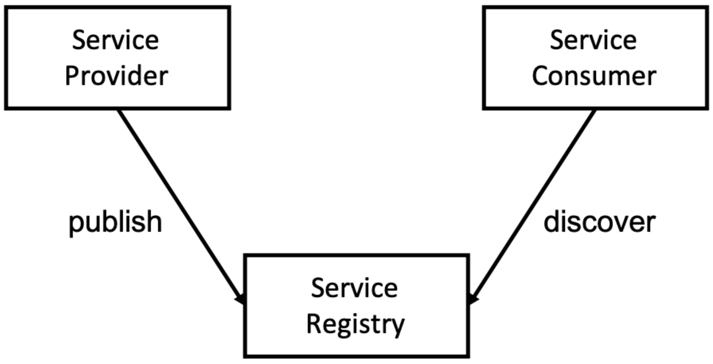


Figure 2.2: Service Oriented Computing

The remainder of this section first presents service communication protocols, service architectures and deployment environments. It then introduces different service description languages and agreements. Finally, it describes approaches for specifying service APIs.

2.2.1 Service Communication Protocols

Service communication protocols enable two or more services to communicate together and transmit data of any type and under any syntax. Generally, a given protocol defines the syntax, semantics, synchronisation communications, and possible error recovery methods. Over the years, with the advancement of technology, a multitude of service communication protocols have been proposed that aim to respond to emerging applications' needs, including UDDI (Universal Description, Discovery, and Integration), HTTP (HyperText Transfer Protocol), and SOAP (simple object access protocol) among others.

Service's stack Service-based systems are described according to a service protocol stack used to define, locate, implement, and make services interact with each other. The service protocol stack consists of four protocols:

- (Service) Transport Protocol: responsible for transporting messages between network applications and includes protocols such as HTTP, SMTP, FTP, and the more recent Blocks Extensible Exchange Protocol (BEEP).
- Messaging Protocol encodes messages in a standard pivot format (e.g., XML for Web services) so that they can be understood at either end of a network connection. Currently, this area includes such protocols as XML-RPC, WS-Addressing, and SOAP.
- (Service) Description Protocol is used for describing the public interface (API) to a specific service. The WSDL interface language for Web services is typically used for this purpose.
- (Service) Discovery Protocol centralizes services into a standard registry to publish their location and description and makes it easy to discover what services are available on the network. Universal Description Discovery and Integration (UDDI) was intended for this purpose, but it has not been widely adopted.

UDDI is an XML-based standard and a specification for a distributed registry of web services that enable organizations to discover and to interact with each other over a network of a service-oriented architecture (SOA).

HTTP stands for HyperText Transfer Protocol. It is a protocol for fetching resources on the web such as HTML files, JPEG images or even data. HTTP is a client-server protocol where Communication between clients and web servers is done by requests and responses and is based on XML language.

SOAP The simple object access protocol (SOAP) is a messaging standard defined by the World Wide Web Consortium based on XML. SOAP supports a wide range of communication protocols found across the internet, such as HTTP, SMTP and TCP. SOAP is also extensible and style-independent. The SOAP approach defines how the SOAP message is processed, the features and modules included, the communication protocol(s) supported and the construction of SOAP messages.

2.2.2 Service-based Architectures

A service-based architecture is a system and application design that emphasises services as the primary component used to implement and perform business and non-business functionality. We identify 5 major service architectures described hereafter, including monolithic, SOA, microservices, serverless and REST.

Monolithic application is a single unit and complex application that encompass several tightly coupled functions. It is designed to handle multiple related tasks and typically can not couple with other applications.

SOA (Service Oriented Architecture) is a decentralized architectural approach for designing and developing applications. SOA enables applications to use services available across different platforms over the network regardless of the languages or vendors in a loose-coupling manner. Therefore, it allows the reusability of the services in multiple applications independently without interacting with other services. Moreover, with independent services in the SOA, it is easy to scale the application and tell which service is failing when there is a problem instead of checking the whole complex code of a monolithic application.

Microservices are a variant of SOA (see figure 2.3). Microservices are both an architecture and a software development approach that decomposes applications into the simplest, most independent elements called complementary but independent services.

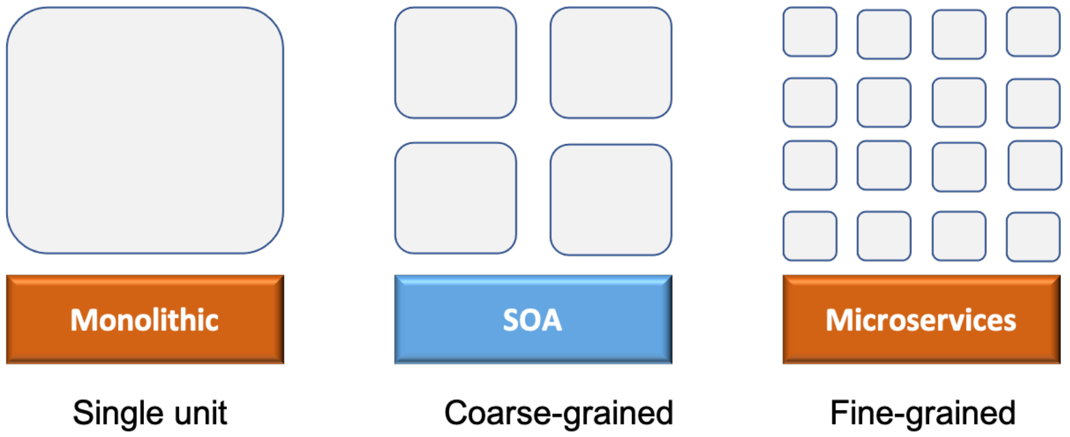


Figure 2.3: Monolithic VS SOA VS Microservices

Serverless enables developers to create and execute an application without having to manage servers that are delegated to a cloud provider.

REST is an architecture style for building hypermedia systems and applications. REST is a set of conventions and good practices to follow. It uses the term resources to abstract any information. Resources consist of data, meta-data (describing the data) and hypermedia links. For example, a REST resource can be a document or an image. REST can be combined with SOA.

The principle of REST[1] is based on a uniform interface for accessing a resource (URI). The

[1]`https://restfulapi.net`

interface should provide a way to fetch data. It promotes a client-server design with the following characteristics:

- stateless (no history or session details are saved about the made requests),
- cache-able constraint (caching of data and responses to improve performance and scalability),
- layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior.

2.2.3 Service Deployment Environments

Environments describe places where applications' codes are deployed. Generally, it is the organisation's choice or the developer to choose the service deployment environment. This choice is influenced by the service consumers' needs (e.g., scalability, performance) and the price they are willing to pay. Possible deployment environments include and are not limited to on-premise deployment, Web deployment, and Cloud computing deployment. A provider may choose one of these options or combine two or more to deliver her applications.

On premise deployment is when an organization or company manages its infrastructure where it has its servers running locally. Indeed, services are hosted locally, and data are stored inside the organization or company premises. This choice is generally made when the organization handles everything internally for security and privacy reasons. It becomes more profitable for the organization when it has fewer consumers. However, when the number of consumers is escalating along with their needs and requirements, other choices must be considered.

Web services (WS), especially those based on REST (REpresentational State Transfer), are the major technology for SOA [MMY09]. One of the most popular expressions of service orientation is represented by Web Services. WS definition is given by W3C[2] as follows:

Definition 2.2
A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network.

They are deployed on Web servers -a computer software and the underlying hardware that accepts requests via HTTP-. Therefore, WSs become platform-independent and accessible to the World Wide Web.

[2] `https://www.w3.org`

Cloud Computing Based on the definition provided by the National Institute of Standards and Technology (NIST)[MG+11], Cloud computing can be defined as follows:

Definition 2.3
Cloud computing is a model for enabling ubiquitous, convenient, on-demand based network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services). Resources can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model comprises five essential characteristics, three service models, and four deployment models.

Cloud computing offers developers and IT departments the opportunity to focus on the essential and avoid similar tasks such as provisioning, maintenance and capacity scheduling using distant servers accessed through Internet connections. As Cloud computing has grown in popularity, several deployment models and strategies have emerged to meet the specific needs of different users. Each type of cloud service (XaaS: Everything as a Service) and deployment model offers different levels of control, flexibility and management. As such, the cloud model can be considered as a highly dynamic and advancing service environment presenting several advantages and opportunities for organizations and individuals but at the same time brings a lot of challenges for IT developers.

The technology of Cloud computing entails the convergence of Grid and Cluster computing, virtualisation, Web services and Service-Oriented Architecture - it offers the potential to set IT free from the costs and complexity of its typical physical infrastructure.

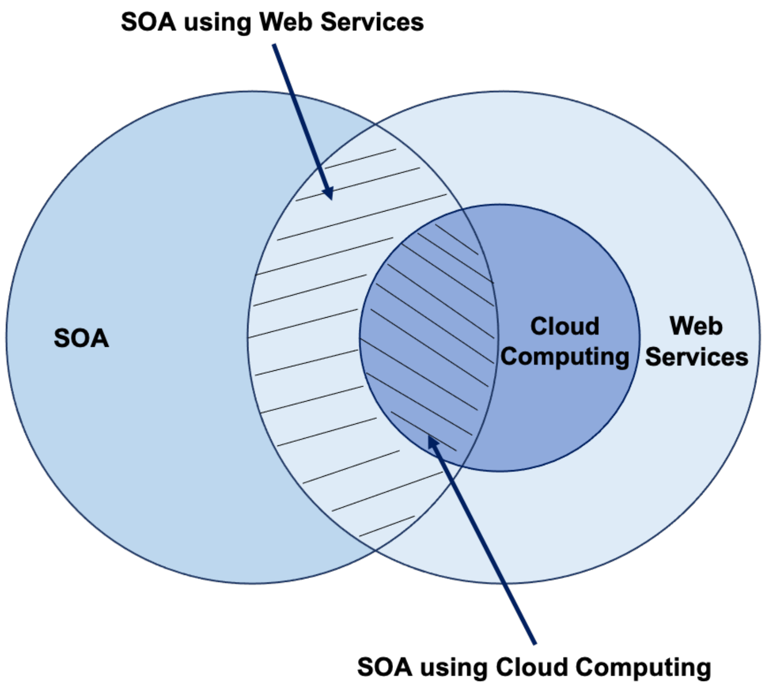


Figure 2.4: SOA, Web Services and the Cloud Computing

Most SOA-based applications are migrated to the cloud. Migrating applications and solutions to the cloud is an essential and practical step for any enterprise, especially for those who know the power of data and use it as business revenue. Cloud computing brings benefits, including scalability (usage of multiple servers to provide extra capacity) - better performance (load balancing across multiple servers) - reduced costs (may reduce the cost of managing and maintaining data application and IT systems). In this sense, SOA and Cloud computing are concurrent but complementary. In essence, Web services are deployed increasingly on the cloud using the SOA architecture. Cloud-based Web services are accessed via the Application Programming Interfaces (APIs) (see figure.2.4) and are implemented using protocols such as HTML, REST and SOAP.

On top of the basic cloud services (IaaS, PaaS and SaaS), several new types of cloud services (Xaas) have emerged since the appearance of the cloud model. Mostly to respond to new emerging technologies and consumers needs including but not limited to *MLaaS* (Machine Learning as a Service), *DIaaS* (Data Integration as a Service, *BaaS*(Blockchain as a Service), and *DaaS* (Data as a Service).

2.2.4 Service Description Languages & Agreements

Before a consumer can interact with a service, some details of the service's implementation have to be known. To do so, the consumer uses the available service description and agreements provided by the service provider, which differs according to the service architecture and deployment environment.

Definition 2.4
A service contract is an agreement between a (usually remote) service and a service consumer that specifies the input and output data along with the contract format (e.g., XML, JSON).

WSDL is a description language based on XML (Extensible Markup Language) that defines the characteristics of the service and all the methods, together with parameters, descriptions, and return type, exposed by the service. In it, a standard is used for describing web services. Indeed, WSDL is generally used by UDDI to describe the interfaces of web services. Together with SOAP, they are identified as the core components for supporting web services [Ng06].

WSLA Service level agreements have been around ever since service providers charge clients for service use[Lar98]. They are provided as an assurance for service users and are crucial for setting standards of a "good" service. WSLA (Web Service Level Agreement) [Lud+03] is an agreement that details the guarantees regarding a WS.

SLA According to authors in [ADC10b]; [Kha16], SLA can be defined as follows:

Definition 2.5

The SLA is a legal contract between a cloud service provider and a cloud service consumer documenting how services will be delivered and providing a framework for service charges.

Indeed, well-defined services and their associated quality of service (QoS) are fundamental components of any successful contract between a service provider and a service consumer, whatever the deployment environment and architecture. Service providers use this foundation to optimize their use of infrastructure to meet signed terms of services. Service consumers use the agreement to ensure the quality of service they need and maintain acceptable business models for the long-term provisioning of services.

To the best of our knowledge, the key components of any well documented (W)SLA especially in the cloud environment are:

- **Agreement overview**. This section contains the basics of the agreement including the involved individuals, the start and expiry date for service usage and the general introduction of the provided services.

- **Description of services**. This section contains details about the provided services including the description of how the services are delivered, whether maintenance/monitoring service is offered, the used technologies and applications etc.

- **Service Level Objectives (SLOs)**. This section is an agreement within the SLA defining objectively measurable conditions and performance levels like uptime, throughput or response time. SLOs are individual promises made by a service provider to the client that establish the client's expectations regarding each functional property of the service. This section also guide the consumer how SLOs are measured (KPIs).

  Example: [Service Provider] guarantees that response time should be less than 100 ms.

- **Service tracking and reporting** This section details how service performance is going to be monitored including tracking intervals, recovery plans etc. and the involved stakeholders in the agreement.

  Example: [Service Provider] will inform [Customer] regarding scheduled and unscheduled service outages due to maintenance, troubleshooting, disruptions or as otherwise necessary.

- **Exclusions**. This section should clearly stipulate the excluded offers from the agreement (if any exist).

  Example: [Service Provider] may not be able to credit reimbursement for service errors (i) caused by factors outside of [Service Provider]'s reasonable control; (ii) that resulted from Customer's software or hardware or third party software or hardware, or both.

- **Repercussions** This section details acts to be made in case of violation of SLOs by the service provider. It is generally about the definition of compensation or payment in case if a cloud provider cannot properly fulfill their SLA.

Example[3]: "If Google does not meet the SLO, and if Customer meets its obligations under this SLA, Customer will be eligible to receive the Financial Credits described below. Monthly Uptime Percentage and Financial Credit are determined on a calendar month basis per Project or, for a Single Instance, per instance. This SLA states Customer's sole and exclusive remedy for any failure by Google to meet the SLO."

2.2.5 Application Programming Interface

An API is a set of method signatures expressed using an interface definition language (IDL). An API enables the implementation of an application logic defined as method calls within a program. As shown in figure 2.5, an API specifies access points to a software entity (service, server, database or application).

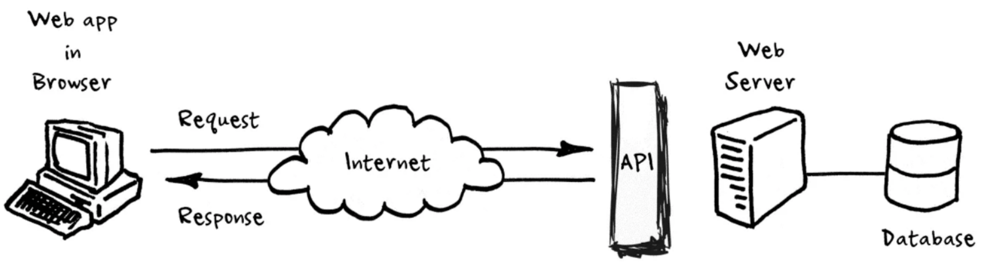


Figure 2.5: API illustration

The notion of service was the starting point to the emergence of services that give access to data (e.g., *Twitter*, *Meta*) and devices that can collect data (e.g. IoT devices, smartphones). For instance, *Meta* provides APIs that enable developers to build quick solutions that use *Meta*'s generated data when authorized. Another example is an API provided by Alice's smartphone provider that enables the hospital to build a storage solution and link it to its services.

Services' APIs can be private (accessible only to a selected group of developers), public (accessible by any outside developer), and composite (composition of services).

APIs can be further categorized into various types based on application designs and other constraints, such as Web API, HTTP API, SOAP API, REST API among others. A Web API (or Web Service) conforming to the REST architectural style is a REST API[4]. SOAP and REST APIs are presented hereafter.

SOAP API is an API that uses SOAP as a communication protocol and WSDL as a description language.

[3]https://cloud.google.com/compute/sla
[4]https://restfulapi.net

REST API is an application programming interface (API) that uses a representational state transfer (REST) architectural style. The REST architectural style uses HTTP to request access and use data. This allows for interaction with RESTful web services taking advantage of HTTP's native capabilities, such as GET, PUT, POST and DELETE. When a request is sent to a RESTful API, the response (the "representation" of the information "resource" being sought) returns in either the JSON, XML or HTML format. A web address defines a RESTful API, or Uniform Resource Identifier (URI), typically following a naming convention.

2.3 Stream Data Services

Companies are putting in place solutions and taking initiatives namely data services to access their data in order to exploit them and analyze them efficiently [HMS02].

Definition 2.6
Data service is a term for defining a service gives access to data collections.

Essentially, *data services* export APIs with CRUD (create, read, update, delete) operations hiding the backend managing (collecting, storing, updating) data. Therefore, data services often adopt black-box models with few or no meta-data describing the provided data. When data are produced continuously, services are referred to as stream data services.

Definition 2.7
A stream data service is a service that gives access to a continuous flow of data.

For instance, a stream service can collect and provide Alice's temperature that is continuously captured using a smart device.

Data services can be configured differently using their quality of service (QoS) and the quality of data they are delivering, the nature of data they are accessing (e.g. frequently-changing, long-term changing, stable), their data pipeline configuration, including the rates at which continuous data are captured and inserted into their corresponding databases.

The remainder of the section defines the QoS and data quality factors associated with data services. It describes the types of data that services can manage and the pipelines they implement to manage them. The section also characterises a black box data service.

2.3.1 Data Service Quality

Quality of Service

Quality of Service indicates the quality level a service can ensure. For instance, it shows how much the service takes to respond to a request (e.g., response time, latency). QoS is defined in various ways, and measured by different metrics. After examining existing QoS research efforts [Man15]; [AMM07]; [Men02]; [Li+18], three QoS factors are identified including capability and capacity, performance, and security and privacy (see table 2.1). Indeed, these factors described below are commonly used to evaluate the QoS level.

Performance : indicates how well the service is functioning w.r.t its promised QoS and specifications. Performance can be evaluated using different metrics, including availability, response time, data integrity, cost, etc.

Capacity/Capability (C & C) Capacity represents specific functionalities of a service (e.g., mean time to recovery, mean time to failure). Capability represents the maximum amount of resources a service can provide (e.g., the maximum number of requests, the average number of concurrent access).

Security & Privacy : Security represents the various security levels. The privacy measures include authentication, encryption of personal data etc.

C & C and Security & Privacy factors are generally expressed through Service Level Agreements (SLAs) and service contracts. They are defined by standards such as NIST[JG11], ISO/IEC[SG05], and SMI[SP12], and certifications. For performance evaluation, service providers often provide KPIs that are used to measure the level of service performance and compare it to the SLOs (Service Level Objectives) both defined in SLAs.

Factors	**Metrics**	**Evaluation**
Capability	e.g. Mean time to recovery / Mean time to failure / Failure handling (backup frequency)	It depends on the used trust evaluation approach. For instance, when using *AHP*, the evaluation is obtained by pairwise comparing to the other capacities and capabilities of other target services.
Capacity	e.g. CPU capacity, Memory size, Network bandwidth, Service throughput, Storage capacity, number of parallel sessions.	same as capabilities.
Performance	**(i) Availability**: being accessible and usable upon demand by an authorized entity	(i) Current (CPU-memory-bandwidth) utilization rate, Percentage of successful requests, Percentage of downtime/uptime.

	(ii) Time efficiency	(ii) ratio of the number of times the cloud provider is time effective to the number of service requests successfully completed.
	(iii) Data integrity	(iii) Ratio of the number of times the data integrity is preserved to the number of service requests successfully completed)
	(iv) Response time	(iv) Average.
	(v) Task success ratio or **Cost**	(v) Average.
Security & Privacy	**(i)Authentication type** (Simple password,X.509, Kerbeos), **(ii)Authorization type** (Simple password, Identity-based authorization, Role-based authorization), **(iii)Self Security competence** (Malware/Firewall protection, Intrusion detection system, the number of malicious access), **(iv)Mean time required to revoke user access (v)service reliability** (property to function correctly without failure)	Comparison to the security and privacy terms fixed through policies or performing security controls: e.g. continuity management and disaster recovery etc. (vi) Frequency of scanning of important ports etc.

Table 2.1: Service trust evaluation factors and associated metrics

Data Quality

Data Quality indicates the quality level of data accessed by a given service which implies fitting the purposes for which data will be used. Data quality is an area of research that has long been of great interest but which has taken on a crucial dimension in recent years due to the multiplicity of data sources, their heterogeneity and their increasingly accelerated scalability [BEM05]; [Ard+18]; [CFP04]. Several studies in the literature focus on the definition of metrics for the data quality dimensions' evaluation [CPB12]; [ADB18]; [Jin+18]; [LR04]; [NU16]. Several dimensions related to data quality are identified in [CZ15]; [PLW02] including *data accessibility*, *data freshness*, *data completeness*, *data interpretability*, *free-of-error* and *data security.*

Table.2.2 presents these dimensions as defined in these works along with the existing evaluation methods.

Dimensions	**Definition**	**Indicators**
Accessibility	The extent to which data are available reflecting ease of data attainability	Whether a data access interface is provided.
Freshness	The extent to which data are sufficiently up to date. It reflects how up-to-date the data is with respect to the task it's used for.	Whether data are regularly updated. Evaluated using the timeliness metrics.
Completeness	The extent to which data are not missing and is of sufficient for the task at hand	Whether the deficiency of a component will impact data accuracy and integrity
Interpretability	The extent to which data are in appropriate languages, symbols, and units and the definition are clear	Data description, classification, and coding content satisfy specification and are easy to understand. Verifying its meta-data.
Free-of-Error	The extent to which data are correct and reliable	Data provided are accurate OR Data and the data from other data sources are consistent or verifiable
Security	The extent to which access to data are restricted appropriately to maintain its security	Authentication and Authorization type AND/OR certifications: The better the authentication and authorization type, the better the data quality AND/OR Verifying certifications validity and provenance

Table 2.2: Data Quality Dimensions

2.3.2 Data Pipeline Configurations

Definition 2.8
A data pipeline defines a series of data processing steps to transfer data from a source to a target consumer.

Data nature according to its change frequency

The nature of data is essential when evaluating the freshness of data because it enables the recognition of users' expectations. Authors identify three natures of data related to their frequency of change in [Per+04] including stable data, long-term-changing data and frequently-changing data described hereafter.

Stable data Data that do not change and remain valid throughout time, such as dates related to the world war, the name of the president of the US for the year 2000, data of birth etc.

Long-term-changing data Data that change in a matter of weeks, months or years, such as the residency address of a person, the job of a person, the number of employees in an organization etc.

Frequently-changing data Data that change with a high frequency (in a matter of hours and days such as the weather) to very high frequency with intensive change (in a matter of seconds or minutes such as trading currency, stock prices etc.).

The notion of "frequency" and its interpretation (low, high, very high) is domain and context-dependent. In an e-commerce application, where the prices of the same products change once or twice a week, the frequency of change of prices is considered very high. In stock trading applications, if the stock information is updated once an hour, the frequency of change (U_f) is considered low. In our scenario, Alice's health state is critical and thus, her body temperature is expected to evolve quickly with a very high frequency of change.

Case Scenarios: Data Production & Update

When working with data services, changes are related to how often data are captured and how frequently the service updates data. Therefore, the consumed data freshness depends on the source (how often it captures data) and the update. Data collection and database updates can occur in different moments and frequencies. Therefore, there are four data collection/update pipelines' represented in figure 2.6. They lead to different freshness levels:

- **Static** production rate - **Static** insertion rate (CS1) is the most simple case scenario for a data pipeline configuration where data values are produced with a constant production rate and inserted (updated) into a database with a regular insertion (update) rate.
- **Static** production rate - **Dynamic** insertion rate (CS2): data values are captured with a regular production rate and inserted (updated) into the corresponding service's database with a dynamic insertion (update) rate. Dynamic insertion rate implies inserting data with a variable rate. In this case, devices are probably programmed to insert data on-demand. The device user performs the insertion manually, like in our e-health scenario where Alice inserts her temperature level in her smartphone manually.
- **Dynamic** production rate - **Static** insertion rate (CS3): data values are captured with a dynamic production rate and inserted (updated) with a static insertion (update) rate into the corresponding service's database.

- **Dynamic** production rate - **Dynamic** insertion rate(CS4): data values are captured with a dynamic production rate and inserted (updated) with a static insertion (update) rate into the corresponding service's database.

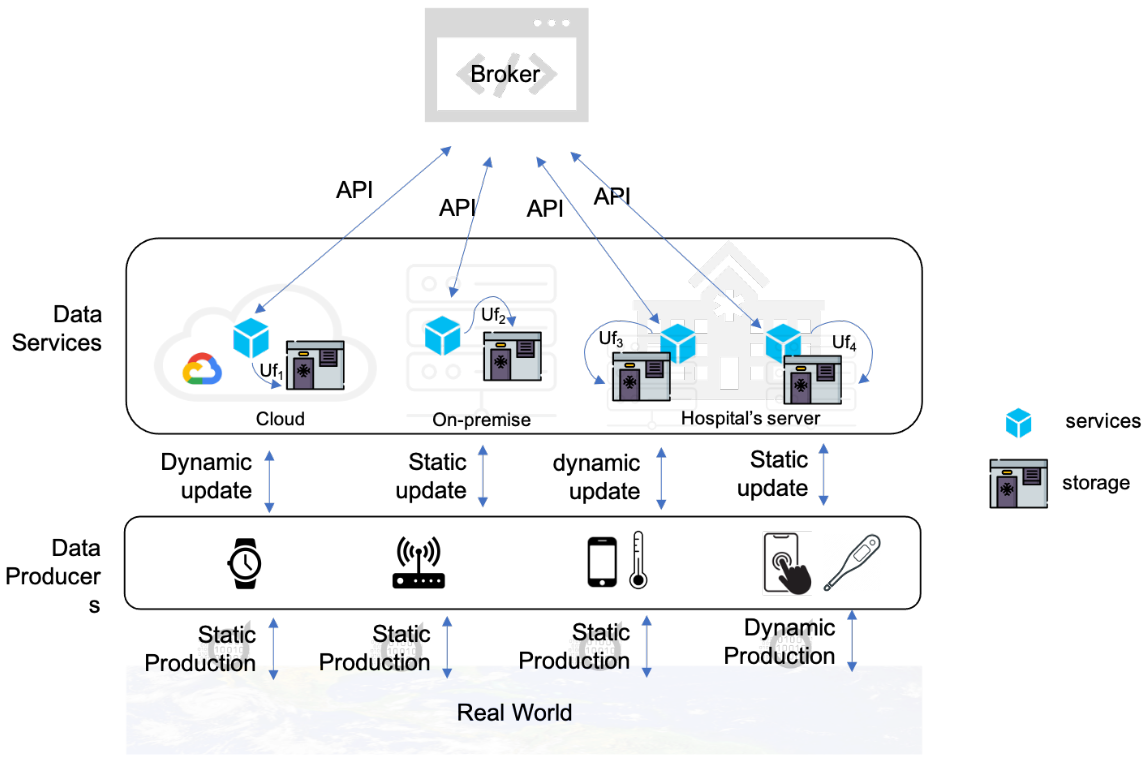


Figure 2.6: Stream Data Services: illustration of case scenarios

2.3.3 Black Box Data Services

Black Box is a model used to abstract a system or a process and can be viewed as a system only in terms of its input and output (figure 2.7). All information related to the system's functioning is hidden because it is considered complex as the user is only interested in the output.

In the service domain and the SOA architecture, a black box has become widely used to facilitate the interactions between the different services in the environment. As a result, the user can only see the input/output data and perform CRUD manipulations on the data (e.g., update data, read data). Indeed, through the usage of APIs, data services abstract the sources of data from consumers.

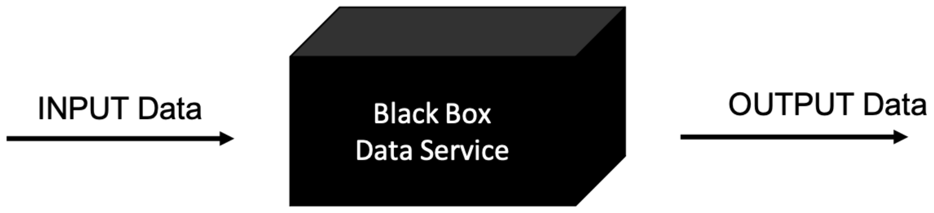


Figure 2.7: Black Box Data Service

2.4 Conclusion

This chapter defined the concepts and technologies underlying our work. It introduced service-oriented computing and the principle adopted for building service-based applications. The chapter described the building elements of the services-oriented programming like service communication protocols, API, architectures, deployment environments and SLAs. The chapter defined stream data services' quality characteristics, including QoS and data quality, the possible data configuration pipelines'. Finally, the chapter introduced the notion of black box data services.

Stream data services are services that give access to continuously produced data. They may be deployed on one or several service environments (e.g., the cloud, the web, on-premises) using several service-based architectures (e.g., SOA, REST). Web services are the most commonly used services with SOA and cloud. Depending on the environment and the architecture, many service communication protocols and description languages/agreements can ensure communication with a given service. Under the black-box model, services hide information about their deployment configurations (architecture, environment, data pipeline) and only APIs are provided. Indeed, APIs offer a unique abstract and standard interface to perform simple and direct operations with the service. Therefore, we propose a trust solution for black box stream data services regardless of deployment conditions.

The chapter 3 presents the state of the art of data and systems trust evaluation. It identifies open issues regarding the evaluation of stream data black-box services trust.

Chapter 3

State of the Art: Systems and Data Trust Evaluation Models

Contents

"Trust starts with truth and ends with truth."

Santosh Kalwar

3.1 Introduction

The definition of trust has been used to qualify and to refer to specific properties of systems, networks and data. The common aspect is that trust is defined using evidence, factors, and metrics (i.e., cues) that are combined to evaluate and judge whether an entity is trustworthy. This chapter gives an overview of trust definitions and introduces the state of the art of trust solutions including evaluation models for systems or services, and data. Indeed, it studies approaches and models to evaluate the trust level of data services.

Accordingly, the remainder of the chapter is organized as follows. Section 3.2 provides a multidisciplinary trust definition overview. Then, it discusses trust different in computing science disciplines. Section 3.3 introduces service trust and data trust factors. Section 3.4 presents trust evaluation factors models and trust evaluation techniques adopted in the literature for services and data. Section.3.5 states the problems and challenges related to selecting trustworthy data services. Finally, section 3.6 concludes the chapter summarising our study and discussing the position of our work with respect to existing work and to open issues.

3.2 Defining Trust

According to the dictionary, trust refers to a "firm belief in the reliability, truth, or ability of someone or something"; or "allow credit to (a customer)". In day-to-day life, trust guides decision-making processes [Gid91] determining the way transactions are executed and the choice of the entities or tools used for performing a specific task. For example, choosing food products according to a belief about the quality warranty and expiration date or a good reputation associated with a provider, or trusting the accuracy of a smart metering device for monitoring physiological and physical training measures and collecting telemetry data for a person wanting to control her weight and physical activity.

The definition of trust depends on the context and on the environment in which it is applied [Tam18]. In the literature, trust has been defined in sociology [Pow+11]; [Gid91]; [Bau19]; [LCRS96], psychology [Rou+98]; [DF11] (known as "social trust" or "interpersonal trust"), in economics [Feh09], workplace [Cos03], healthcare [Ber03], in computer science/systems [AG07] etc. The following sections overview trust definitions in social sciences and show how this concept has been adopted and interpreted for computing systems.

3.2.1 Trust in Social Sciences

In sociology, the authors in [Rot67] state that "one of the most significant factors in the effectiveness of our complex social organization is the willingness of one or more individuals in a social unit to trust others". Authors also introduce measures or factors for assessing interpersonal trust (i.e., trust factors), including consistency, reliability, and position in the family, socioeconomic level, religion, and religious differences between people. For instance, a person may trust more a mother to look after her child while away than a father. The more this mother proved she could take care of the child, the more she was reliable.

Trust also underlies a host to day-to-day decisions that we have to take when doing some activities [Gid91]. Indeed, humans use the trust to lower the complexity of decision-making in complex social interactions.

Bauer [Bau19] defines trust as the interaction among three elements (trustor - the one to trust -, trustee - the one who's trustworthiness to be judged -, and object - the purpose of this trust -) and involves expectations of future behavior. In [LCRS96], trust is defined as the belief in the truth - following fact or reality - of someone or something.

In psychology [Rou+98], trust is a state comprising the intention to accept vulnerability based upon positive expectations of the purposes or behavior of someone else. Trust can be defined as a process that evolves incrementally over time and includes getting in touch with other parties, forming a connection and a code of honor, and maintaining it through several actions [1].

In summary, trust is a type of relationship built over time between two parties, A a trustor and B a trustee (i.e., trust target), that applies to a specific domain of action. This relationship is affected positively or negatively by their experience as they interact over time and by particular criteria called trust factors. A party that we can trust is considered *trustworthy* and a party that we cannot trust is considered “untrustworthy”.

3.2.2 Trust in Computer Systems

Trust in computer science has been studied in various computer science *environments* [Eva89]; [May90]; [MG00] like the Web [WZM08], Service Oriented Architectures [KT09], Edge [Wan+19], Fog [ZZF18], Pervasive computing [QHC06]; [DRP17], and Cloud computing [ZKZ18]; [Ahm+18]; [Li18]. Trust has also been studied in the context of *technologies and systems* like IoT [YZV14]; [Jay+17], Blockchain [Wer18]; [HNT18], Wireless Sensor Networks [Han+14], RFID [LMF07]; [YJ05], Peer to peer networks [TE06]; [Liu+10], E-commerce [Sal+05], agent-systems [RHJ04], distributed systems [LS07], databases [PWE13].

Figure.3.1 demonstrates the levels in a computing system stack that can have an associated trust property meaning the different entities that can be considered a trust target. The stack assumes that a computing system adopts a distributed client-server functional architecture and has its associated software. It consists of three main components forming its infrastructure: the network, the storage, and computing servers (machines). These components can be deployed in remote computing environments like the fog, the edge, or the cloud. A computer system can receive user input, has the ability to process data, and the capability to create information for storage and output[2]. Different solutions and technologies implement/form a computing system: a Web server that can host a process running the application logic of a service (service server in the figure) and a client application connecting to the service server for executing target functions/operations. The client application process can run on the same or different (Web) servers. Finally, data storage servers can be running in other execution environments and collect and provide data (on-demand or continuously) using different protocols. These data are produced using hardware devices, including connected devices. All these entities can be considered as trust targets. As summarised in the following, existing work has addressed these entities' trust.

Authors in [WZM08] are interested in the trust of Web services and providers. [KT09]

[1]`https://bit.ly/3ql8W1G`
[2]`https://www.techopedia.com/definition/593/computer-system`

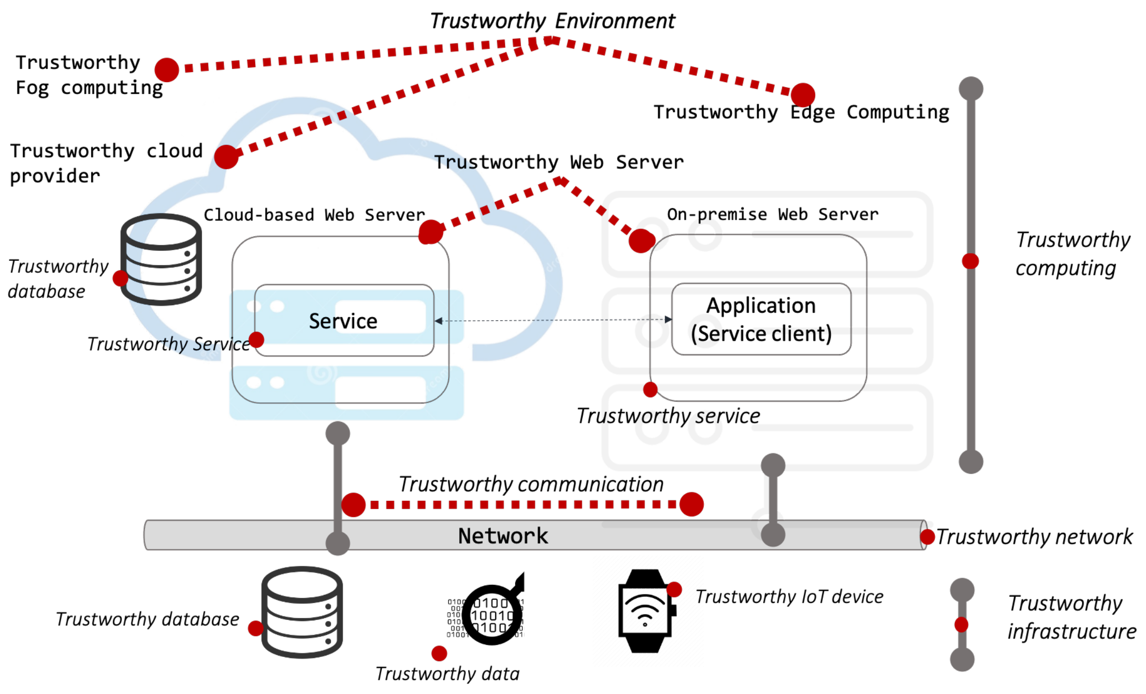


Figure 3.1: Trust stack of computing systems

studies trust among cooperating agents for every e-business transaction. The trustor is supposed to be a cognitive entity with an ability to make assessments and decisions about the received information and past experiences. Authors in [Jay+17] are interested in evaluating the trust level of data items and entities in an IoT environment. They introduce two types of trust, social and non-social. In non-social trust, the idea is to find the trust or rely on physical or cyber-entities. Social trust determines whether trust can depend on other social entities. They define four parameters; Competence, Disposition, Dependence and Fulfilment, which represent non-social trust, and three parameters: willingness, persistence and confidence. These parameters define social trust when it comes to delegation and decision making.

Furthermore, independently from the environment or technology, trust management in computer science suggests that trust can be managed differently, for example, by putting in place solutions to enhance and assure trust, evaluating trust, and making decisions based on trust.

Some solutions establish and ensure trust. According to the history of IT, trust issues trusted computing was one of the essential challenging past standards[May90]; [Trc11]. Therefore, solutions like [Li18]; [Lin+14]; [Fei98]; [BFK98]; [Bla01] establish and ensure trust for building a trusted computing environments by focusing on security and privacy measures. AT&T's PolicyMaker [Fei98], and KeyNote [BFK98]; [Bla01] propose a trust-management system to help applications answer questions of the form, “does this potentially dangerous operation conform to my security policy?". The IBM trust establishment module [Her+00] introducing a Trust Policy Language (TPL), used to define the mapping of strangers to pre-defined business roles based on certificates issued by third parties. TRUSTe [Ben99] allows companies to communicate their commitment to privacy and lets consumers know which businesses they can trust. Authors in [Han+14] discussed and analyzed security solutions to assure trust malicious attack detection, secure routing, secure data aggregation, secure localization and secure node selection. Other works propose trust schemes [Sha+08]; [YW03]; [CLC18], protocols [RS09]; [Kas+15]; [RPL17], mechanisms [RMVS05]; [Wan+20], and systems [Sha00].

Other solutions evaluate/assess trust and use the trust level measure as a benchmark to differentiate between multiple entities (trust targets) or as a ranking criterion for these entities [Wan+11]; [GGD07]; [VHA05]; [QB14]; [Som+18]; [AAT18]. Microsoft proposes the zero-trust strategy[3] for securing every digital layer from securing identifications validating device health to capturing and analyzing telemetry to understand the digital world better.

Trust can be modeled and evaluated quantitatively or qualitatively using *trust factors* like security, data management, performance. Trust factors are evaluated using *trust metrics* regarding different components of a system or service (e.g., response time, ability to deliver data). For example, [SC17] proposes a multi-dimensional trust evaluation model using several factors such as evidence, reputation, expectations, feedback, history, risk, motivation, session behaviours. The score *trust level* indicates the level of trustworthiness (i.e., trust level) of the target party: how much a service is trustworthy? Existing work has addressed the study of trust for each entity. The following sections provide a detailed analysis of approaches, models and systems that have associated these entities with trust indices. In our study, we refer to the following definitions of trust [ABT17]; [MDZ93]. First, it defines the conditions in which a trustor and a service or trustee interact (see Figure.3.2).

Definition 3.1
Trust is the belief of a trustor that a service X will behave as expected for a specified time within a specified context.

The second definition defines trust as a measure that can be evaluated when a trustor and a trustee interact:

Definition 3.2
Trust is a measurable quantity and quantifies a party A's trust (trustor) in a service X (trustee) provided by a party B.

In our work, trustees are stream data services, and trustors are data consumers that want to assess in which conditions services provide data of a certain quality. Existing work has addressed trust modelling, including the factors for measuring and evaluating trust, the techniques to monitor systems (services), data provision processes and data quality and approaches to compute, estimate or predict systems or services trustworthiness (i.e., trust level). The remainder of this chapter synthesizes existing work and discusses open issues regarding trust level assessment for stream data services.

[3]`https://www.microsoft.com/fr-fr/security/business/zero-trust`

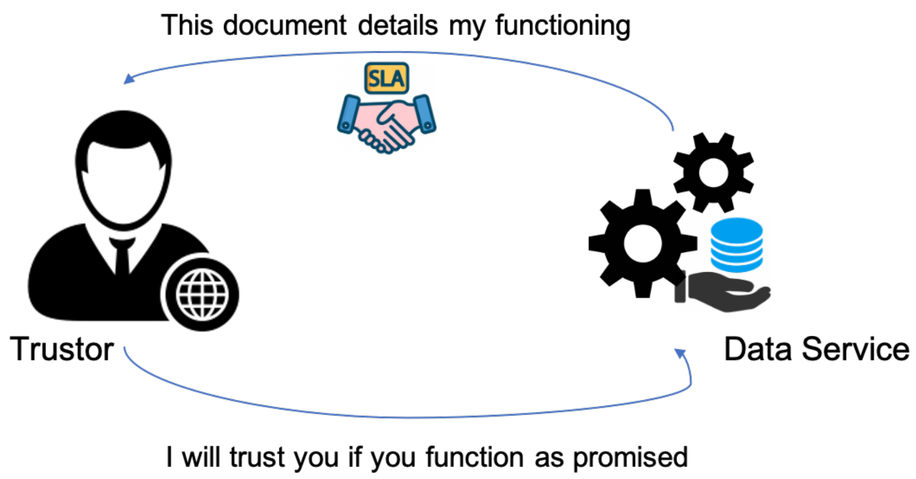

This document details my functioning
SLA
Trustor
Data Service
I will trust you if you function as promised

Figure 3.2: Trust in data services

3.3 Data and Service Trust Factors

Data and services' trust is evaluated using trust factors. A trustworthy service is a system that is believed to be capable of operating within defined levels of quality despite structural failures expected to occur in its environment of operation. Trustworthy data (streams) are considered to ensure quality properties regarding the conditions in which they are collected and stored in a database, and their content (e.g., the extent to which they are representative of the status of an environment or mini world, their freshness, their provenance, etc.).

Definition 3.3
Trust in (stream) data services is defined as the belief of a data consumer that, upon a request, a service X will provide "fresh" data representing the current state of the world (i.e., up-to-date) and will adhere to the promised QoS including availability, response time and task success ratio (see Figure. 3.2).

For instance, Alice's doctor trusts the smartphone service is performant (capable of providing timely data) and will give her up-to-date temperature levels.

In the literature, we can find works dealing with service and data trust factors independently. The following sections introduce them, and discuss the hypothesis and principles adopted by existing work for defining and measuring these factors.

3.3.1 Service Trust Factors

In general, existing work has used quality of service (QoS) metrics as significant criteria for evaluating services' trustworthiness (i.e., trust level). After examining existing service trust research results, we identify three QoS-related trust factors for services as described in chapter 2 including: *security and privacy*, *capacity/capability*, and performance. Works that have studied service trust evaluation factors are presented in table.2.1.

The performance trust factor can be modeled using metrics concerning the quantitative non-functional properties of services presented in chapter 2. Authors in [MSS17] evaluate cloud resources performance using four metrics:

1. *availability*, the percentage of job acceptance among submitted ones,
2. *reliability*, a measure of successful completion of accepted jobs,
3. *turnaround efficiency*, the promised turnaround time to the actual turnaround time, and
4. *data integrity*, percentage of jobs where data integrity is preserved. A loss of integrity is the unauthorized modification or destruction of data.

Authors in [CN16] evaluate the performance of a cloud resource using *availability*, *reliability*, and *data integrity*. They define availability for service resources and data storage within a cloud as the extent to which a system can continue to work when significant components go down (weighted sum of the percentage of accepted jobs locally and globally). Authors in [CN16] define reliability as the percentage of completed jobs among accepted ones locally and globally.

Authors in [Man15] use four trust evaluation metrics related to cloud service performance:

- *availability* measured using the percentage of accepted jobs (A) by a cloud resource per N submissions,
- *reliability* measured through the success rate of a resource computed as the percentage of completed jobs successfully per A accepted jobs,
- *data integrity* measured through the average number of preserved data by a resource per C completed jobs successfully, and finally,
- *turnaround efficiency* measured computing as the ratio of the promised turnaround time to the actual turnaround time of a resource.

Authors in [Tan+16]; [Tan+17] present a trust evaluation model for cloud providers using the performance of their services. In their work a user submits n transactions each has t tasks. To evaluate performance, they use certain metrics including *availability* measured using the average response time per successful transaction, *reliability* measured using task success ratio per transaction, and *integrity* measured through task loss rate per transaction.

Paper	Trust factors

Manuel [MSAEB09]	**Security:** Authentication type, Authorization type, Self security competence, **Performance:** Processor speed, Free RAM size, Network Parameters (bandwidth, latency)
Chakraborty et al. [CR12]	**Capacity/Capability:** CPU capacity, Memory size, Storage capacity, Number of parallel sessions, Failure handling (backup frequency, mean time to recovery), Average throughput
Xiaoyong et al. [Li+15]	**Capacity/Capability:** CPU frequency, memory size, hard disk capacity and network bandwidth/ **Performance:** Availability, Average response time, average task success ratio, and the number of malicious access.
Mrabet et al. [MSS17]	**Performance:** Availability (% accepted requests), Reliability (% requests successfully completed), Time efficiency, Data integrity
Liao et al. [LLL16]	**Performance:** Generic/ **Certifications**
Chiregi et al. [CN16]	**Capability:** Processor speed, Memory speed, and Network (latency and bandwidth) / **Performance:** Availability, Reliability (task success ratio), data integrity (includes privacy and data accuracy) / **Security** identity (it is the weighted sum of Authorization level, Security level, Entity Protection level and Recovery level)
Singh et al. [SS17b]	**Performance:** Data processing accuracy, Data privacy, Data storage success, data transmission (% of success), and data security. Availability, Reliability, Turnaround time and service use factor of the service (related to the number of users that uses the service)
Saxena et al. [SD18]	**Security**: standards, Guidelines and Certifications
Bao et al. [Bao17a]	**Security**: Prevention of unauthorized access/ **Performance:** Reliability (average response time, average task success ratio), Availability
Xiaoyong et al. [Li+18]	**Security:** Authentication type, Authorization type, Self-security competence / **Performance:** Availability, Average response time, Average task success ratio
Al Masri et al. [AMM07]	/ **Performance:** Availability, Response time, Throughput, Accessibility (probability a system is operating normally), Interoperability analysis (compliance with standards: % of errors and warnings reported) / **Capacity/Capability:** Cost /

Table 3.1: Existing work defining service trust factors' metrics

3.3.2 Data Trust Factors

The data trust factor can be modeled using measures including *data similarity*, *data provenance*, and *data quality*.

- ***Similarity***: the principle is to determine to which extent data values referring to the same event are similar. The more similar data values referring to the same event, the more likely the data are correct and trustworthy.
- ***Provenance***: provides the origins of the data and its historical record -how it arrived at the present state -: if any changes are applied on the data from its production by a given intermediary entity different from the origin, it is added to the historical record. Data with good Provenance are more likely to be trustworthy.
- ***Data quality***: indicates to which extent the data provided by systems are of good quality. Data with good quality are more likely to be trustworthy. [ZWJ17]; [Gol+18].

  Data freshness is one of the essential data quality dimensions for database systems and applications. It indicates to which extent data are up-to-date and correspond to the requirements of a given task. It is evaluated using the timeliness sub-dimension (see table 2.2).

  Data freshness evaluation solutions can be regrouped into two categories including *data freshness ensurance* solutions and *data freshness evaluation* solutions.

  Data Freshness Ensurance Ensuring Data Freshness delivery to the consumer, addressing information update scheduling issues (e.g., schedule web pages update w.r.t its related databases updates) or finding a trade-off between preserving data freshness and reducing refreshment cost of web pages or databases[RB19]; [Sin07]; [APT20]. For example, authors in [BR02] propose a solution to keep cached data up-to-date. They use user profiles to determine whether the user requires fresh data. If a user needs fresh data, it is necessary to download a new object from a remote server. If the user does not require fresh data it is possible to deliver a cached object.

  Data Freshness Evaluation is based on strategies to evaluate the data freshness level. Data freshness evaluation is influenced by the delay implied by transferring the data from a source to a destination. The evaluation of this delay varies according to the type of the used network (data pipeline) including Blockchain [Kim+20], sensor networks, wireless communication network [Zho+20], data integration systems[Per+04], Cyber-physical systems [LLG+19], the web[LR03]; [ZYS18] etc.

  Authors in [BP85] measure data freshness through the timeliness dimension computed through *Min or Max operation.* Therefore, data timeliness is measured as the maximum of one of two terms: 0 and one minus the ratio of currency to volatility. Here, the currency is defined as the age plus the delivery time minus the input time. Volatility refers to the length of time data remains valid; delivery time refers to when data is delivered to the user; input time refers to when the system receives data. Age refers to the period between time t and when the system first received data.

In [NU16]; [Per+04], authors propose a framework for data freshness evaluation in a data integration system (DIS). This framework assesses data freshness value changes in DIS for data moving from data sources to users. Besides, it evaluates the evolution of data freshness using graphs representing the integration workflows in a controlled environment: authors suppose they know the system, all provenance information and data freshness information are available to the data freshness evaluation system.

Neumaeir et al. in [Jin+18] evaluate data freshness for open data portals resources using a resource frequency of change. Data freshness requires the resource update frequency as absent evidence and thus, needs to be estimated. Indeed, authors consider the lack of information about a resource update frequency and changes' history. In their approach, first, they learn the change history of a data source using sampling techniques (static sampling) and then apply a heuristic to estimate how up-to-date a data resource is (see figure.3.3). The freshness of a portal is defined as the average freshness of all data resources in a portal.

In [NU16]; [Per+04], authors define data freshness using two sub-dimensions including the *currency factor* and the *timeliness factor*. *Data currency* captures the gap between the extraction of data from the sources and its delivery to the users. *Timeliness factor* captures how often data changes or how often new data are created in a source. It can be computed through a timeliness metric that measures the time elapsed from the last update to a source. This paper also discusses the different data types (frequently changing data, long-term changing data) and data integration system types and accordingly advise the best data freshness evaluation definition.

Table 3.2 presents the adopted data trust factors by some of the existing data trust evaluation solutions. It also explains the targeted data type (section 2.3.2) for works evaluating data freshness. Authors in [Jay+17] present a data trust evaluation framework for IoT. The trust framework combines direct and indirect evaluation, and supports multiple trust factors including reputation, recommendation, feedback, and others such as temporal factors (timeliness), competence, etc. Data timeliness is evaluated in a controlled environment and is defined as the difference between the last update to the current one. Authors in [PSB20] address the issue of trust in streams. They use accuracy, which refers to the correctness of data measurements as a trust evaluation factor, and they assume that data arrive on time, and are complete.

Paper	Trust factors	Data type
Saad et al. [SAJM13]	**Data provenance**.	
Zawoad et al. [ZHI18]	**Data provenance**.	
Peralta et al. [NU16]; [Per+04]	**Data quality - freshness:** Data currency, Data timeliness	All
Neumaeir et al. [Jin+18]	**Data quality - freshness:** Data source update frequency.	Long-term changing data

Dai et al. [Dai+08]; [Dai+09]	**Data provenance**, **Data similarity**.	
Bertino et al. [BL10]; [LMB10]	**Data provenance**, **Data similarity**.	
Peng et al. [PSB20]	**Data quality - correctness**	

Table 3.2: Existing work defining data trust factors' metrics

The presented trust factors are measured using cues and evidence presented in the next section.

3.3.3 Trust Evidence

Each trust process requires collecting the necessary information/ meta-data, which we call *trust evidence* for the establishment and the evaluation of trust factors, and, thus, the trust level. Concerning (stream) data services, evidence must be collected to evaluate service performance and data quality focusing on data freshness. We need to assess the service time efficiency, availability, and task success ratio for performance evaluation. Thus, performance evidence consists of providing proof on whether or not data service is available at a given instant, whether the service can deliver data successfully, and the response time measures of the service in response to requests. For data freshness evaluation, we need to evaluate the data timeliness related to the frequency of production of data and the database timeliness associated with the frequency of update of the database. Therefore, evidence for data timeliness evaluation consists of providing data production timestamps or data production rates. Evidence for database timeliness evaluation consists of providing the database update frequency or the timestamped-history of updates.

The absence or lack of evidence could result in a poor evaluation/judgment about the trustworthiness of a given trust target. Therefore, we believe that in order to perform a credible data services' trust evaluation. The trust evidence should have the following four properties including *availability*, *accessibility*, *continuity*, the *update.*

- ***Availability*** This property indicates whether the trust evidence are available (i.e., exist). Evidence is only available for the providers (i.e., provider side). From the user side, having the black-box model, some trust evidence are hardly available such as database update frequency. For instance, SLAs are available for both user and provider. Information about the data source is only available for exclusive use by the provider.
- ***Accessibility*** This property indicates whether the used trust evidence is accessible by the data consumer: Whether an interface is provided to access that information, and shows how much evidence are available. For instance, if a service consumer is interested in having a follow up on the performance of the used services, *Accessibility* determines whether access is provided for extracting observations about service performance.

Also, information about how often a data service refreshes its database or the adopted *CS* (section 2.3.2) may exist but remain exclusive for the data provider, and no interface is provided to access this information for other parties due to privacy or security concerns.

- ***Continuity*** This property indicates whether the used trust evidence is one-time provided or can be continuously pulled/provided for the trust evaluation.
- ***Update*** This property indicates whether the used trust evidence is up-to-date (represent the current state of the trust target) for the trust/trust factor evaluation. The trustworthiness level of an entity/target or the level of any trust factor (e.g., performance, data freshness) is not a constant measure, and it evolves throughout time. With experiences: it either increases or decreases. Thus, a measured trust evidence level at a time instant t loses its validity slowly throughout time. For instance, *Update* indicates whether observations about service performance are periodically measured.

Table 3.3 summarizes the characteristics of trust evidence in related works including performance evidence and data freshness evidence described hereafter.

Performance evidence. There are works [Li+18]; [HHH14]; [KL03]; [Ced+14] that propose solutions for the absence of evidence (*Accessibility*) for the evaluation of service performance through continuous monitoring (*Continuity*), and the deployment of agents for sensing and making observations, especially the response time, and availability of services, etc. This enables these solutions to keep the performance level up-to-date (*Update*). The monitoring is performed with the help of KPIs that are available in the services' SLA (*availability*). [CR12] proposes an SLA-based framework for evaluating the trustworthiness of a cloud. They identify several parameters (trust evidence) that can be extracted from SLA (*Availability*) or retrieved during sessions (*Update*). [Tan+17] evaluates the fulfilment of SLA per transaction, and updates the performance level as the average of all transactions. Authors in [Zhe+12] predict the ranking of services according to their performance metrics collected from previous experiences of similar users. For every transaction, they compute the similarity with other users w.r.t their preferences, pull performance measures related to similar users, and predict a ranking of services using those measures.

Data freshness evidence. Only one work [Jin+18] studied data freshness and its definition for open web portals with the assumption that trust evidence is unavailable/incomplete (*Availability*) and that resources follow a Poisson process. They collect evidence on whether a database has been detected since the last access. Open data portal indicates the frequency of changing open data sources (weekly, hourly, biannual, irregular, unknown). Data freshness is evaluated using only database update frequency, computed using Poisson estimator. They discuss the different methods to adopt w.r.t the level of accessibility. (*Accessibility*) of web portals (how much meta-data is available and can be used to infer information about the resource change history). They propose to continuously pull evidence in a static manner for this evaluation (*Continuity* (see figure 3.3), *Update*).

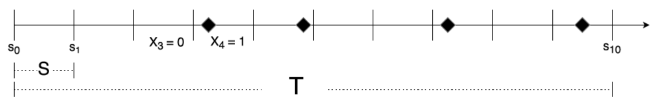


Figure 3.3: Example: sampling method [Jin+18]

However, web resources concern long-term changing data and follow a Poisson process. Thus, the Poisson estimator can assess data freshness. For telemetry, mainly stream data services, the distribution model of updates remains unknown. This distribution can be determined through experiments and deeper investigation that we did not tackle for time constraints. Moreover, as argued by the author in [Per06]; [Per+04], frequently-changing data require more than the change frequency to evaluate its freshness, and the definition of the timeliness metrics need to be adapted to the context. Therefore, the evaluation of data freshness requires redefining the timeliness metrics differently to adapt them to the type of data we are targeting (frequently changing). In essence, we want to check if the database is refreshed/updated as often as necessary by the related data service, and whether this update concerns mainly new data. We assume that data inserted into a database may be out-of-date at the time of insertion (it does not represent the actual state of the world). As mentioned in our scenario, devices continuously capture data using different production rates. They are inserted into the various services' databases using different update/insertion rates, which are not guaranteed to be static. Also, we continuously evaluate data freshness using data timeliness and database timeliness to assume that evidence is unavailable and only production timestamps can be collected from meta-data. The timeliness metrics values are continuously updated.

	Service Performance			
Paper	*Availability*	*Accessibility*	*Continuity*	*Update*
Tan et al. [Tan+17]	SLA	SLA	per transaction	per transaction
Xi et al. [Li+18]	SLA	absence	monitoring	yes
Zheng et al. [Zhe+12]	past experiences	past experiences	per transaction	prediction
Manuel [Man15]	SLA	absence	per transaction	per transaction
	Data Freshness			
Paper	*Availability*	*Accessibility*	*Continuity*	*Update*
Neumaeir et al. [Jin+18]	existence of update	partly meta-data	yes	estimation

Table 3.3: Trust Factors Evidence: Related Work

3.4 Trust Evaluation

According to the literature, trust evaluation solutions can be classified into two categories as depicted in figure 3.5: *Trust establishment* for ensuring trust between the concerned parties and *Trust level evaluation* for evaluating and monitoring trust before or/and during service usage.

Trust Establishment & Ensurance Trust establishment and ensurance consists of providing solutions and protocols to gain *initial trust* (reputation) and ensure security and privacy requirements. Solutions consist of setting up policies/schemes [Li+14b]; [Bao17b]; [Pat+18], providing certifications [Lin+16], proposing an ontology where the different involved parties can semantically describe their trust policies [Gal06], signing, and designing a service level agreement[ADC10a]; [Ser+16], putting in place a privacy-preserving solutions, security (e.g., cryptography) and access-control mechanisms [Lin+14]; [FG06]; [NV11]; [LP09] etc.

- *Policies*: establish trust among the involved parties and provide transparency and efficiency. These policies specify permissions and minimum QoS thresholds that must be satisfied to access the provider's services.
- *Certifications and standards*: ensure the compliance of security and privacy standards. A certified service should have more chances to give a better quality of service.
- *Service Level Agreement* (SLA): serves as a starting trust agreement between the service provider and the service consumer. By signing a contract that stipulates the agreed terms, conditions, and procedures to follow in case of a concern, a problem, or a violation of the agreed terms, the service provider assures the service consumer and gains his initial trust.

Trust Evaluation aims to determine how well a service or system performs when handling users' requests.

Trust evaluation processes can be categorized according to their evaluation technique (Direct or indirect evaluation), trust evaluation approach (ML, FA, Prob, MCDM, MC), and trust evaluation factors (C&C, Perf, S&P) (see figure.3.4).

Solutions for evaluating services' or systems' trust levels can be categorized into three types: subjective, objective, and hybrid.

- **Subjective Evaluation** : the trustor decides to use the opinion of the user about a service [QWO13]; [HRM11]; [CN16]; [VHA05]; [Noo+13]; [KT09] expressed as:
  - *User preferences*: in this approach, users' preferences and requirements are subjectively mapped (i.e., using subjective solutions) to the different capabilities of

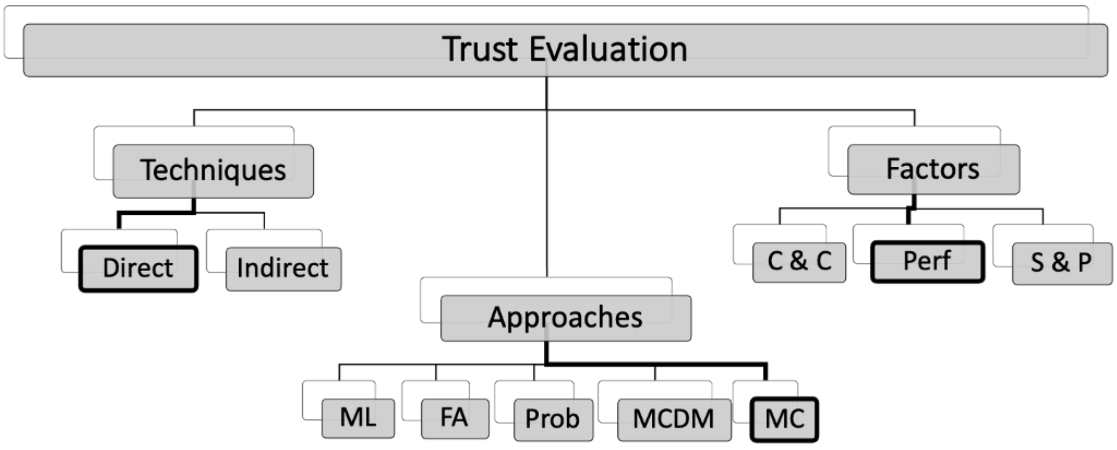


Figure 3.4: Categorization of Trust Solutions

offered services in their provided SLA. In this sense, the offer that best suits the user is the most trustworthy and, thus, selected.

- *Feedback analysis*: This approach consists of collecting users' satisfaction and opinions about previous experiences with the services to manage trust levels. Users' feedback may be delivered as a quantitative opinion or as a qualitative opinion in textual form. Note that users' subjective feedback may be influenced by several factors and may exhibit significant variations in evaluating the same service from one user to another. Besides, there is no guarantee of users' honesty. Therefore, the challenge lies in assessing the credibility of these feedback by considering only the reliable ones in the trust evaluation process of services to provide reliable trust scores.

• **Objective Evaluation** : the trustor prefers to use measurable and quantifiable information for the trust evaluation rather than subjective opinions and ratings [Man15]; [Bao17b]; [GGD07]. This can be done through monitoring and auditing [JM16], or assessing [Alh11] capabilities

- *Monitoring and auditing*: These solutions are also referred to as *QoS-based trust evaluation* or *SLA-based trust evaluation* given that the trust evaluation is based of information existing in the SLAs [Man15]. There are two kinds of information present in the SLA, including quantitative information and qualitative information. Quantitative information includes measurable factors such as response time, task success ratio, and availability. Qualitative information includes immeasurable factors such as security. Trust evaluation is performed before or during service usage. In the first case, before selecting services, data collected from previous experiences are integrated and evaluated to measure the degree of SLA fulfilment. If the SLA is satisfied, then the service is considered trustworthy and vice versa. The service that best satisfied SLA is selected. In the second case, SLA is monitored during service usage to detect violations. In case of violation, the service is either penalized or abandoned, and replaced by another. Therefore, trust evaluation provides more flexibility when it comes to service selection.

Service providers rarely allow their customers to trace the performance of the used services using an internal monitoring service (i.e., white box monitoring). Instead, customers would deploy and operate a monitoring solution they bought or developed (i.e., black-box monitoring) to distantly observe the services' KPIs.

– *Capabilities assessment*: in this approach, the different services' capabilities presented in their SLAs are objectively evaluated. For instance, pairwise comparing services using the information (i.e., evidence) in their SLA.

Trust evaluation solutions require a trust evaluation technique that determines the nature of the evaluation (direct, indirect) and a trust evaluation approach that determines how to combine trust factors.

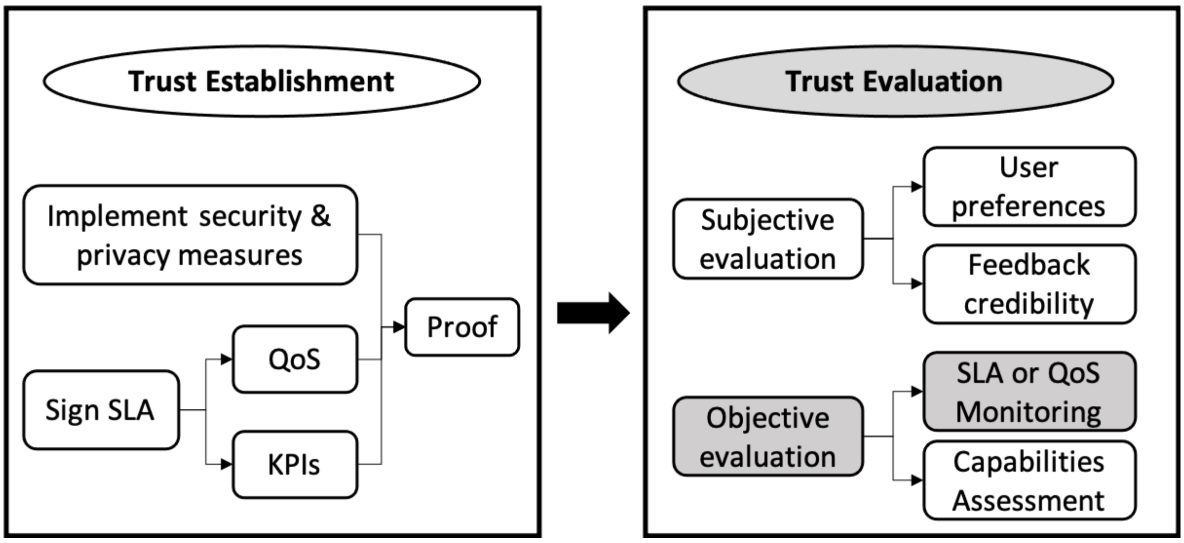


Figure 3.5: Trust Evaluation Solutions

3.4.1 Trust Factors' Evaluation Models

We classify trust evaluation models into 5 categories: *fuzzy approach* (*FA*) [MRR15]; [Jai+16]; [Zha+15]; [Li+14a], *probabilistic approach (Prob)* [JHS14], machine learning models (*ML*) [WZ16]; [CB19]; [Mao+17]; [Jay+18], multi-criteria decision-making models (*MCDM*) [ABT17]; [SS17a]; [Li+14a] and *classical mathematical models* (*CM*).

Fuzzy approach In this approach the trust evaluation is based on fuzzy mathematics including the *fuzzy set theory*[Zim10]; [QB14] and the *cloud model*[LLG09]; [Wan+08]; [Li+14a]. This method shows a quantifiable score of the uncertainty in the trust measurements, especially the subjective evaluation. It aims to map the trust evaluation factors to a set of measurements categories, a.k.a, fuzzy sets (bad, good, excellent) using fuzzy rules. Fuzzy rules are the steps used to map a given input value to a fuzzified output value: (1) linguistic fuzzy rules (2) if-then-else rules. For instance, a given score 6 for the service performance can be mapped to the fuzzy set ("bad", "good", "excellent") with a membership score of (*0,1*, *0,8*, *0,2*), while a score 2 can be mapped to the fuzzy set ("bad", "good", "excellent") with a membership score of (*0,7*, *0,3*, *0*).

Probabilistic model In this approach, the trust is evaluated using the theory of probability (e.g., Bayesian model), which quantifies uncertainty and randomness, and is not deterministic. Generally, authors consider the fact that we cannot know everything about a service to evaluate its trustworthiness. Thus, they use this method to approximate the trust level (i.e., guessing) using different factors.

ML model In this approach, authors apply machine learning algorithms to evaluate a given service's trustworthiness or QoS level, including neural networks and the naive Bayes model. Those algorithms are based on probability theory, and the objective is to predict these levels based on evidence and continuously learn from past experiences.

MCDM models This approach is used when the decision maker (trustor) faces a decision-making problem where (s)he needs to choose between multiple services based on some set of trustworthiness factors. It includes the usage of techniques such as the *Analytic Hierarchy Process* (AHP) [Li+14a] and the *Technique for Order of Preference by Similarity to Ideal Solution* (TOPSIS). *AHP* is based on weighted aggregation, and its criteria and alternatives weights are obtained by pair-wise comparison and user preferences. It starts with ordering the objective hierarchically, evaluating factors/criteria and lastly, the other options. This method consists of comparing two alternatives at a time. *TOPSIS* is based on distance. It is assumed that there is an ideal and non-ideal choice. The method aims to find the shortest distance to the positive ideal choice and the farthest distance to the ideal negative choice. It is based on Linear Programming.

CM models This is the most straightforward trust evaluation approach where authors opt for using the most basic mathematical models such as weighted-sum[Man15], ratio or multiplication for modeling the trust evaluation of services using the chosen trust factors. However, weighted-sum appears to be the most used approach for trust modeling as the trustor can determine the importance given to each trust factor. For instance, ff the trust is modeled using service security and capabilities, the trustor can choose to prioritise security and, thus, higher weight.

3.4.2 Trust Evaluation Techniques

Trust evaluation techniques have been proposed in the database, service and web domains. They can be classified into two categories: *Direct Evaluation* and *Indirect Evaluation* as pointed out by [Noo+13].

Direct Evaluation consists of evaluating the trust level of a *service* or *system* from the perspective of the user experience of a single user (one trustor - trustee). The concerned user evaluates the service's trustworthiness based on his/her own past experiences.

Indirect Evaluation consists of evaluating the trust level of a *service* or *system* from the perspective of the users' experience (N trustors - trustee). This technique is used when the users' direct experiences are not rich enough to determine the trustworthiness of services, or when the user is having a cold start with the service (first experience). We identify three indirect trust techniques including *Reputation*, *Recommendation*, and *Prediction*.

- *Reputation* [VHA05]; [Noo+13]; [KT09]; [Zhu+14]; [SC17] : it consists of collecting trust evidence about other users' experience (indirect) with the target service and taking it into consideration when evaluating the trustworthiness level. In this technique, the user (trustor) do not necessarily know the other users. Besides, we need to collect trust evidence only from similar-context-based user experiences when adopting this trust evaluation technique. Therefore, it requires prior knowledge about the other users and which services they have used. When the trust evaluation process is subjective, several challenges need to be faced when using this technique, including filtering the indirect evidence in a way to keep only those provided by trusted users,
- *Recommendation* [CN16]; [Li+14a]: This technique consists of collecting trust evidence about the target service through third parties/ service experts and taking advantage of their knowledge and expertise. Recommendations are most valuable when service experts and entities issue them with high knowledge.
- *Prediction* [Bao17a]; [JM16]: This trust evaluation technique is most useful when we do not have all information needed for the trust evaluation, especially with no historical records and no previous interactions with the concerned services. In this sense, the idea behind this technique is to transform users' preferences and quality aspects into meaningful information to predict the best suitable service. One solution is comparing these requirements to similar-minded users and learning from their experiences.

3.5 Selecting Trustworthy Data Services

Trust evaluation solutions require the choice of four key elements: a trust evaluation environment, a trust evaluation technique, a trust evaluation approach, and finally, the trust evaluation factors. In particular, we distinguish the following critical challenges of our solution related to: (1) The choice of the trust targets. (2) The selection of trust factors. (3) The properties of the trust evidence required for evaluating these trust factors.

Trust-based service selection solutions choose *services* as a trust target. Trust-based data selection solutions choose *data* as a trust target.

Trust-based service selection In recent years, trust evaluation solutions gained the attention of researchers in the service domain. They use trust for multiple reasons, including service selection and ranking. The evolution of trust solutions towards service-based environments is presented in table 3.4 which summarizes the number of published papers on trust per 5-year range from 2002 until 2019 for every service-based environment.

In the first five years (2002-2007) that correspond to the emergence of SOA, we can see that trust solutions in service environments started to appear and gained interest, generally for ensuring Web services reliability. Later, the number of papers decreased gradually while the number of papers addressing the cloud grew. These results can be explained by the growing success and democratization of the cloud [JoS+10]. Most researchers reoriented their interest to this new environment in which new trust challenges were to be addressed [JG11]. Under the cloud, service providers have direct access and control over many security and privacy aspects. In doing so, users confer a high level of trust onto providers, and at the same time, they expect them to respect this trust. This explains the importance of trust in cloud environments. According to our study, few solutions address trust in multi-cloud environments (26 - 5,8%).

Table 3.4: Number of papers per year range

	2002-2007	2008-2013	2014-2019
Cloud computing	0	63	109
Multi-Cloud	0	6	20
Other environments	41	157	50
Total	41	226	179

We are interested in objective trust evaluation solutions, particularly those that are based on monitoring and auditing, including SLA-based and QoS-based monitoring (see figure.3.5) in the different service environments.

In the cloud, solutions [Man15]; [CR12]; [CN16]; [Li+18] address the selection of services deployed at the "software as a service" and "infrastructure as a service" layers. In the Web and SOA environments, trust evaluation solutions evaluate Web Services' trustworthiness. Generally, these trust solutions differ in the used trust factors and trust metrics. They are not defined with the same formulas (depending on the use case application and the researcher's objective).

Most solutions [CN16]; [Man15]; [Tan+16]; [Tan+17] use the performance factor for the trust evaluation of services independently of their type and its associated metrics, including service response time/time efficiency, task success ratio and availability.

Trust-based Data selection Data similarity and data provenance are the most commonly used data trust evaluation factors in the database and web domains or peer-to-peer networks [Dai+08]; [Dai+09]; [BL10]; [LMB10]; [SAJM13]; [ZHI18]. The use of data quality as a data trust evaluation factor remains rare/immature in these domains (no standard evaluation, defined differently in different works etc.) but somehow exploited in other disciplines such as IoT etc.

When it comes to evaluating data freshness from the user side for data services when no information/trust evidence/meta-data are available (*Availability*), the general assumption is that data providers may export data quality observations as proof of goodwill to cooperate and show that they provide "good" data. Therefore, data quality metrics are evaluated in these works, assuming that the necessary evidence is available for this evaluation or can be easily interpreted using the available information about the trustee.

Data service trust evaluation We have observed that trust is an important property considered by proposals dealing with data provision and services. Selecting data-provisioning services is one of the most relevant challenges. Trust in data services concerns trust in telemetry. Data services regroup characteristics related to QoS and the way they function. These characteristics include performance, the conditions and strategies to capture data, data refreshment/update rate, and data sources production rate. Trust includes:

Considering the conditions in which data are collected and produced. In our scenario, the temperature data used for determining the general status of Alice is made using different production rates and are inserted into the services database using different insertion rates. Which data are most recent and most trustworthy? It will depend on the context of their use. A doctor can decide which ones to consider to observe Alice.

Selecting reliable data services that maintain and give access to the data. Are they available? Do they deliver data on time? In our scenario, services providing access to the different data are deployed in other conditions: The smartphone's services may be less performant than those deployed using the hospital's resources as their architecture is designed for efficiency.

According to the challenges mentioned in our scenario, we firmly believe that trust in data services should consider the trust level of two trust targets, namely service and data. To the best of our knowledge, no solution exists for the trust evaluation of data services, considering services and data as trust targets and their quality as trust factors.

3.6 Conclusion

This chapter proposed a study about trust in distributed (data centred) systems. Trust has been defined, modelled, evaluated, and interpreted in different disciplines and approaches. Since it is an "abstract" concept, in computing systems, trust is modelled as a combination of metrics observed at the different levels of the stack that defines such systems. Trust evaluation is thereby interpreted as trust level, also referred to as trustworthiness in the literature. The chapter has shown that families of existing work have used different metrics and different metrics compositions to define trust level formulae. Other works have proposed observation techniques and mathematical definitions of the metrics, generally based on statistics, probabilities, and machine learning methods performed on data collected by observing the behaviour or systems and/or their components. Our study shows that plenty of work has been devoted to measuring systems' performance. In contrast, data trust has been studied in the database domain, often related to data freshness, update ratio, and provenance. In general, approaches are intrusive, meaning that they deploy monitoring components within the systems or rely on metadata associated with data, their producers, and management systems.

The emergence of services-based architectures, particularly those centered in data, introduces new challenges regarding trust level evaluation. Indeed, in this type of system, services are independent external components that work in stateless contexts, as black-boxes, only accessible through API. The trust level of (black-box) data services is determined by non-functional and data measures. Therefore, trust level evaluation must combine non-functional behavior and data measures. However, under a black-box setting, it is necessary to develop original protocols that can observe services and data to evaluate their trust level. How to deploy mechanisms that can sample enough observations without perturbing the data service? How to observe the quality of the data service's data without having details about its back-end and metadata? How to assess data trust level when data are produced with high velocity (e.g., streams)? For how long do streams remain fresh and representative? Are they collected frequently enough to stay representative and up to date?

Our work aims to propose a solution for data service trust evaluation that considers, at the same time, service trust and data trust. Our work addresses the evaluation of the trust level of stream data services through *direct* evaluation. We adopt *classical mathematical models* for modelling their trust using performance and data quality as trust factors. Since data services are black-boxes, we develop protocols and strategies for collecting data to evaluate the trust level. These contributions are introduced and discussed in the following chapters.

Chapter 4

Data Freshness Evaluation Model for Black Box Data Services

Contents

"Trust is very hard if you don't know what you're trusting."

Marianne Williamson

4.1 Introduction

This chapter presents our model for data quality evaluation of black box data services using data freshness.

Knowing the quality of data delivered by data services for a given request is necessary to determine data reliability and whether it is adapted for a given task [Per06]; [CZ15]. However, the data quality evaluation process is challenging when this evaluation is done in IoT, Big Data, and Cloud Computing environments that introduce heterogeneity of data quality due to the number of data producers. Moreover, streaming services in these environments produce data continuously under different quality conditions (freshness, security, privacy, provenance, etc.). In consequence, data are provided under different quality levels.

Generally, data quality evaluation is context-dependent and can be achieved using different quality dimensions [Per06] (presented in chapter 2) including completeness, accuracy, freshness etc. Depending on (and not only) the environment, the application, and the target users' requirements and needs, the definition of data quality and its evaluation model differ. The process is the following: (1) defining and knowing users' requirements and needs, the environmental setting, etc., (2) selecting the quality dimensions accordingly, and (3) using these quality dimensions for defining the data quality evaluation model. Let's demonstrate this through the following examples. Suppose we were to consider a user using Google search engine to look for general information about some disease. In that case, the data quality can be evaluated using the completeness and exactitude of data. Indeed, the user requires complete and exact information. However, in our e-health scenario, we demonstrated that doctors need data to be up-to-date to know the current health state of Alice when doing chemotherapy. Therefore, the time factor, especially data freshness (currency), is more adapted and essential for modeling data quality evaluation.

Since we apply our results to applications for which continuous data must be up to date (e.g., e-health applications), we model data quality using only the time dimension focusing on data freshness. *Data freshness* indicates the extent to which data are up to date in the sense that they are representative for a target application [CPB12]; [Per+04]. The principle behind this is that fresh data are valuable and trustworthy compared to outdated data that lose their value throughout time and thus, can negatively affect (critical) decisions made using them. Therefore, data freshness is related to the degree of timeliness a data service can ensure. Evaluating *timeliness* can be achieved from the knowledge (i.e., evidence) we have about the configuration of the data sources (e.g., production rate) and the conditions under which data services continuously collect and store data (e.g., insertion rate). However, data services are black boxes and don't share such information about data configuration. Subsequently, it is essential to (1) define a model for data quality evaluation using *data freshness* for continuous data services and (2) define an observability protocol for collecting the necessary evidence for this evaluation.

As we pointed out in chapter 3, existing solutions related to data quality evaluation would target an application or a domain one at a time, and to the best of our knowledge, no data quality evaluation model was proposed for black-box continuous data services. All solutions would assume the evidence is available online or provided by the data provider. Thus, we proposed the model for data quality evaluation of black box data services using data freshness. We address issues related to the absence of the necessary evidence for this evaluation through the model. The objective of our solution is to rank data services according to their data quality levels.

This chapter is organized as follows. We define a data quality evaluation model for continuous data services in section 4.2. Section 4.3 introduces *TUTOR*: our da*T*a q*U*aliTy *O*bservability p*R*otocol which helps capturing the required information for this evaluation. Section 4.4 presents the implementation and validation of *TUTOR* applied on our e-health scenario. In section 4.5, we validate the entire proposal namely the data quality evaluation model using TUTOR. Section 4.6 concludes the chapter.

4.2 Data Freshness Evaluation Model

Data quality is generally defined by [CFP04]; [CZ15] as "fitness for use" - whether data satisfy users' expectations and quality requirements-. When data are continuously produced and updated into databases using services and have a limited validity duration, data requesters expect (telemetry) data to be up-to-date (i.e., fresh) when needed. We are considering data that are frequently changing, meaning data that have a tiny shelf life (i.e., very volatile), the kind of data whose value decreases rapidly over time in a matter of seconds. Thus, we are interested in measuring the extent to which data describe the best the current real-world situation [CPB12]; [Per+04]. For example, we want to know if the telemetry sent by Alice's devices and updated through services are fresh enough to represent Alice's current health status. Therefore, we model data quality through the freshness dimension (presented briefly in chapter 2). Indeed, the fresher the data, the best they describe the current situation, the more valuable they are, and the more trustworthy they become. For instance, in our scenario, the fresher the telemetry data accessed by a given data service are, the more helpful they are for doctors' diagnosis for Alice.

We evaluate data freshness using the *timeliness* sub-dimension - "data are made available as quickly as necessary to preserve the value of the data" [Kub+18]; [Bou04] -. We cover this criterion by *data timeliness* and *database timeliness.* The former describes the extent to which data are up-to-date (i.e., timely) when requested for use, and the latter indicates how often (telemetry) data are being inserted into the corresponding data service database (see figure 4.1). In other words, we want to make sure that the database behind a given service is refreshed or updated as often as necessary. This refreshment concerns mainly new telemetry data that reflect the real state of the world.

These two timeliness sub-dimensions are correlated: if the service often refreshes its database within the data validity interval, it will likely preserve the timeliness of data. This remains true only when the inserted data are already timely. Indeed, the service can refresh its database with stale data (no longer valid). However, the reverse is not necessarily true: the timely data does not necessarily indicate that the service often refreshes its database. For example, if we were in a hurry because of an emergency and requested traffic jam information, we would want a data service that provides the latest information. Let us take two data services providing the same information (e.g., traffic jam) stored in different databases and provided by the same sensor sensing the traffic jam data every $5s$. Let us suppose that

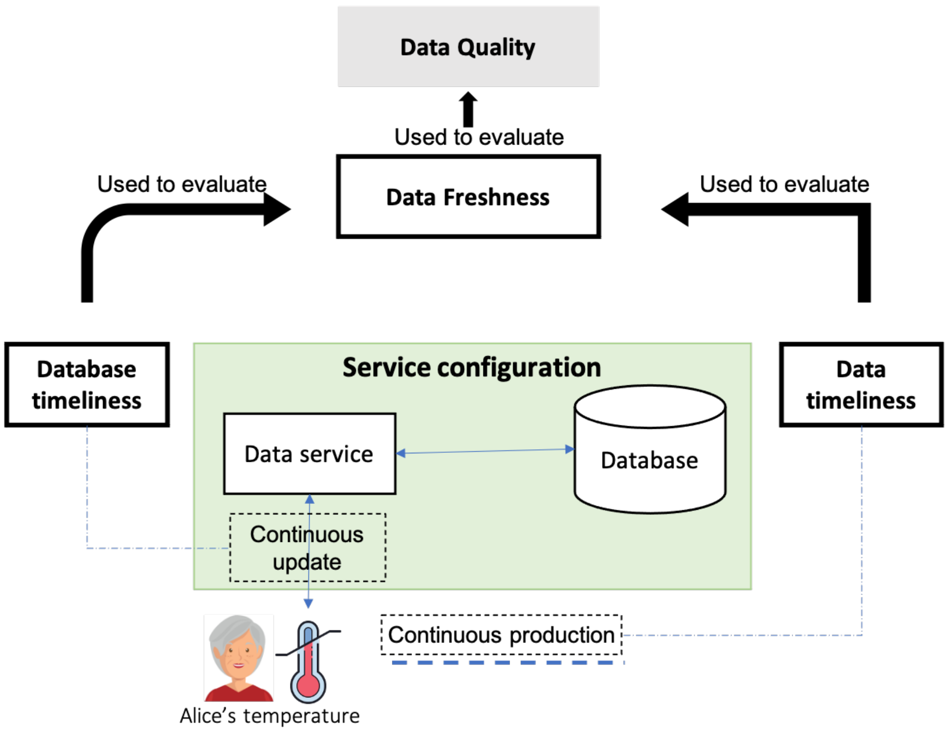


Figure 4.1: Data Quality Evaluation: Illustration

the information of the first data service is being updated (i.e., inserted) every minute into the first database. In contrast, the information of the second data service is being updated every $5m$ into the second database. In this case, the first service is likely to give us more trustworthy information than the second service because data are updated more frequently in its database. Thus, it is expected to provide more timely data that best describes the current traffic situation.

Hence, we define *data timeliness* and *database timeliness* as follows.

- ***Data Timeliness*** captures the gap between the production of data and the time when they are needed and makes sure this gap is in the data validity duration (see figure 4.2).

  Data validity duration T defines $[t_{min}, t_{max}]$ -the time duration in which data remains valid and is predefined according to the needs/requirements and the scope of the application domain. We use the term validity interval interchangeably with the term validity duration. For instance, the data validity duration T for temperature can be shorter than for weight because the temperature is more likely to fluctuate more often than weight. In our scenario, Alice's temperature is important information that indicates a possible infection. Considering the critical state of Alice and its variability during chemotherapy, we suppose that temperature remains valid for only $60s$. Thus, after a minute, data produced by devices are no longer fresh (i.e., outdated).

  Generally, continuous data form a batch of data (i.e., data set, DS) - a collection of ex-

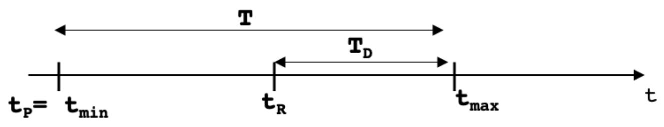


Figure 4.2: Data Timeliness.

tracted data from a data provider -. Therefore, data timeliness represents the timeliness of a data set. We define timeliness for DS noted as T_{DS} in $[0, 1]$ as follows:

$$T_{DS} = AVG(T_D) \tag{4.1}$$

Where T_D is the timeliness of data that belongs to DS defined in $[0, 1]$ as follows:

$$T_D = 1 - \frac{t_R - t_P}{T} \quad \text{if} \quad t_R < t_{max} \tag{4.2}$$

$$T_D = 0 \quad \text{if} \quad t_R > t_{max} \tag{4.3}$$

Where t_R represents the request time, t_P is the data production time, t_{max} is the maximum time for data to be fresh and $t_{min} = t_P$. The closer T_D is to t_{max}, the less the data are considered timely, that is, the less fresh they are. Beyond t_{max}, data in no longer fresh (see figure 4.2).

For example, back to our scenario, data timeliness of DS indicates to the doctors that the data service managing home thermometer measures might provide outdated or stale temperature readings that might not be accurate concerning the actual patient's health state.

- ***Database timeliness*** noted as T_{DB} indicates how often data source database is updated. The updates that we are interested in are the INSERTS.

  We say that a database is updated when a new data set is inserted into the database. The intuition is that frequent updates can contribute to preserving the freshness of the data within the database. We assume that if the database has not been updated during data validity interval, data in that database are more likely to be outdated. Therefore, the more a database is updated, the more likely data timeliness is preserved. The frequency of insertions gives the database timeliness.

To this end, the challenge is to determine to which extent the set of data (telemetry) provided by a given data service is fresh for the task at hand: does the database contain fresh data? At which rate the related data service refreshes the database? To which extent data are fresh?

Consequently, we define data quality in $[0, 1]$ as follow:

$$DataQuality = T_{DS} * T_{DB} \tag{4.4}$$

This section defines the timeliness metrics, including data timeliness and database timeliness, that we use to evaluate data freshness. We have also defined a formal model for data quality using both of these metrics. In the next section, we will present our data quality observability protocol, which helps capture evidence for measuring data timeliness and database timeliness and, thus, data quality level.

4.3 TUTOR: daTa qUaliTy Observability pRotocol

One of the most frequent ways to retrieve information from a data source is by querying a data service [CN02]; [Cam+14]. Querying data services may provide enough direct or indirect information about the data sources. Direct information is when meta-data about the data quality of the data source is available when querying the corresponding data service, and it needs no further calculation/measuring. Indirect information is observations about data quality inferred from evaluating, comparing and analyzing the different states of a data source via repeatedly querying the corresponding data service over time. [RB13]; [Sin07]; [CN02]; [Ada+09]; [NCO04]; [Mat05]; [GFT08]; [GF08] predict content change on the web. The overall objective is to keep local copies as fresh as possible. [BC00] try to answer questions about how fast a search engine must re-index the Web to remain ‘current’ concerning a definition of currency.

To tackle the problem of the absence or lack of data quality evidence, we propose a solution that creates knowledge about the data quality of black box data services using sampling techniques (see appendix 3.3 for more information). This knowledge is developed by continuously making observations (Indirect information) about a data source accessed by a given service through an original protocol, namely *TUTOR*. In other terms, *TUTOR* enables us to validate our data quality model.

Therefore, in the following subsections, we first present a general overview of *TUTOR*, describe its related timeliness knowledge design, and finally explain its process.

4.3.1 General Principle

The idea behind our observability protocol is to observe the database state changes of a given data service to evaluate the data freshness level and, thus, the data quality level, validating the proposed model.

TUTOR follows a series of steps as depicted in figure 4.3 and as described in the following lines:

- **Data sampling process**. First, knowledge is constructed about the data change history for a candidate data service, which will give us insights into the timeliness measure. This knowledge is built using sampling techniques and stored in *KDB*, our timeliness knowledge database.

  We believe that sampling helps make statistical inferences about the data quality of the

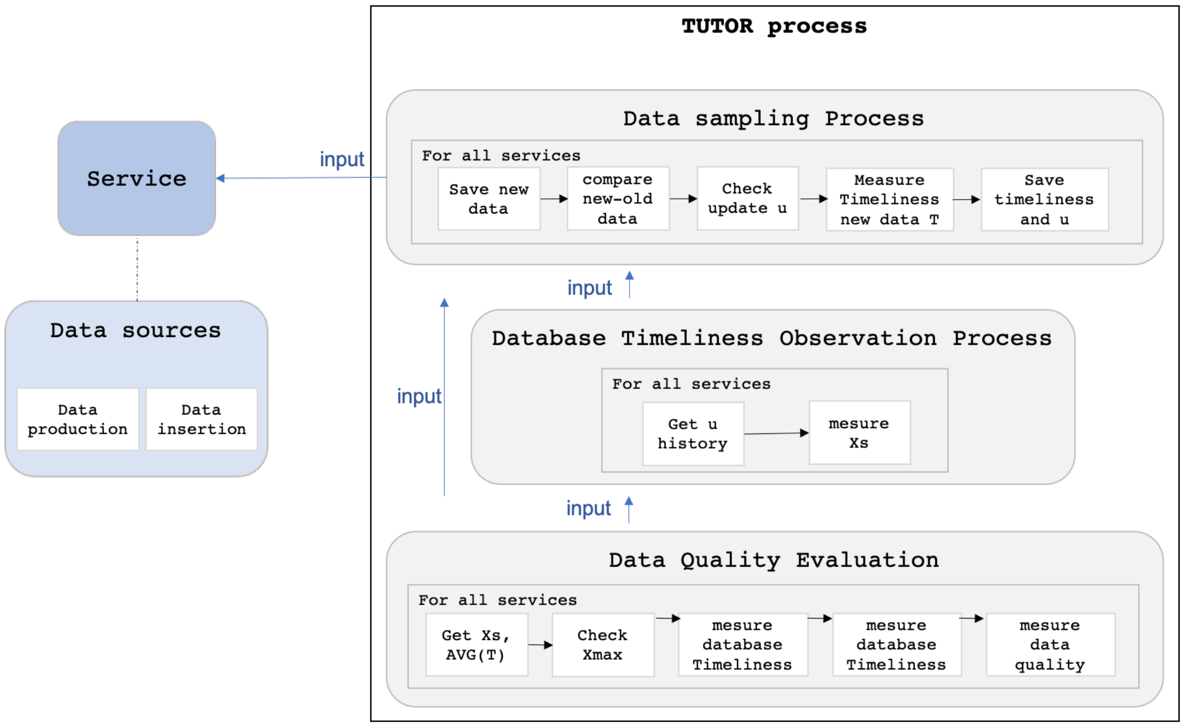


Figure 4.3: TUTOR Process

data source, especially data freshness. Therefore, to make good observations, choosing the right (i.e., informative and useful) sampling features for *TUTOR* is crucial. A sampling feature is an individual measurable property.

Gathering all information for the data quality evaluation can be time-consuming and costly regarding the number of sampling requests. Therefore, data must be sampled to find a trade-off between gathering enough representative samples about the data source in question for the measurement of the timeliness metrics and reducing the overhead induced by repeatedly accessing data services and pulling data samples. Thus, the choice of the sampling technique and frequency is crucial as it indicates how often *TUTOR* is accessing a data service.

- **Database timeliness observation process** Second, the made observations are periodically used as input for the database timeliness observation process, which evaluates how often a database behind a candidate data service is updated within an observation period -duration from the last evaluation-.

- **Data quality evaluation process** the observations made in the previous steps are periodically used as input for the data quality evaluation process to evaluate the possible data freshness level of every candidate data service. More details are given in section 4.3.3.

In the following subsection, we present *KDB* schema that is essential to store the made observations, and we describe *TUTOR* in more depth.

4.3.2 Timeliness Knowledge Model

The process devoted to building data quality knowledge uses *timeliness knowledge*. Therefore, we designed a *timeliness knowledge* database KDB to store the observations that we make about every candidate data source through our observability protocol $TUTOR$.

Choosing an informative feature is crucial for the observation in sampling solutions. To explain the choice of the observations we are making, we first need to describe the data service's environment to show what information is available. Second, how we can use them to measure the data quality metrics.

Hereafter, we describe the service's data configuration. We present the KDB data model and $TUTOR$'s sampling features. Next, we describe our proposed data quality observability and quality evaluation processes.

Data Service Configuration

Data sources feeding services and the way they are configured is depicted in figure 4.4. The premises used for adopting configurations come from the use cases addressed in the project SUMMIT.

Multiple devices populated data sources that produce data values at different production rates P_R (e.g., Alice's devices for capturing temperature values). Data production can be triggered as a result of an event (e.g., sudden temperature rise), periodically (e.g., every minute/hour or day), by a person (e.g., Alice checking up on herself at a given moment) etc. Therefore, the data production rate can be either static (remains the same over time) or dynamic (changes over time).

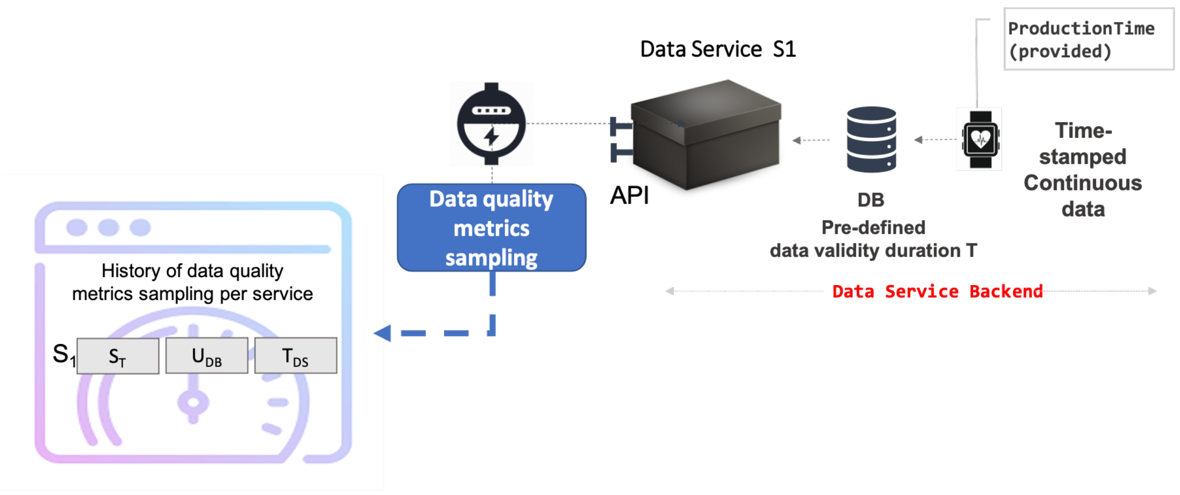

ProductionTime (provided)
Data Service S1
Time-stamped Continuous data
API
DB
Pre-defined data validity duration T
Data quality metrics sampling
Data Service Backend
History of data quality metrics sampling per service
S_1
S_T
U_{DB}
T_{DS}

Figure 4.4: TUTOR's data service background

Thus, data values are continuously produced and timestamped with the production time

ProductionTime. The production time corresponds to the instant the device has captured a data value from the real world. Therefore, for each produced data value V, we associate its production time *ProductionTime*. While data production rate is unknown, we suppose that both data values and corresponding production times are available and provided meta-data.

As aforementioned, data have a limited shelf life which means data only remain fresh over a time duration T. Therefore, the device must continuously refresh data by capturing new data values and time-stamping them. The sooner applications can use these captured data and turn them into valuable information, the more profitable they will be for the application users.

These devices give access to continuously captured data through black box data services. They are programmed to upload data streams (compose of a set of captured data values and their corresponding data production timestamps) to be inserted into a database of the data service that is linked to. This insertion is performed according to an insertion frequency called update frequency or frequency of update/change (*Uf*) that we believe differs from one candidate data service to another. Like the insertion rates, these update frequencies can be static (i.e., constant) or dynamic and are unknown by the applications using this data. Therefore, four case scenarios are possible for the production and the insertion of data that we have discussed in chapter 2.

Note that we are not interested in determining whether the production and insertion rates are static or dynamic (i.e., the case scenario of the data source) but to optimize our sampling process regardless of the case scenario (CS_i).

Finally, we define every black box data service $S = (API, ObsR, T)$ as being a data service which is accessible via a unique interface API. Each service holds one observation resource *ObsR* which contains the most recent captured set of data values and their production timestamps.

Please note that :

- A batch of data is inserted into a single insertion.
- Each database is accessed by one data service.
- An insertion is an update (a change) and it concerns one unique service resource Observation.
- We assume there are no delays in the performed insertions.
- Only the last version of a resource is available upon a request: we suppose that the observation resource *ObsR* is constantly updated with the most recent captured data. Therefore, by pulling a data sample at the instant S_t,, *TUTOR* only detects new data insertions. However, it is unable to determine the actual number of inserts that happened between the current sampling instant S_t and the last performed sampling at the instant S_{t-1}.

To summarize, by requesting data from a given black box data service, the last produced set of data values and their corresponding data production rates are received. Data production rate and database update frequency are unknown and not provided within meta-data. Therefore, the data quality of black box data services is observed using only data values and their production timestamps.

Next, we present the KDB database model that stores the observation made by TUTOR, including the description of these observations.

KDB Database Schema

Figure 4.5 represents the UML (Unified Modeling Language) class diagram describing the KDB database schema used to store observations.

- **ServiceCategory** class identifies and categorizes the available data services in the target service environment using their I/O data and validity duration.

  Therefore, each service category class is identified by a $category_{ID}$, *Output* data and the validity duration *T*.

- **Service** class describes services using their identifiers S_{ID}, their API, and *Input* data. Note that the *Input* attribute may be "NONE" as there are services that does not require an input data.

  A service category consists of several services, and one service may belong to only one service category.

- **SamplingFeature** class describes the sampling features that characterize the different sampling requests. This helps to identify the observations made within a given sampling setting.

  A given sampling setting may refer to one or many samples (i.e., sample set). Each sample set is characterized by: a $Feature_{ID}$, a start sampling time *StartSampling*, an end sampling time *EndSampling* and a sampling frequency S_f.

- **DataRec** class represents the data samples for a given service, including the several data values and their corresponding production timestamp.

- **SamplingRec** class represents the sampling record for each performed sampling using the sampling timestamp.

  A sample *E* is of size *m*, represented by the class *DataRec* and is defined as:

  $$E = (V_{i\in[1,m]}, ProductionTime_{i\in[1,m]}) \tag{4.5}$$

  Example of sampled data at time instant S_t:

  $$\text{E}=(V_1,ProductionTime_1),(V_2,ProductionTime_2),(V_3,ProductionTime_3)$$

V_i	$ProductionTime_i$
V_1	14h52
V_2	15h00
V_3	15h10

Table 4.1: Example of sampled data at time instant S_t

For each performed sample request, we fill the class *SamplingRec* which keeps the records of each sampling with the sampling identifier *Sampling_ID* and contains observations as defined in the vector v as :

$$v = (S_t, T_{DS}, U_{DB}) \tag{4.6}$$

Where:

- S_t represents the sampling time t.
- T_{DS} represents the timeliness of the data sample E at instant S_t and measured as defined in equation 4.1.
- U_{DB} is a Boolean value (0 or 1) which indicates whether a change is detected in the data service's database state since the last performed sampling.

  If database state changes (i.e., new data inserted into the database since the last sampling):

  $$U_{DB} = 1 \tag{4.7}$$

  If database state did not change (i.e., no new data inserted into the database since the last sampling):

  $$U_{DB} = 0 \tag{4.8}$$

  We suppose that an insert containing outdated data is still considered an update/new insert (U=1). For instance, consider a device that lost its connection to the internet and continues to produce data. Once the connection is re-established, it sends old produced data to update the database.

S_t	T_{DS}	U_{DB}
S_{t1}	0,5	1
S_{t2}	0,73	1
S_{t3}	0,1	0

Table 4.2: Example of database knowledge which contains observations about 3 samplings

We can perform zero or several sampling requests to the corresponding service. Table 4.2 presents an example of the database containing three consecutive data samples.

Figure 4.6 illustrates an object diagram as an example of a *KDB* database that consists of two sampling points. For the service *Service1*, we have one category of sampling features

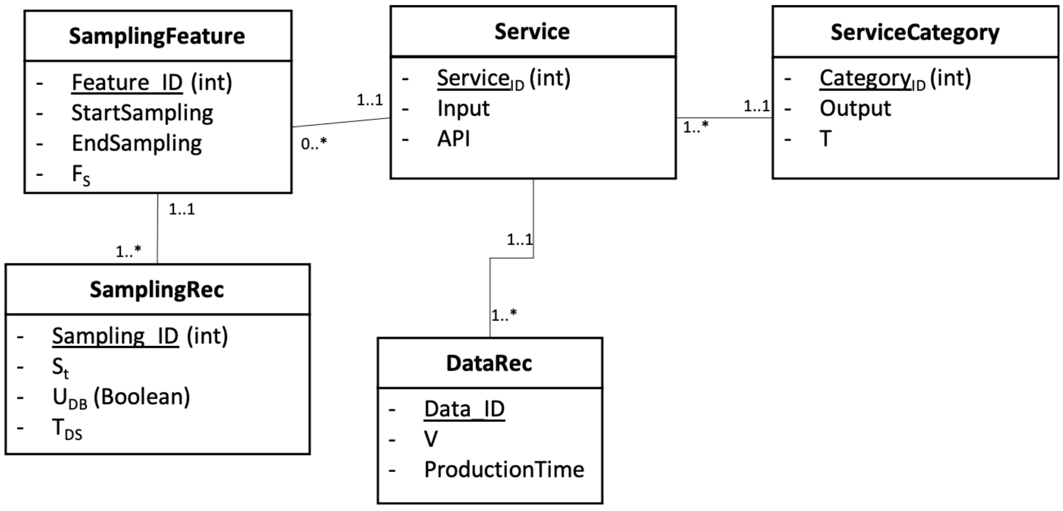


Figure 4.5: KDB Schema Class Diagram

which regroups samplings records (*SamplingRec1*, *SamplingRec2*) obtained between instant *t1* and *t2* and with a fifteen-second duration (S_f) between those samplings (static sampling frequency).

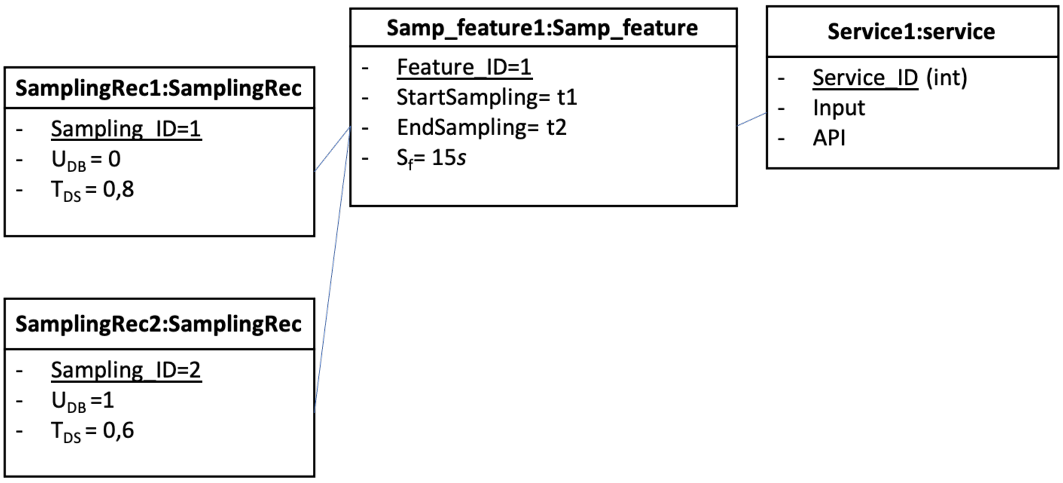


Figure 4.6: KDB Object Diagram: Example

As mentioned, the KDB database stores the observations made using TUTOR's process that we present in the next section.

4.3.3 Data Quality Observability Process

To understand how *TUTOR* works and how it transforms data samples into knowledge, we describe its process, composed of two steps: *knowledge development* and *data quality evaluation*, and present their corresponding algorithms. Table 4.3 matches the used terms in our algorithms to their meanings.

Cts	Current time system
csid	Current data sample identifier
$data$	values extracted from the sample
f_T	Frequency of change per period T
k	Number of T intervals per monitoring period
lsid	Last sample identifier
lf_T	List of frequencies of change per period T for all candidate services
lAPIs	List of APIs of all candidate data services
ROP	The recent observation period of the history of database state change.
S_{ID}	Unique identifier of the target data Service S
S_f	Sampling frequency of TUTOR
T_{DS}	Average timeliness of data sample of the target data Service S
ts	list of data productions timestamps of the values in the sample.
T	Validity interval for the target data services.
U_{DB}	Database Update

Table 4.3: Observability Protocol Notations

Knowledge Development Algorithm

Knowledge Development step is achieved continuously in the back-end through blind sampling data using either random or systematic (i.e., static) sampling frequencies. For each service category, our knowledge development algorithm follows three steps as presented in algorithm 1 including extracting the list of candidate services, sampling and collecting sampling features, and constructing knowledge through observations.

- (**STEP 1**). First, we extract the list of available candidate data services related to a given category, their related identifiers (line.3), and their validity duration T (line.4).

  Note that service categories are known and fixed. The process must refresh the list of available services according to their type.

- (**STEP 2**). Second, we start our data quality observability process by data sampling, which is performed through RESTful GET operations performed on the resource *Observation*. The sampling is performed for all the available data services using their APIs (line.6). Also, we use the same sampling frequency S_f for data services providing access to the same data/observation output.

  The identifier of the last sample for the corresponding service is then extracted (line.7).

  The output of the sampling E for every data service is a list of data values and their corresponding production timestamps (line.8, see description above).

  Recall that only the latest update on the observations are detected. When querying the services' resources through their provided URL, they will respond to the requester by delivering the latest modified versions. Therefore, *TUTOR* may miss some updates

Algorithm 1 TimelinessKnowledgeConstrucstion ($Category_{ID}$, S_f)

```
1: $INPUT \leftarrow Category_{ID}, S_f$
2: while true do
3:     $ls = extractLs(Category_{ID})$
4:     $T = ExtractValidityDuration(Category_{ID})$
5:     for $services \in ls$ do
6:         $API = GETapi(services)$
7:         $lsid = extractLsid(services)$
8:         $data, ts = GETdata(API)$
9:         $csid = insertData(data, ts)$
10:        $U_{DB} = checkUpdate(lsid, csid)$
11:        $Cts = currentSystemTime$
12:        $T_{DS} = measureTimeliness(csid, Cts, T)$
13:        $sample = (Cts, T_{DS}, U_{DB}, S_{ID})$
14:        $createSample(sample)$
15:        $sleep(S_f)$
16:    end for
17: end while
```

between two data sampling points which may consequently reduce its effectiveness in detecting updates.

For instance, let us say that data is being posted by the *data producer* and inserted into the data service database every *Xs* (second). Suppose we query the database at an instant after three performed inserts. In that case, we will be only able to observe the last inserted (posted) data, and we won't be able to see the first two inserts, which will reduce the effectiveness of TUTOR in evaluating the database update frequency.

- (**STEP 3**). At an instant *t*, when a sample number *N* is pulled, it is explored, and thus, TUTOR starts developing knowledge for every candidate service. This is established via the following process.

  - (1) After storing the output (line.10), we check if the database has been updated (line.10) using the function *checkUpdate* which takes as input the current sample ID *csid*. We suppose that a database of a given service is changed if the sampled data *N* is different to the previous data sample *N-1*.

    If the two samples are different (i.e., the intersection size is null), then U=1. Else if the samples are identical, then U=0. Recall that a sample that fully contains outdated data and is different from the last sample is still considered as an update,
  - (2) We compute the function *measureTimeliness* (line.12) which is programmed as follows. It applies the equation 4.2 to compute *data timeliness* T_{Vi} for each data value in the sample with the ID *csid* using the currents timestamps *cts* and the validity duration *T*. Then, it applies the average of the computed *data timeliness* measures in order to have the timeliness level of all the data set T_{DS} as in equation 4.1.

As described above, these observations, including T_{DS} and U, are stored in *KDB* with the sampling timestamp and the corresponding service identifier (line.14).

– (3) Lastly, this process sleeps for the duration S_f before executing the subsequent sampling (loop).

Now that we have presented the algorithm of the first step of our observability protocol about knowledge development, we explain how this knowledge is used to evaluate data freshness and, thus, data quality.

Data quality evaluation algorithm

As presented in our data quality evaluation model, data freshness is evaluated using metrics for database timeliness. Data quality is evaluated per monitoring period (ROP) expressed in seconds (s). In this sub-section, we show how we make use of the *KDB* for the computation of these metrics by presenting the data quality evaluation algorithm along with how data freshness is computed.

- **Database timeliness evaluation**. This evaluation requires knowledge about the update frequency U_f of the related data service's database, which is unknown.

  Therefore, using algorithm 2, the history of U_{DB} for all services during the period ROP ($Lts < S_t < Cts$) is extracted, and used to evaluate the U_f of each data service Si defined as follows:

  $$Uf_{ROP,Si} = \frac{x_{ROP,Si}}{k} \tag{4.9}$$

  Where

  $$k = \frac{ROP}{T} \tag{4.10}$$

  evaluated in line.5 and

  $$x_{ROP,Si} = \sum U_{DB} \tag{4.11}$$

  evaluated in line.7.

  According to this definition, data services with higher U_f are more likely to have better data freshness levels as their accessed data are more likely to adhere to the real world. However, this does not remain true when inserts concern out-of-date data. Thus, *database timeliness* (line.14) is defined as follows:

  $$T_{DB,ROP,Si} = \frac{Uf_{ROP,Si}}{UfMax_{ROP}} \tag{4.12}$$

  Where $UfMax_{ROP}$ is the maximum observed value of *update frequency* during the validity duration T among the available data services for the monitoring period *ROP*.

- **Data timeliness evaluation**. This metric is also measured for the monitoring period *ROP* and is defined as follows :

$$T_{D,ROP,Si} = AVG(T_{DS})_{Si} \tag{4.13}$$

- **Data freshness evaluation:** *Data Quality Evaluation* process measures data freshness of the available services using the evaluated timeliness metrics $T_{D,ROP,Si}$ for the period *ROP*. The data freshness level gives us the data quality level during the period *ROP* for the service Si as follows:

$$DataQuality_{ROP,Si} = T_{D,ROP,Si} \times T_{DB,ROP,Si} \tag{4.14}$$

  Finally, data services are tagged with their measured data quality levels that are stored in a database, namely *EDQ* along with the timestamp of the evaluation.

Algorithm 2 EvaluateChangeFrequency (lAPI, T, ROP)

```
1: INPUT ← $lAPI, T, ROP$
2: OUTPUT ← $DatabaseTimeliness_{ROP,S_{ID}}$
3: $lf_T = ()$
4: for $API \in lAPI$ do
5:     $k = \frac{ROP}{T}$
6:     $Lts = Cts - ROP$
7:     SQL = " SELECT count(*) FROM samplingRecord where $S_t$ BETWEEN Cts AND Lts AND $U_{DB}$=1 "
8:     $X_S = Execute(SQL, ROP, API)$
9:     $Uf_{ROP,S} = \frac{X_S}{k}$
10:    $lf_T.append(Uf_{ROP,S_{ID}})$
11: end for
12: $UfMax_{ROP} = max(lf_T)$
13: for $row \in lf_T$ do
14:     $DatabaseTimeliness_{ROP,S_{ID}} = \frac{Uf_{ROP,S_{ID}}}{UfMax_{ROP}}$
15: end for
```

In this section, we presented TUTOR, our data quality observability protocol, by presenting (1) its general principle, (2) the observations it makes about a data source, the design of KDB which stores these observations and (3) its process. In the next section, TUTOR is validated through experiments.

4.4 TUTOR : Implementation and Evaluation

Motivated by the e-health domain that served as application context to our work, *TUTOR* was validated using medical data services. Still, the choice of a sampling frequency is required for its validation and the validation of our data quality evaluation model. Indeed, a significant challenge for data quality observation is to select a sample from a data set that effectively represents the whole data set while reducing the overhead caused by sending repeated sampling requests. The selection of samples is guided by the method, including the sampling frequency. This method lets *TUTOR* determine how often a data sample should be selected (i.e., at which rate) from a given data source. Sampling methods are described in appendix 3.3.

We target two categories of random sampling, namely simple random sampling (random sampling) and systematic sampling (static sampling). Random sampling requires setting an interval from which our protocol needs to randomly select a value that will indicate when the subsequent samplings will happen. Static sampling requires setting a fixed value which suggests that *TUTOR* needs to repeatedly wait for a duration equal to this value before performing another sampling. Indeed, we aim to select one of these two methods for choosing samples and observing the timeliness measures for the data quality evaluation process while performing an optimized sampling: *"Sample as much as possible to maximize data quality evaluation accuracy (success rate), Sample as little as possible, to conserve computing cost (detection rate)."*

However, the choice of the sampling method should be based on the knowledge about the data pipeline's configuration (case scenarios presented in chapter 2). We believe that this choice may influence the effectiveness of *TUTOR* in measuring the data quality by reducing its *success rate* as well as reducing the spectrum of the observed *data timeliness* values on average. This influence can be more or less significant for the different case scenarios.

Illustration: Let us consider the case scenario *CS2* where data values might have tendencies to be produced and inserted into a database more frequently at night than during the day (e.g., data about apnoea which detect the cessation of breathing during sleep). In this case, sampling with a random sampling frequency seems more suited to minimize the number of access to the data service (better detection rate) while still detecting database updates (good success rate) and capturing and measuring the different values of data timeliness. Performing data sampling with a static sampling frequency is worst for observing the frequency of change of the given data source with this type of data pipeline configuration. We are likely to access the data service more than necessary, resulting in a poor detection rate. This means that the number of sampling times is superior to the number of detected updates).

This section is organized as follows. First, we introduce the architecture of *TUTOR* and the chosen experimental setting. Second, we discuss the feasibility/practicality and the effectiveness of our proposed protocol TUTOR, including the choice of its sampling frequency.

4.4.1 TUTOR general architecture

Figure 4.7 depicts the architecture of our solution which is composed of *HAPI FHIR* data services, a *data provider* component, and a *Data Quality Evaluation Module*, our main component (*DQEM*) detailed hereafter.

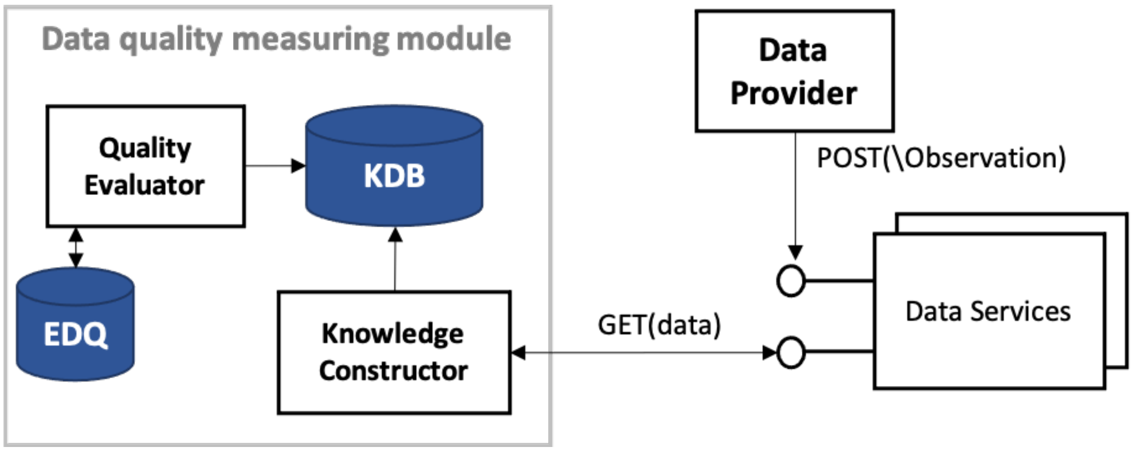


Figure 4.7: General Architecture: Data Quality Evaluation for Black Box Data Services

HAPI FHIR Data Service. is a complete implementation of the HL7 FHIR (Health Level 7 Fast Healthcare Interoperability Resources) standard for healthcare interoperability and data exchange developed in Java, published by HL7®. *HAPI FHIR* data services are built from a set of components called *Resources* used to exchange and/or store data and can be easily assembled to solve a wide range of healthcare-related problems. A resource is an entity that (1) has a known URL by which it can be accessed, (2) contains a set of structured data items and (3) has an identified version that changes if its contents change. We mainly measured the quality of data accessed through the resource *Observation*[1] which is a central element in healthcare, used to support diagnosis and monitoring progress. Uses of the Observation resource include (1) *vital signs* such as blood pressure and temperature, (2) laboratory data like blood cells count, (3) device measurements such as EKG data, and so on. Therefore, we can develop our e-health scenario using this resource for monitoring temperature. *HAPI FHIR* resources are queried using RESTful APIs (With the normal operations; GET, POST, PUT, DELETE, UPDATE).

Data Provider. This component simulates the process of data production, and posts produced data to the resource *Observation* of *HAPI FHIR* servers through a POST operation. The production (respectively the posting/insertion) process is performed according to a configurable production (respectively posting/insertion) rate. We can set these rates to be dynamic or static when we test *TUTOR* (see the different case scenarios presented in section 2.3.2).

Data Quality Evaluation Module. This component is responsible for observing and evaluating the quality of data accessed by the data service "*HAPI FHIR*". As described above, the data quality evaluation process consists of two steps: first, constructing knowledge about the data source using our knowledge database *KDB*, and second, using it in evaluating the data quality level which is stored in the database *EDQ*.

[1]`https://www.hl7.org/fhir/observation.html`

4.4.2 Choice of the Random Sampling Method

We suppose that we do not know about the data pipeline configurations of the candidate data services due to the black-box model, which makes selecting a data sampling method a challenging task. The challenge is to configure *TUTOR* in a unified way by setting a sampling method to provide the best observations regardless/independently of the case scenario (i.e., that is best applicable for the presented case scenarios). One can intuitively sample with minimal static frequency to detect database inserts as much as possible, which is resource consuming. Moreover, we want to make sure that *TUTOR* capture any variation in the observed data timeliness on average since it reassures that the captured data samples by *TUTOR* represent the different states of the data source. Indeed, data production and insertions rate may variate suddenly at the data source level in the background, and thus, its timeliness change. At the same time, we want to make sure that our protocol captures timely all changes. Therefore, our strategy is to study each case scenario separately and then find an optimal base ground sampling method for all of them.

To this end, the objective of the following experiments is to study in-depth, for each case scenario, the effect of the choice of the random sampling method, including the selection of the sampling frequency on the effectiveness and efficiency of *TUTOR* in detecting updates, and on the variation of the evaluated *data timelines* metric on average. This is done by (1) performing the same experiment for each case scenario separately and (2) evaluating and comparing their results. Note that we only study the first three case scenarios (static production-static insertion, static production-dynamic insertion, dynamic production-static insertion) since the fourth case scenario can be seen as a variety combining *CS2* and *CS3*.

In the following sections, we (1) discuss the setting of these experiments, (2) present the sampling evaluation metrics that enable the comparison of the sampling methods, (3) present the results, and (4) discuss the choice of the sampling frequency for *TUTOR* while validating it.

Experimental setting

To experiment and prove the feasibility of *TUTOR*, we propose a data quality monitoring solution developed using the *Docker* container technology[2] as a hosting environment with a complete execution stack. This platform enables the configuration and deployment of data services in self-contained execution environments. The choice of this deployment solution was motivated by the need of testing the collection of data freshness metrics from data services hosted on separate containers with different configurations and, thus, distinct data quality. The scenario runs on a 64GB Macintosh MacBook Pro. We use Alice's e-health scenario to experiment *TUTOR*. As aforementioned, T is fixed for our e-health scenario to $60s$ since data about Alice's temperature change frequently given her illness. Thus, her medical state requires monitoring in terms of seconds.

[2] `https://www.docker.com`

Our experiment consists of running *TUTOR* using two sampling methods including **static** sampling using three different static sampling frequencies ($30s$ - half the validity interval, $70s$ - around the validity interval, and $150s$ - a bit more than twice the validity interval) and **random** sampling with a sampling frequency value in the range of $[5, 150]s$. Therefore, a total of 4 sampling frequencies are used in this experiment (i.e., 4 instances of *TUTOR*).

The strategy behind the choice of the sampling frequencies values for our experiment is to check the effectiveness and the efficiency of *TUTOR* when the sampling frequency is static VS random and faster VS slower: (1) we repeatedly sample once per validity interval, (2) we repeatedly sample more than once per validity interval, (3) we repeatedly sample once per two validity intervals, and (4) we randomly sample until 10 times per validity interval and once per two validity intervals.

We run the same experiment for three different case scenarios including *CS1*, *CS2* and *CS3* using one *HAPI FHIR* data service as follows:

- For *CS1* (static insertion-static production), the production rate is static and set to $5s$ and the insertion rate is static and set to $50s$.
- For *CS2* (static production - dynamic insertion), the production rate is static and set to $5s$ and the insertion rate is random in the interval of $[30, 100]s$ (The waiting duration before performing another data insertion into the database is each time chosen randomly from this interval).
- For *CS3* (dynamic production - static insertion), the production rate is random in the interval of $[3, 10]s$ and the insertion rate is static and set to $50s$.

Indeed, the four instances of *TUTOR* are run independently for *CS1*, then, *CS2*, and then, *CS3*. We recall that our objective is study each *CS* independently, and then, compare the obtained results of the three *CS*s. Note that we variate the size of the inserted data as it happens in real-world due, for instance, to network failure, disconnected device, unavailability, etc. Thus, data may be stored locally to be sent later and inserted into the corresponding database.

Each instance of *TUTOR* is run for a duration over $24h$ where it executes several number of sample requests N for two hours approximately over 20 iterations (i.e., loops) j. The corresponding sampling frequency value determines the duration between two sampling requests. TUTOR is programmed to wait a random small amount of time (seconds) before executing the next iteration. The choice of the number of iterations j, their duration, and the time to wait before executing each iteration is motivated by the need to observe data timeliness on average over the longest period and have more insights for the interpretation of results.

When the sampling frequency is random, for each iteration j, we re-initialize the random seed, which sets the sampling frequency interval. TUTOR follows the process as depicted in figure 4.3.

Concerning data timeliness observation, we follow a series of steps for our experiment (see figure 4.8): (1) For each sampling request, we measure the sample *data timeliness* as in equation 4.1. (2) At an instant *Cts*, we measure the average of the obtained data timeliness values for the last 500s meaning data timeliness values measured between *Cts* and *Cts*-500s to obtain data timeliness on average per 500s. The choice of the duration 500s is random, and motivated by the need to average data timeliness over a period that is superior x times to the validity interval, and at the same time, for more insights, to have enough number of values of data timeliness on average per iteration j (AVG$(T_{DS})_1$.. AVG$(T_{DS})_{12}$). Therefore, we obtain 12 values i of data timeliness on average per iteration j (total of 6000 sec) at different instants namely $t_{i,j}$ as illustrated below. A row corresponds to the data timeliness measures on average per 500s for one iteration.

$$\begin{pmatrix} t_{1,1} & . & . & t_{1,20} \\ t_{2,1} & . & . & t_{2,20} \\ . & . & . & . \\ t_{12,1} & . & . & t_{i,j} \end{pmatrix}$$

(3) For each i (i.e., for each line), the obtained 20 values of data timeliness on average are averaged for all iterations to obtain *data timeliness* on average per instant t_i=1:12. For instance t_1 is obtained through averaging ($t_{1,1}$, .., $t_{1,20}$).

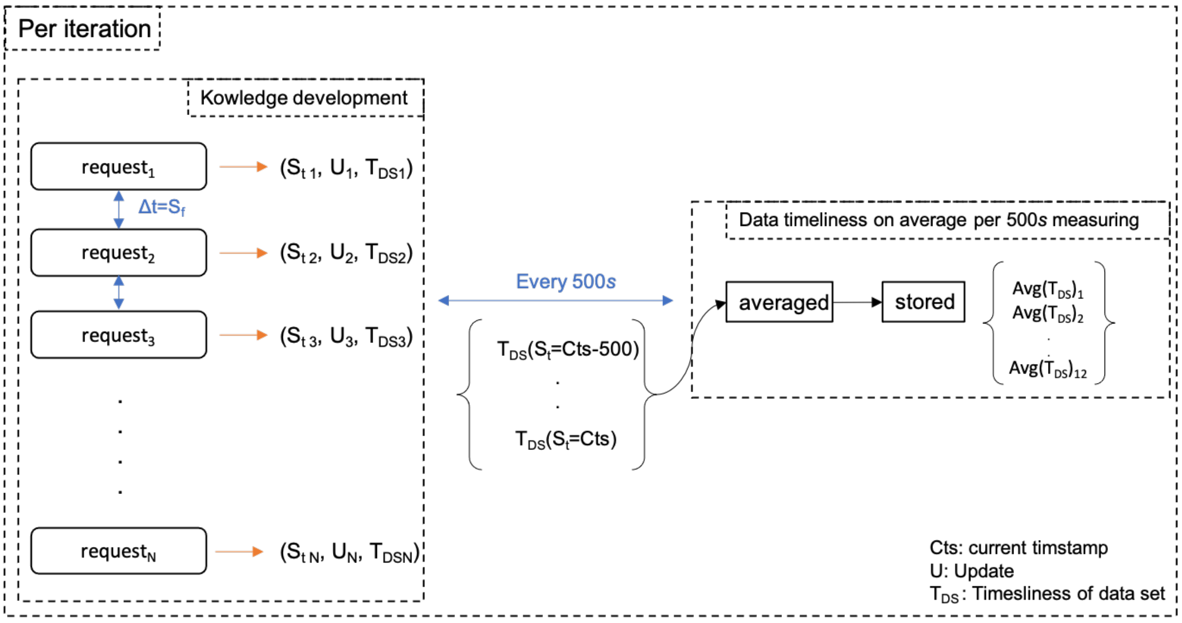


Figure 4.8: Data timeliness evaluation per 500s

Concerning database update detection, we launch our experiment and measure the database update as described in section 4.3.3 using the observation U. Then, the results obtained using each instance of TUTOR are evaluated using the metrics presented in the next section.

Sampling Evaluation Metrics

As aforementioned, the objective of the experiments is to select a sampling method for $TUTOR$ that enables it to provide compelling observations for the data quality evaluation process while being efficient in reducing the overhead caused by the knowledge development. Therefore, we evaluate the effectiveness and the efficiency of $TUTOR$ for various sampling methods using two evaluation metrics, including the success rate [NU16] and the detection rate. Moreover, we evaluate the trade-off between effectiveness and efficiency. Indeed, these metrics enable us to compare the sampling methods according to the efficacy of $TUTOR$ and its efficiency to detect updates. Subsequently, these metrics allow the selection of an optimal sampling method for $TUTOR$ while proving its feasibility.

Effectiveness. We evaluate the *effectiveness* of $TUTOR$ by measuring its ability to detect data sources updates of a given data service using the *success rate* metric S_R. We define S_R, the rate of successfully detected database updates, in $[0, 1]$ as follows:

$$S_R = \frac{X_S}{RealX_S} \tag{4.15}$$

Where X_S is the number of detected updates during the monitoring period for service S using $TUTOR$ and *RealXs* is the actual number of realized updates during the same monitoring period for service *S*. Note that since we have a white-box setting, and given the objective of our experiment (finding the best sampling frequency for $TUTOR$), the actual number of the realized updates by a given data service is known.

Efficiency. We evaluate the *efficiency* of $TUTOR$ by measuring its ability to detect updates with a minimum number of sampling points using the *detection rate* metric D_R defined in $[0, 1]$ as follows:

$$D_R = \frac{X_S}{N} \tag{4.16}$$

Where X_S is the number of detected updates during the monitoring period for service S using $TUTOR$ and N is the total number of the sampling points performed during the monitoring period using $TUTOR$ for service *S*.

Trade-off. The measurement of value trade-offs is central to applied decision analysis [Fis95]. Trade-offs between two factors can be evaluated by deciding which factor is more important than the other. [Bar01]. The *trade-off* of $TUTOR$ is evaluated by considering two factors including its effectiveness and its efficiency and is defined in $[0, 1]$ as follows:

$$T_o = \gamma * S_R + \theta * D_R \tag{4.17}$$

Where γ, θ weights are attributed consecutively to the success and detection rates. They vary according to the importance we give to each of these factors when validating $TUTOR$. The objective is to see the effect of changing these weights on the choice of the sampling frequency.

Based on those introduced evaluation metrics, including detection rate, success rate, and trade-off, a sampling method is selected for *TUTOR* which is then validated.

Experimental Results & Discussion

In the following, we present the results for the experiment described above concerning *data timeliness* variation and the ability to detect updates in an optimized way.

Variation of *data timeliness* Results about data timeliness variation on average for the different iterations for each case scenario are presented below.

The observed values of data timeliness on average per instant t_i=1:12 are represented on the horizontal axis and then linked together to form the graph in the different figures hereafter.

Table 4.4 and figure 4.9 present the obtained results related to the data timeliness variation for the first and the most simple case scenario *CS1* for the different sampling frequencies.

According to the figure, throughout time (t_i), the measured data timeliness on average does not variate much (low variation). Variation measures the difference between the maximum value and minimum value of data timeliness of average. We observe that the data timeliness measure obtained through the random sampling frequency variates more for the different iterations on average.

	$30s$	$70s$	$150s$	$[5, 150]s$
Min	0,34	0,33	0,30	0,32
Max	0,38	0,38	0,38	0,45
Variation	0,04	0,05	0,08	0,13

Table 4.4: CS1 : Data timeliness variation

Table 4.4 also show that the biggest variation is obtained through the random sampling frequency (± 0,13). Moreover, the bigger the duration between two static sampling points, the bigger the variation in the observed data timeliness on average (± 0,04 when S_f is $30s$ and ± 0,08 when S_f is $150s$).

	$30s$	$70s$	$150s$	$[5, 150]s$
Min	0,12	0,10	0,09	0,10
Max	0,14	0,14	0,15	0,15
Variation	0,02	0,04	0,06	0,05

Table 4.5: CS2 : Data timeliness variation

Table 4.5 and figure 4.10 present results about data timeliness variation for *CS2* for the

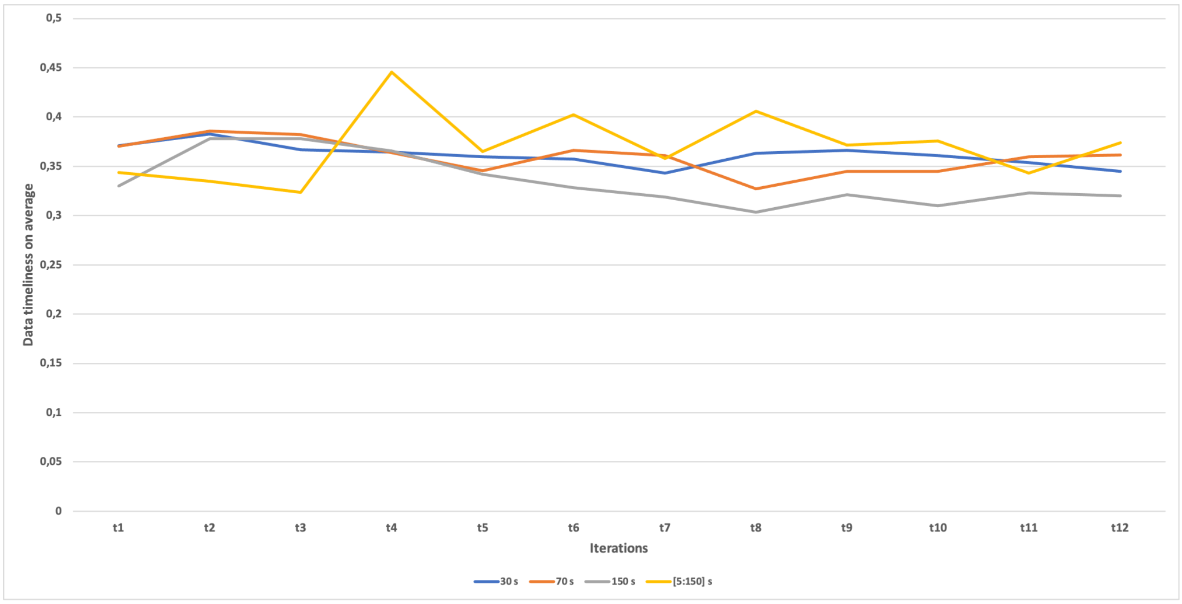


Figure 4.9: CS1 - Data timeliness variation for the different sampling frequencies

different sampling frequencies.

According to the figure and the table, we observe that for static sampling frequencies, the smaller the sampling frequency, the less variation is observed in data timeliness measure. The variation is about ± 0,02 when S_f is 30s, ± 0,04 when S_f is 70s, ± 0,06 when S_f is 150s, and ± 0,05 when S_f is random.

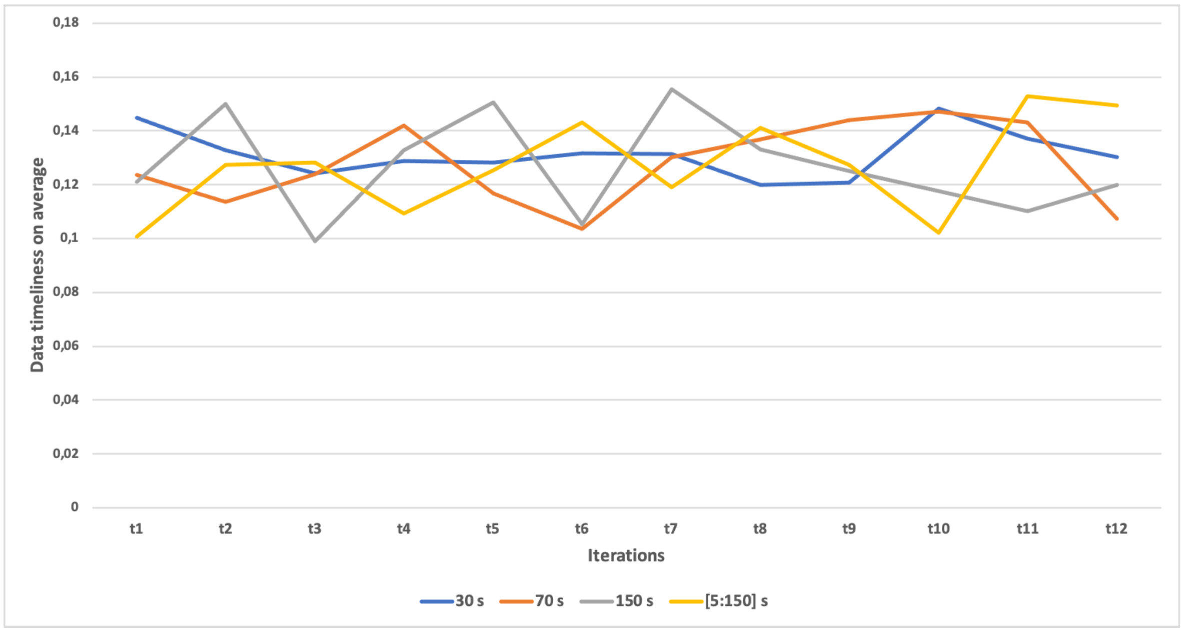


Figure 4.10: CS2 - Data timeliness variation for the different sampling frequencies

Table 4.6 and figure 4.11 present results about data timeliness variation for *CS3* for the different sampling frequencies. According to the figure and table, we observe that the variation of data timeliness on average for all frequencies is more or less the same and follows somehow the same pattern except for S_f=150s which variates more.

According to table 4.6, the evaluated variation in the data timeliness measure on average is bigger using the static sampling frequency with a larger spectrum of observed data timeliness measures (from ± 0,02 when S_f is 30s to ± 0,13 when S_f is 150s).

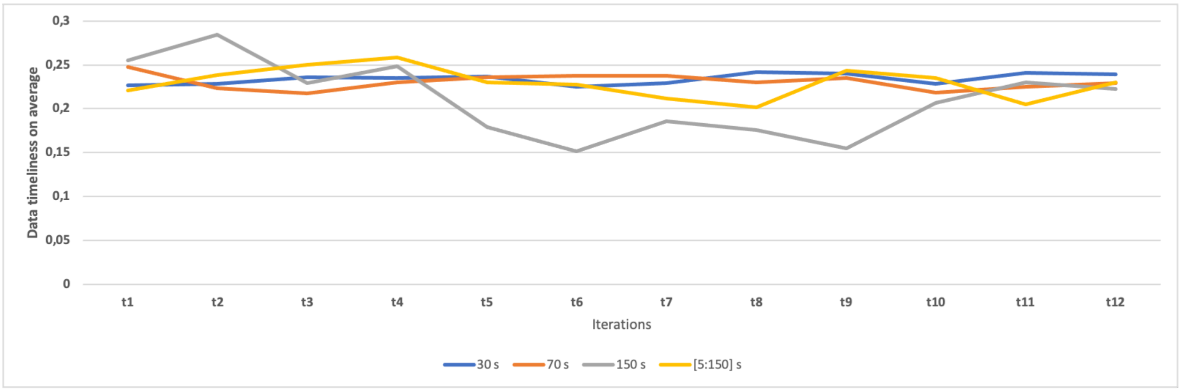


Figure 4.11: CS3 - Data timeliness variation for the different sampling frequencies

	$30s$	$70s$	$150s$	$[5, 150]s$
Min	0,22	0,21	0,15	0,20
Max	0,24	0,24	0,28	0,26
Variation	0,02	0,03	0,13	0,06

Table 4.6: CS3 : Data timeliness variation

The obtained results for the three case scenarios showed that the smaller the sampling frequency, the minor variation observed in the average data timeliness. This can be explained by the fact that the more we reduce the time interval between two sampling points, the smaller the sampling size, the fewer differences there are between the different observed data timeliness measures on average [Ren19] (lower variation). However, we observe the most inferior variations for the case scenario *CS1*. Indeed, in this scenario, the production and insertion rates are static. Thus, the actual data timeliness value is preserved throughout time (i.e., same level), and so are the measures on average using TUTOR. Moreover, sampling using the random sampling frequency provides more variation than sampling with the static sampling frequencies 30s and 70s enlarging the spectrum of the observed data timeliness measures. Therefore, TUTOR can get closer to the actual data timeliness level (measured at the source level), which we believe may enhance its measuring accuracy.

Our analysis regarding data timeliness variation for the choice of the sampling frequency must be backed up using the results obtained to evaluate the effectiveness and efficiency of TUTOR in detecting updates presented hereafter.

Effectiveness & efficiency to detect updates As above-said, the ability of $TUTOR$ to detect updates in an optimized way using the different sampling frequencies for the other case scenarios is of high importance. To do so, we calculate the success rate and detection rate for each sampling frequency for the different case scenarios. Then, we analyze and compare the results. Using the obtained results for the evaluation of data timeliness of average previously discussed, we select the optimal sampling frequency method for $TUTOR$ (static or random).

S_f	N	X_S	$RealX_S$	S_R	D_R
$30s$	2940	1773	1800	0,99	0,60
$70s$	1260	1260	1800	0,70	1
$150s$	594	594	1800	0,33	1
$[5, 150]s$	1145	994	1800	0,55	0,87

Table 4.7: CS1 : Update Detection Evaluation

Table 4.7 presents the evaluation results for *CS1*. According to the results, the success rate for the sampling frequency $30s$ is the highest with a value of 99% and decreases when the static sampling frequency is bigger, unlike the detection rate, which is higher when the sampling frequency is bigger with a minimum value of 60%. Concerning the random sampling frequency, the success rate is around 53%, and the detection rate is around 87%.

Table 4.8 presents the evaluation results for *CS2*. According to the results, the success rate for the sampling frequency $30s$ is the highest with a value of 97% and decreases when the static sampling frequency is bigger, unlike the detection rate, which is higher when the sampling frequency is bigger with a minimum value of 35%. Concerning the random sampling frequency, the success rate is around 94% (second-best), and the detection rate is around 87%.

S_f	N	X_S	$RealX_S$	S_R	D_R
$30s$	2923	1030	1059	0,97	0,35
$70s$	1260	902	1054	0,85	0,72
$150s$	593	593	1043	0,57	1
$[5, 150]s$	1145	994	1057	0,94	0,87

Table 4.8: CS2 : Update Detection Evaluation

Table 4.9 presents the evaluation results for *CS3*. According to the results, the success rate for the sampling frequency $30s$ is the highest with a value of 97% and decreases when the static sampling frequency is slower. Concerning the random sampling frequency, the success rate is around 36% (second-best), and the detection rate is around 93%.

Table 4.10 presents the evaluation of the trade-off on average, taking into consideration all case studies for different configurations of the weight/importance we give to the success rate and detection rate. The objective is to show the variation of the choice of the sampling method for the various users' needs w.r.t S_R and D_R. When we move from left to right, we emphasise the success rate S_R (γ) and less the detection rate D_R (θ).

S_f	N	X_S	$RealX_S$	S_R	D_R
$30s$	2066	1302	1305	0,99	0,63
$70s$	906	906	1305	0,69	1
$150s$	427	427	1305	0,33	1
$[5, 150]s$	817	714	1305	0,55	0,87

Table 4.9: CS3 : Update Detection Evaluation

S_f	***θ=1,γ=0***	***θ=0,7,γ=0,3***	***θ=0,5,γ=0,5***	***θ=0,3,γ=0,7***	***θ=0,γ=1***
$30s$	0,52	0,66	0,75	**0,85**	**0,98**
$70s$	0,91	**0,86**	**0,83**	0,79	0,75
$150s$	**1**	0,81	0,70	0,58	0,41
$[5, 150]s$	0,87	0,81	**0,77**	0,74	0,68

Table 4.10: Evaluation of Trade-off on Average for all CSs

The results show that when we give only importance to D_R (i.e., efficiency), the biggest static S_f value ($150s$) provides the highest trade-off (T_o=1). When less importance is given to D_R (from left to right), the highest trade-off is obtained for the smallest static S_f values. With S_R and D_R pondered equally: (1) static frequency sampling, T_o between 0,66 and 0,86 (average of 76%), (2) random frequency sampling, T_o is equal to 0,77. In e-health scenarios like those proposed by the project SUMMIT, detecting database updates (i.e., effectiveness) should be done with the lowest sampling cost (i.e., efficiency). Therefore, equal importance is given to S_R and D_R. For more insights, we take a closer look at this case in table 4.11 presented below.

Table 4.11 presents the evaluation of the trade-off for each case study when we give equal importance to the effectiveness and efficiency of *TUTOR* ($\gamma = \theta = 0,5$). The last column presents the average trade-off for all case studies. The objective is to analyse in more depth the trade-off when we give equal importance to S_R and D_R for each *CS*, and then for all *CSs*.

The table show that for *CS1* and *CS3*, when the insertion rate is static, S_f of $70s$ has the highest T_o equal to 85%. For *CS2* when the insertion rate is dynamic, the random S_f has the highest T_o equal to 90%. For all *CS*s, S_f of $70s$ has the highest T_o followed by the random S_f.

Considering and analyzing these results, we notice the following.

- The results show that using a static sampling frequency is tricky (i.e., sensitive) and difficult for the static sampling frequencies. The reason is that there is a considerable difference between the obtained success rates and detection rates for the studied static sampling frequencies values for the multiple case scenarios. The difference in the detection rate (respectively the success rate) through the different selected static frequencies reaches ± 0,65 % (respectively ± 0,66 %), which is significant.

	CS1			CS2			CS3			All
S_f	S_R	D_R	T_o	S_R	D_R	T_o	S_R	D_R	T_o	avg(T_o)
$30s$	0,99	0,60	0,79	0,97	0,35	0,66	0,99	0,63	0,81	0,75
$70s$	0,70	1	**0,85**	0,85	0,72	0,78	0,69	1	**0,85**	**0,83**
$150s$	0,33	1	0,66	0,57	1	0,78	0,33	1	0,66	0,70
$[5,150]s$	0,55	0,87	0,71	0,94	0,87	**0,90**	0,55	0,87	0,71	**0,77**

Table 4.11: Results: Trade-off Evaluation ($\gamma = \theta = 0,5$)

– The success rate of *TUTOR* increases when S_f is faster (smaller) because it is able to detect more updates (see figure 4.12).

– The detection rate of *TUTOR* increases when S_f is slower (bigger). It is more likely that at least one insert has been performed on the database between two performed sampling points. Therefore, the chances increase for *TUTOR* to detect a database update when it samples data (see figure 4.12).

  As a result, when the sampling frequency is superior to the insertion rate, the detection rate always tends to 100%.

In fact, without any knowledge about the data insertion rates, the static sampling frequency value is chosen w.r.t the validity duration: the selected value would be higher or lower than the validity duration. Intuitively, the guess would be that as the selected static sampling frequency value becomes small, one should select more updates. Our results proved this intuition. However, they also demonstrated that the detection rate decreases when the insertion rate increases.

In this case, as discussed in section 4.4.2, importance levels must be set to both the success rate and detection rate: whether we are more interested in having a high S_R or a high D_R (table 4.10). Suppose we are more interested in having a higher success rate. In that case, we must select the smallest possible static sampling frequency value and risk accessing the service more than necessary (low detection rate). Suppose we are more interested in having a higher detection rate. In that case, we must choose the highest static sampling frequency value and risk missing updates (low success rate)(see figure 4.12).

- For the random sampling frequency, our results showed that whatever the presented case scenario, and that no matter the importance we attribute to the detection rate and success rate, *TUTOR* at worst reaches the trade-off of 68% of efficiency and effectiveness (table 4.10).

Comparing both methods revealed that sampling with a random sampling frequency for *TUTOR* is preferred for detecting updates. Indeed, considering all *CS*s and no matter the importance we attribute to the detection rate and success rate: while the trade-off variate between 52% and 98% for S_f=30s (± 46 %), variate between 75% and 91% for S_f=70s (± 16 %), and variate between 52% and 98% for S_f=150s (± 46 %) (table 4.10), T_o is satisfactory when S_f is random as it does not variate much (± 11 %).

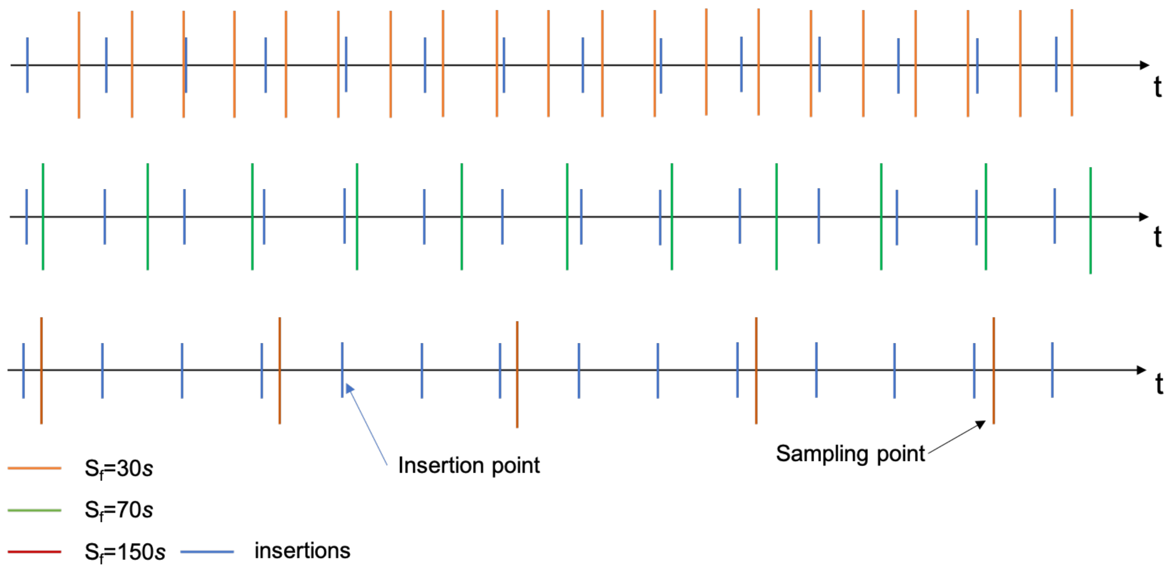


Figure 4.12: Illustration *CS1*: Sampling points for the static S_f

Results analysis: Data timeliness & ability to detect updates Combining the analysis of *data timeliness variation* results with the *ability to detect updates*, the sampling with a random S_f for *TUTOR* provides the most accurate observations and timeliness measures. This result is the closest to the real world and reduces the overhead induced by repeatedly accessing data sources for monitoring purposes.

To sum up, in this section, we have experimented and proved the feasibility and pertinence of our protocol *TUTOR* that is proposed for observing data quality for black-box medical data services. Indeed, we have concluded that the choice of random sampling frequencies appears to be more effective and efficient for detecting the maximum of database updates and measuring data timeliness at the lowest cost (i.e., minimal number of access times). Based on TUTOR, we validate our proposed data quality evaluation model in the next section.

4.5 TUTOR-based Data Quality Evaluation Model

Data quality is evaluated focusing on data freshness (section 4.2) which is defined as the product of data timeliness (related to how often data are produced) and database timeliness (related to how often data are inserted into the database). To validate our data quality evaluation model presented in section 4.2, we use our observability protocol *TUTOR* with a random sampling frequency S_f. Indeed, we experiment with this model for the same e-health scenario and test the effectiveness of our solution in ranking black box data services according to their evaluated data quality level.

This section is organized as follows. First, we present the experimental setting of our solution based on *TUTOR*. Then, we define the used evaluation metrics for ranking data services. We continue by discussing and analyzing the obtained results using these metrics. Finally, we experiment with the performance of our solution in terms of response time.

4.5.1 DQEM : Experimental setting

As in section 4.4, we have used the docker technology to deploy three *HAPI FHIR* servers: one simulating the hospital's server, one simulating Alice's smartphone' server and the last one simulating the SOS server of the hospital. Each server has its independent database on the corresponding server and is reachable through its URL. Note that these servers are configured the same way in our controlled environment.

To give access to these servers, we have deployed ten data services with a fixed production rate and 6 different update frequencies, including static and dynamic (i.e., random) frequencies (illustration of CS1 and CS2). Each service has its access point. We deployed four data services giving access to the four devices used by Alice are deployed on the first and the third *HAPI FHIR* servers giving us a total of eight data services. Two data services are providing access to the second *HAPI FHIR* server. Likewise the past experiment, the validity interval is set to $60s$.

	$8s$	$15s$	$50s$	$[10, 120]s$	$[30, 200]s$	$300s$
S_{11}, S_{21}, S_{32}	*					
S_{12}						*
S_{13}, S_{31}		*				
S_{22}, S_{33}			*			
S_{14}					*	
S_{34}				*		

Table 4.12: e-heathcare scenario: data services update frequencies

Table 4.12 presents the chosen update frequencies for those 10 services in a way that they have different data quality levels w.r.t our experimental setting: (1) all data services have the same data production rate ($3s$), (2) Data services S_{11}, S_{21} and S_{32} have the highest insertion rate equal to $8s$ (7 insertions per data validity interval), (3) followed by data services S_{13} and S_{31} with an insertion rate of $15s$) (up to 4 insertions per data validity interval), (4) data services S_{22} and S_{33} with an insertion rate of $50s$ (1 insertion per validity interval), (5) data service S_{34} with a random insertion rate in the interval of $[10, 120]s$ (5 insertions per validity interval to one per two validity intervals), (6) data service S_{14} with a random insertion rate in the interval of $[30, 200]s$ (5 insertions per validity interval to one per five validity intervals) and last, (7) data service S_{12} has the lowest insertion rate equal to $300s$ (1 insertion per 4 or 5 validity intervals). The choice of the static values of insertion rates is motivated by the need to deploy services that access timely data (i.e., all insertion rates $<$ validity duration) with different levels of timeliness (S_{11}, S_{21}, S_{32}, S_{13}, S_{31}, S_{22}, S_{33}) and other services that access data with very low timeliness (i.e., insertion rate $> >$ validity duration) (S_{12}). The choice of the random values of insertion rates is motivated by the need to deploy services that can provide timely data or data with low timeliness with a larger (S_{14})/narrower(S_{34}) random interval.

According to this configuration, the following ranking is expected: first, S_{11}, S_{21} and S_{32} have higher data quality as they have the highest data insertion rate that is faster than the validity duration. Then S_{13} and S_{31} and then, S_{22} and S_{33}, followed by S_{34} and then and S_{14} because their insertion rates may be faster or slower than the validity duration and finally, S_{12} (see table 4.13).

Table 4.13: Services data quality-based ranking: expectation

Data Quality
S_{11}, S_{21}, S_{32}
S_{13}, S_{31}
S_{22}, S_{33}
S_{34}
S_{14}
S_{12}

This setting is used twice to represent two instances of $TUTOR$ referred to as $TUTOR_1$ and $TUTOR_2$, launched in parallel, each with a different random sampling frequency interval. The objective is to see if the choice of the random interval (bigger or smaller) would significantly affect the effectiveness of our solution in ranking data services according to their data quality levels.

$TUTOR_1$ observes data services with the smallest/narrowest sampling frequency interval $of f_1$ $= [3, 70]s$ (so that samplings are performed multiple times during a validity interval), followed by $TUTOR_2$ which observes data services with a sampling frequency in the interval of $f_2 =$ $[5, 150]s$ (possibility of sampling multiple times per validity interval to once per two validity intervals). The choice of these frequencies is to compare when we access a data service multiple times during a validity interval versus fewer times. For these two instances, the random sampling frequencies intervals are defined using an integer random seed which enables $TUTOR$ to create reproducible streams of random integer numbers and from which $TUTOR$ selects a random number. Indeed, for each sampling point, $TUTOR$ selects a random number within the generated stream by the seed. By default, when using a small seed interval as the case with TUTOR, the probabilities of generating the same seed and selecting the same values within the seed are high [3]. Therefore, $TUTOR$ is programmed to regenerate a new stream of random numbers after each sampling point by initializing the seed twice.

Once all the experimental setting is ready, and to test our solution, 50 data quality requests (i.e., service selection request) were carried out over twelve hours to the *DQEM*. At the same time, a series of manipulations (Mi) has been performed on all the 10 candidate services (i.e.,

[3]`https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.random.RandomState.html`

Modification of their setting), which aim to affect their final ranking. Therefore, a python test script is developed to choose randomly at an instant t (1) a data service and (2) a manipulation for this service among the following.

- **M0.** Starting a data service with its initial configuration as described above. The purpose is to check the effectiveness of our solution to detect changes in the activity of data services (from inactive to active). We expect active services to be ranked as initially illustrated in table 4.13.
- **M1.** Starting a data service with reduced data timeliness: Part of the inserts that the service performs contain outdated data (already have been inserted before in other previous inserts). This manipulation aims to decrease the data timeliness of data accessed by the corresponding data service to half while keeping the same level of database timeliness. According to our algorithm, its inserts would still be considered as a database update (U_{DB}=1). However, we expect the ranking of the service to decrease. They are still providing part of fresh data that are still in the validity duration.
- **M2.** Starting a data service with null data timeliness: All data inserted by the corresponding data service are outdated and different from the last inserted data: inserts of data captured longer than 60 seconds ago. This manipulation aims to decrease the data timeliness to zero while keeping the same level of database timeliness (U_{DB}=1). We expect the service to have a null data quality level and, thus, ranked last.
- **M3.** Stopping a data service. The purpose is to check the effectiveness of our solution to detect inactive data services. We expect inactive services to be ranked last since they are unavailable, and as expected, their data quality levels are zero.

These manipulations are launched with a random duration that separates them. As a result, following a data quality request, the *DQEM* outputs a ranked list of the available services according to their data quality (i.e., data freshness) level. The ranking effectiveness is evaluated using an evaluation metric presented in the next section.

4.5.2 Ranking Effectiveness Evaluation Metric

The objective of our solution is to rank services according to their data quality level and, thus, select the service with the highest data quality level. Therefore, the Normalized Discounted Cumulative Gain (NDCG) metric [JK02] is employed. NDCG metric, used for evaluating ranking results, is mainly used for information retrieval problems. It measures the effectiveness of a given search engine by ranking the articles it displays according to their relevancy in terms of the search keyword (used by Google). In essence, given an ideal data quality based service ranking (ground truth) and an obtained data quality based service ranking results, NDCG for the top-k services is defined in $[0, 1]$ as follows:

$$NDCG_k = \frac{DCG_k}{IDCG_k} \tag{4.18}$$

Where $IDCG_k$ is the Ideal Discounted Cumulative Gain and DCG_k is the Discounted Cumulative Gain and is defined as follows:

$$DCG_k = \frac{sum_{n=1}^{i=k} rel_i}{\log_2(i+1)} \tag{4.19}$$

Where rel_i is the relevance scale of the data service ranked at the level i. The ideal rank gets the highest gain among the different rankings.

Let us consider our DQEM solution for ranking 10 data services according to their data quality (S_{11}, S_{12}, S_{13}, S_{14}, S_{21}, S_{22}, S_{31}, S_{32}, S_{33}, S_{34}). The assignment of the relevance score is subjective and generally quantifies the pertinence of the retrieved information concerning the requester's needs. This requires the knowledge of the expectations of the user and her preferences.

Existing search engine solutions usually apply machine learning algorithms for learning requester's patterns using their past research experience to assign those relevance scores. However, at this level of our work, DQEM faced a *cold-start* problem as the expectations of the requesters are unknown. Therefore, in a white-box setting, we assign relevance scores according to our knowledge about the services' background and their data quality setting. Since we have 6 different insertion rates, the relevance scale is initially set in $[1, 6]$, where a higher value stands for better data service. We attribute the scale *0* for the non-relevant at all (i.e., data services that are not functioning or have a null data quality level). For instance, consider two data services, one refreshing its database twice per validity interval and the other once per validity interval. We can conclude that the data provided by the first one is more representative of the actual real-world state. Thus, more relevant for the task at hand.

Therefore, services are regrouped per relevancy score to obtain 7 categories of services, including the category that regroups data services with zero relevance as follows:

- S_{11}, S_{21}, S_{32} : 6
- S_{13}, S_{31} : 5
- S_{22}, S_{33} : 4
- S_{34} : 3
- S_{14} : 2
- S_{12} : 1

For each of the manipulations performed by our test script, the relevance scores of the Top-10 data services change as presented in table 4.14. Subsequently, the $IDCG_{10}$ measure changes accordingly. The strategy is as follows: if the *data timeliness* of a data service is decreased but its database timeliness remains the same (manipulation M1), then this data service is

	M0	**M1**	**M2**	**M3**
S_{11}, S_{21}, S_{32}	6	5	0	0
S_{13}, S_{31}	5	4	0	0
S_{22}, S_{33}	4	3	0	0
S_{34}	3	3	0	0
S_{14}	2	1	0	0
S_{12}	1	1	1	1

Table 4.14: Top-10 Data services' relevance score for the different manipulation

downgraded and moved down to another category. In our work, we chose to downgrade them by one grade. Also, if the *data timeliness* of a data service is set to zero (M2) then this data service is downgraded and moved down to the down-bottom category with a relevancy scale of zero.

Moreover, we use a second evaluation metric $\Delta NDCG$ to quantify the added-value of the *data timeliness* metric. $\Delta NDCG$ is defined as follows:

$$\Delta NDCG_{\Delta t,Mi} = \sum_{t=cts-\Delta t}^{t=cts} \delta NDCG_{Mi} \tag{4.20}$$

where Δt is the evaluation duration of our experiment in seconds, *cts* is the current timestamp, and $\delta NDCG_{Mi}$ is defined for a single request r_j as follows:

$$\delta NDCG_{Mi} = NDCG_{DQ,rj} - NDCG_{DB,rj} \tag{4.21}$$

where $NDCG_{DQ,rj}$ is the $NDCG$ score obtained by ranking services according to their obtained data quality level and $NDCG_{DB,rj}$ is the $NDCG$ score obtained by ranking services according to their obtained database update frequency level. Indeed, our objective is to demonstrate the added value of the data timeliness metric in our model. The effectiveness of ranking results obtained using the evaluated data quality (considering both timeliness metrics) is compared to the effectiveness of ranking results obtained using only the evaluated database timeliness level.

We validate our TUTOR-based data quality evaluation model presented in the next section based on these evaluation metrics.

4.5.3 DQEM: Results & Discussion

Given our objective to rank data services according to their data quality levels, tests are performed using the configurations described above to verify TUTOR-based *DQEM*, our ranking solution. To do so, 50 requests r_j (r_0, r_2 .. r_{49}) are launched (represented on the horizontal axis of the following figure)) using two different configurations for TUTOR's sampling frequency ($TUTOR_1$ and $TUTOR_2$) and observe the obtained results.

The resulting $NDCG$ levels for the different requests are presented in figure 4.13 for $TUTOR_1$ and in figure 4.14 for $TUTOR_2$.

These figures depict the $NDCG$ level for data quality-based ranking and database timeliness-based ranking. The objective is to see the added-value and importance of the *data timeliness* metric in the proposed data quality evaluation model. Note that both configurations are run independently and, thus, with a different set of random manipulations. This consolidates the interpretation of the results for this objective.

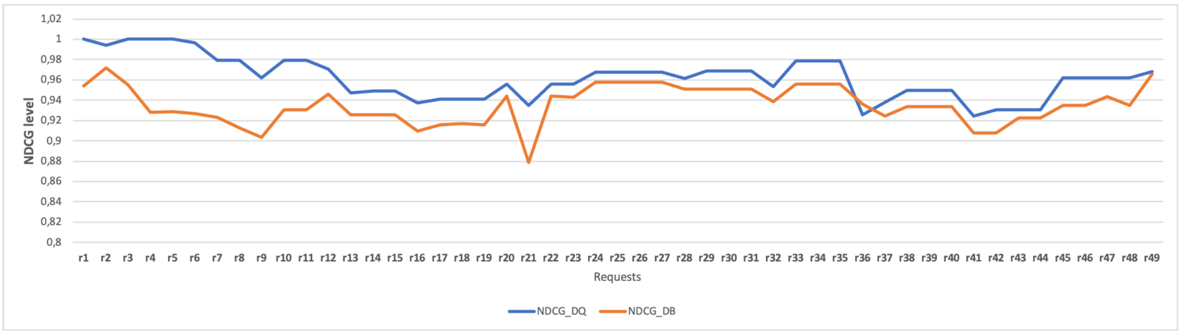


Figure 4.13: $TUTOR_1$: NDCG variation for services' ranking according to DQ and DB

According to figure 4.13, we can see that the obtained $NDCG$ scores through data quality ranking are between 0,93 and 1, which means that our solution using the sampling frequency interval f_1 is at worst 93 % effective. We also see that the $NDCG$ scores obtained through ranking services using their data quality (DQ) level $NDCG_{DQ}$ are always better than the $NDCG$ scores obtained through ranking services using only their database timeliness (DB) level $NDCG_{DB}$.

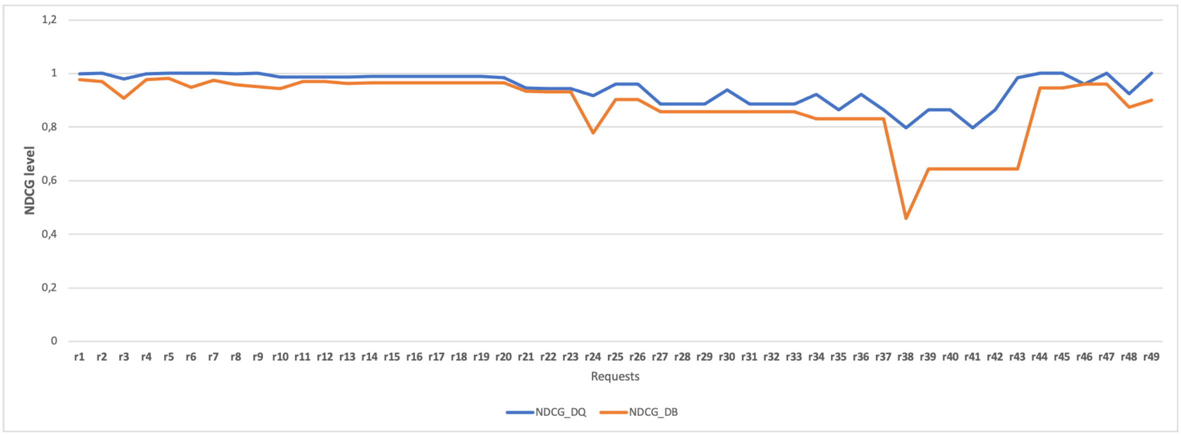
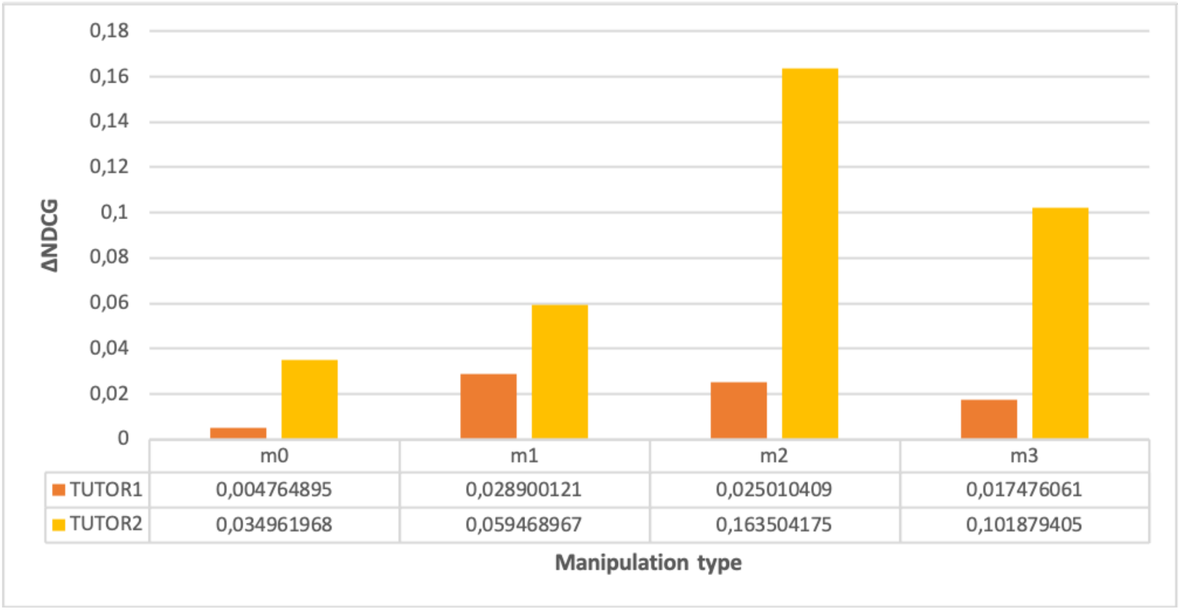


Figure 4.14: $TUTOR2$: NDCG variation for services' ranking according to DQ and DB

According to figure 4.14, we can see that the obtained $NDCG$ scores through data quality ranking are between 0,8 and 1, which means that our solution using the sampling frequency interval f_2 is at worst 80 % effective. We also see that the *NDCG* scores obtained through ranking services using their data quality level is always better than the *NDCG* scores obtained through ranking services using only their database timeliness level.



	m0	m1	m2	m3
🟧 TUTOR1	0,004764895	0,028900121	0,025010409	0,017476061
🟨 TUTOR2	0,034961968	0,059468967	0,163504175	0,101879405

Figure 4.15: $\Delta NDCG$ level per manipulation type for *TUTOR1* & *TUTOR2*

Figure 4.15 depicts the measured $\Delta NDCG$ level per manipulation m_i for $TUTOR_1$ and $TUTOR_2$. According to the obtained results for $TUTOR_1$, we can see that $\Delta NDCG$ level is bigger for manipulations m_1 and m_2 which consists of setting the data timeliness of a service to zero. $\Delta NDCG$ is equal to 2,5% for m_2. According to the obtained results for $TUTOR_2$, $\Delta NDCG$ is bigger for manipulation m_2 which is equal to 16%. These results demonstrate the added-value of data timeliness to the data quality model as the impact on the effectiveness of ranking data services is apparent and higher for manipulation m_2 than m_1 than m_0.

The observed results in the above figures validate our data quality evaluation model by showing the added-value of the *data timeliness* metric to the database timeliness as the ranking results using the data quality level are always better than the ranking results using only database timeliness. Indeed, using only information about the update frequency of data services is insufficient because a data service that often updates its database does not ensure that the inserted data are fresh.

Figure 4.16 compares the obtained *NDCG* results using the data quality level for $TUTOR_1$ and $TUTOR_2$. We recall that the objective is to see the effect of a larger random interval for the sampling frequencies on the effectiveness of *DQEM* in ranking data services. Therefore, for this experiment, both instances of TUTOR are run simultaneously using the same set of manipulations to enable their comparison.

According to this figure, we can see that the obtained *NDCG* scores through using the sampling frequency interval f_1 ($NDCG_{DQ1}$) are better than the obtained $NDCG$ scores through using the sampling frequency interval f_2 ($NDCG_{DQ2}$). Moreover, during the experiment's execution duration, results demonstrated stability starting from the seventh request over 21 requests.

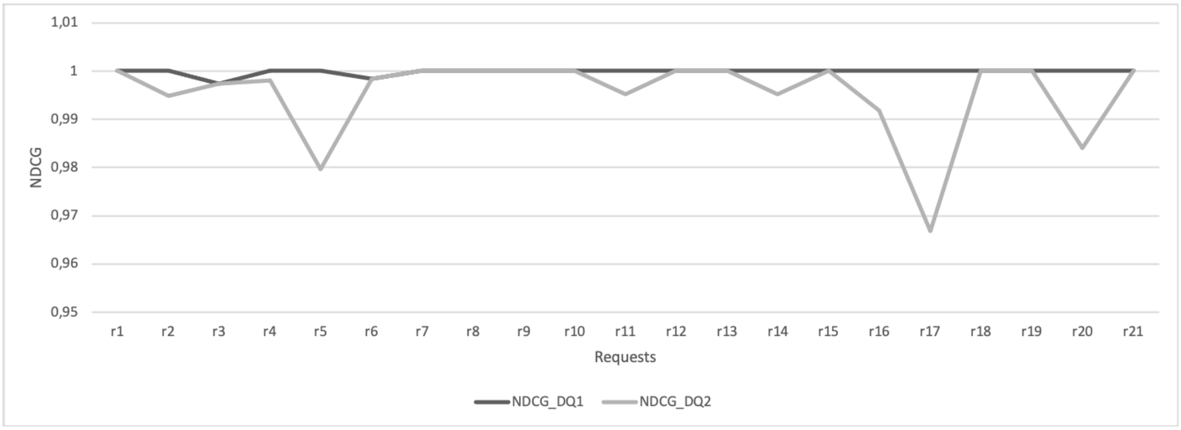


Figure 4.16: $TUTOR_1$ VS $TUTOR_2$: $NDCG_{DQ}$ variation for services' ranking

We can conclude that using a smaller sampling frequency interval is better for ranking results, especially when working with highly changing data (stocks trading etc.). The larger the spectrum of the sampling frequencies (i.e., larger sampling interval), the bigger the duration can get between the different sampling points. Therefore, $TUTOR$ may miss some updates between two sampling points (lower success rate) as has been pointed out in section 4.4.2. Indeed, when the duration between two sampling points is more significant than the insertion rate of the service, some updates are undetected.

Now that our data quality evaluation model is validated using $TUTOR$, we move to test the performance of our solution.

4.5.4 DQEM: Computation Time Evaluation

We evaluate our TUTOR-based data quality evaluation module regarding computation time overhead.

To this end, an experiment is conducted to compute the processing time by *DQEM* to provide a list of ranked data services according to their data quality level. This requirement implies invoking the ***data quality evaluation*** function, which in turn invokes the ***update frequency evaluation*** function.

This processing time is compared to the processing time needed to respond to a request by randomly selecting one of the available data services. In other words, we are responding to a request without the invocation of our *DQEM*.

For our experiments, we incrementally increase the number of the parallel incoming requests to both our *DQEM* ranking solution and the *random-based selection* solution.

The simulation is run for 60 seconds, during which two sets of experiments are conducted: (1) incrementing the number of parallel incoming requests by 100 (figure 4.17), and (2) incrementing the number of parallel incoming requests by 500 (figure 4.18). The objective is to test and compare the performance of both solutions when there is respectively a low and high number of incoming requests.

According to figure 4.17, the maximum processing time for both solutions is reached when sending 200 parallel requests, decreases for 300 parallel requests and then increases slowly at each time we increment the number of incoming requests. The expansion of the number of incoming requests with a step of 100 increases the processing time on average by $80ms$.

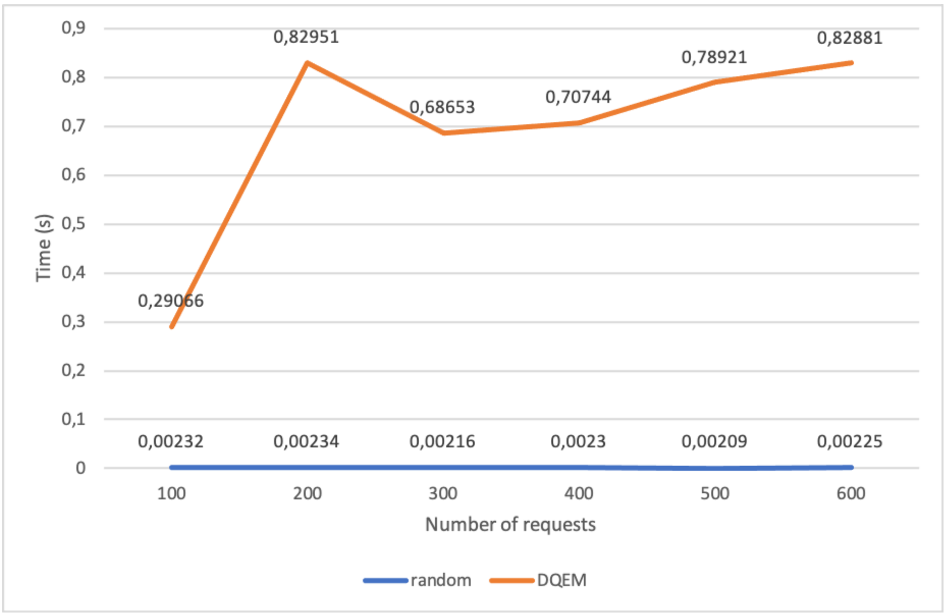


Figure 4.17: Computational time cost of DQEM VS random selection - 600 requests

According to figure 4.18, the processing time for both solutions increases slowly when continuously incrementing the number of parallel requests. However, the difference is more apparent for our solution and reaches $8{,}747s$. The increase of the number of incoming requests with a step of 500 raises the processing time on average by $100ms$.

Thus, and according to both figures, we can observe that the processing time of the *random-based selection* solution is always better than our *DQEM* ranking solution. Moreover, when increasing the number of incoming requests, the rise of the processing time (in a matter of milliseconds) is more significant for our solution. However, the increase of the processing time per 100 steps or 500 steps is more or less the same ($80ms$ and $100ms$ respectively). This lets us conclude that the induced overhead compared to the random solution is only caused by the experimental settings that execute the entire processing steps of *DQEM* on a limited local execution environment. Indeed, the added overhead (figure 4.19) is induced by *TUTOR* (1) when sampling data, computing timeliness metrics and evaluating data quality by (2) accessing **EDQ** database (where data quality levels are stored), pulling these levels, and ranking data services. Moreover, the results demonstrate that the increase of the number

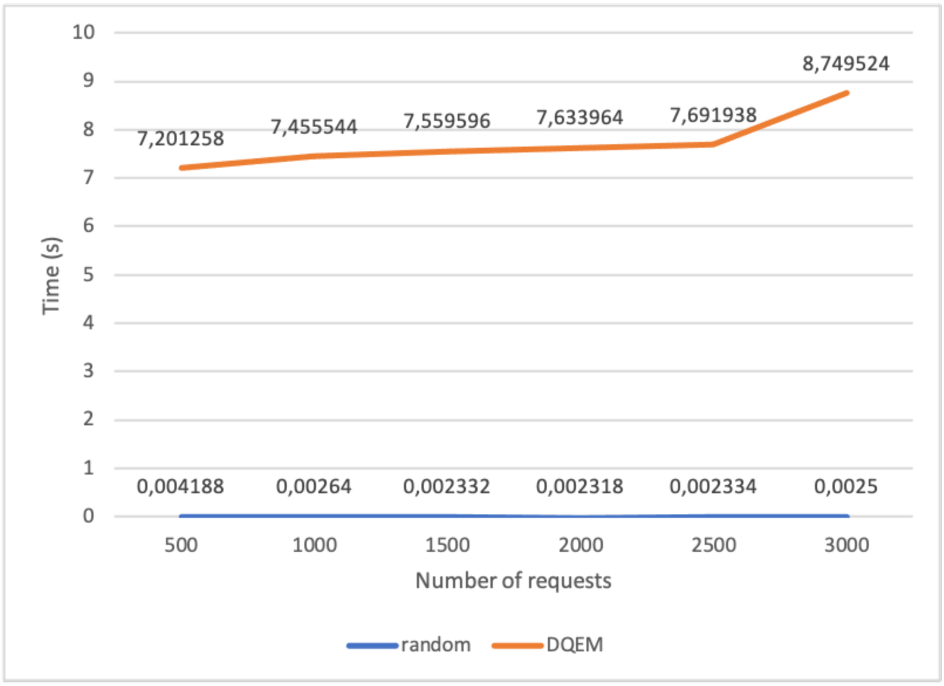
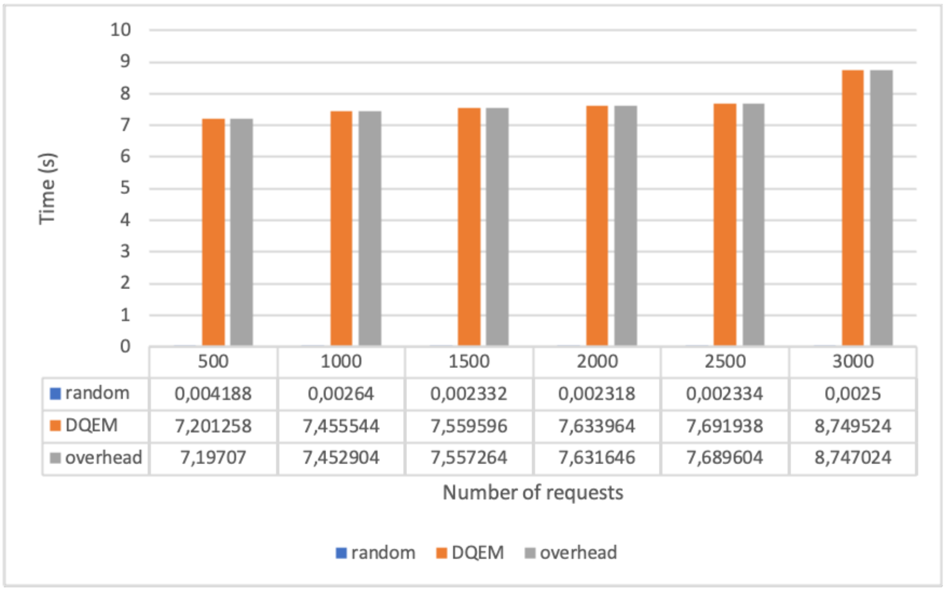


Figure 4.18: Computational time cost of DQEM VS random selection - 3000 request

of incoming requests does not affect the computation time of *DQEM*.



	500	1000	1500	2000	2500	3000
🟦 random	0,004188	0,00264	0,002332	0,002318	0,002334	0,0025
🟧 DQEM	7,201258	7,455544	7,559596	7,633964	7,691938	8,749524
◼ overhead	7,19707	7,452904	7,557264	7,631646	7,689604	8,747024

Figure 4.19: Computational time overhead

4.6 Conclusion

4.6.1 Summary

This chapter answers the first research question of this thesis, namely: *"How to evaluate the quality of stream data focusing on their freshness when data are accessed using black box data services ?"* for which we have defined a data quality evaluation model for black box data services using the *data freshness* as an evaluation factor and *timeliness* metrics. Moreover, we have proposed a data quality observability protocol *TUTOR* to capture the necessary information for this evaluation model. For *TUTOR* we proposed a knowledge database, an observability algorithm, and an update frequency measuring algorithm. These observations are used to define the *timeliness* metrics including *data timeliness* and *database timeliness.* Also, we have presented the experimental setting and results for validating *TUTOR* using *HAPI FHIR* medical data services within the context of the project SUMMIT. Indeed, we have demonstrated the effectiveness and the efficiency of *TUTOR* in detecting updates and evaluating data timeliness for two sampling methods (static and random sampling). Next, we have validated our proposed data quality evaluation model using *TUTOR* through experiments by demonstrating its ability to rank data services according to their changing data quality levels. Finally, we have evaluated the performance of our solution in terms of processing time.

The above-described contributions and results have highlighted three types of limitations, mainly concerning the choice of the sampling frequency for *TUTOR*, the frequency of change evaluation, and scaling our solution.

4.6.2 Limitations & Enhancement ideas

The choice of *TUTOR*'s sampling frequency. As aforementioned, the choice of the sampling frequency is dependent on the studied 4 case scenarios. In this chapter, the sampling frequency method (static, random) is selected for the current experimental setting (choice of production and insertion rates) using the observed data timeliness on average and the effectiveness and efficiency of TUTOR in detecting updates. The limitation of this experiment is that the application of the evaluation function *average* for the observation of data timeliness variation alone does not give much insight about what sampling frequency to choose. The *average* attenuates the observations, and thus, the obtained results are more or less homogeneous. This may hide significant fluctuation (pics) of the measured data timeliness values (min or max), which may give more insights.

To address this limitation, it is possible to consider other statistical measures to observe data timeliness and even experiment for more case scenarios with various settings (choice of production and insertion rates) in future work.

Moreover, we noticed during the experiments that there are specific patterns w.r.t the effect of the choice of the sampling on the sampling effectiveness and efficiency listed as follows:

- When the obtained $D_R = 1$, then the insertion rate is smaller than S_f.
- When the static S_f is decreased (smaller) and the obtained S_R increases, then the insertion rate is smaller than S_f.
- When the evaluated S_R per period remains more or less the same and the variation on average at the data timeliness level is low, then the insertion rate is static.

These patterns could be further studied, validated, and used as indicators that help calibrate the sampling frequency on the fly and enhance the effectiveness and efficiency of TUTOR in detecting updates. Indeed, such indicators help develop an intelligent sampling protocol that calibrates its sampling frequency using machine learning techniques. For instance, at an instant t, TUTOR computes the sampling effectiveness and efficiency levels per period, analyzes the situation considering S_f, and deduces that results fit into the first pattern. Thus, TUTOR must decrease S_f for the next samplings since it would likely generate better effectiveness.

Frequency of change evaluation. Analytical studies on estimating the update distribution under blind sampling have all assumed the data source update, especially web pages and resources (open data portals) update their data (i.e., pages) following a Poisson distribution [Tia+20]; [CN02]; [Jin+18]; [CGM03]. Based on this hypothesis, they evaluate the update frequency using Poisson estimators. In this experiment, we focused on the feasibility and the pertinence of TUTOR and selecting a sampling method. Due to time constraints, we could not experiment and validate whether data services/databases updates follow a Poisson distribution through research. Therefore, we used a naive estimator to evaluate the frequency of updates of a database.

As future work, experiments can be performed on data services (e.g., tweeter data services) to check the distribution model of updates (tweets), their regularity, and if they follow a Poisson distribution. Accordingly, we believe that our solution can be enhanced by using other estimators for the evaluation of the update frequency.

DQEM scalability. Currently, as described in section 4.5, our solution only ranks 7 candidate data services. Moreover, each service is assigned a relevance score as explained in section 4.5.2. Indeed, we have defined a few rules for assigning and modifying the relevance scores attributed to the different candidate data services w.r.t the performed manipulation, which affects their data timeliness. Accordingly, candidate services are regrouped per 7 categories according to their insertion rates (see table 4.12) and can be then moved from one category to another. For instance, we configured a rule that states that when the data timeliness is reduced to half, the service is downgraded by one category grade. These configurations (i.e., assigning a relevance score to services, categorizing services, changing their categorization upon a manipulation) are only suitable for a small number of services and manipulations, limiting our solution's scalability even though this scaling remains dependent on the application system.

As future work, this solution can be scaled in terms of the number of candidate data services, the number of configurations linked to the choice of the production and insertion rates, or the number of manipulations (currently, there are 4). Accordingly, the attribution of scores, the categorization of services, and the rules for each manipulation must be revisited and adapted.

Ideally, when all scaling possibilities are considered, we can develop an intelligent test script that can be scaled in terms of the number of services and configured in terms of rules and the number of categories.

Chapter 5

Trust-based Black Box Data Services' Selection

Contents

"Trust, but verify."

Ronald Reagan

5.1 Introduction

In the previous chapter, a data quality evaluation model for data services as a black box has been defined focusing on the data freshness dimension. To the necessary information for this evaluation, an observability protocol has been proposed to enable "guessing" the frequency of update of a service's database, and thus, the evaluation of the database timeliness, and consequently, data freshness.

This chapter proposes a trust evaluation model for data services as a black box. The evaluation of services' trustworthiness has been broadly discussed in the literature in various service environments. However, the current literature review appears to validate the lack of solutions for modeling the trust in data services considering data trust and service trust simultaneously. Therefore, our model is proposed to select among proposed data services deployed in a service environment, the best one that fits better the user needs both in terms of performance and data quality. The proposed trust evaluation model combines the QoS factor including performance and the QoD factor including data freshness as defined in the previous chapter.

The chapter is organized as follows: (1) we define data service trust factors, (2) we then propose the model to combine QoS and QoD, and (3) present which metrics are used to evaluate QoS including performance. Our proposed model is then validated thanks to *DETECT*, our trust evaluation architecture, and the trust evaluation is applied to our e-health scenario to demonstrate its feasibility in ranking data services according to their trust level. Moreover, we demonstrate the effectiveness of our solution to satisfy users' requirements w.r.t performance and data quality. Finally, we summarize the content of this chapter.

5.2 Data Service Trust Evaluation Model

Service providers propose data services under heterogeneous quality of service (QoS) conditions to support various users' needs related to scalability, fast response time, availability, etc. QoS varies from one service provider to another, from the same service provider at several moments (e.g., from day to day), or even from various applications according to the application domain. As a result, services offer different QoS levels that may fluctuate over the time, and thus, do not allow to rely on the agreed quality [1]. Therefore, measuring and monitoring QoS contributes to chose a service that corresponds to users' needs.

Moreover, data services are accessing various data sources that are continuously updated by new data whenever data dynamically evolve in the real world. Indeed, as discussed in chapter 4, stream data services update their data with heterogeneous rates[2] that influence the freshness of the data accessed by a given service. For example, given a user that requires data captured within the last $60s$, a service whose database is updated each $10s$ is more likely to provide fresher data than a service whose database is updated each $60s$. Thus, it is necessary to provide also a measure that can serve to determine to which extent data accessed by a given service are up-to-date.

To this end, when we seek a data service from a vendor, we consider two factors. First, the past experiences with the data service giving an overview of its performance. Second, we look at the quality of the data delivered by the data service focusing in this work on data freshness. Consequently, to reflect the users' needs, trust level evaluation of a data service should be

[1]Generally, it is provided in their SLA

[2]Note that rates include the production rates and insertion rates as seen in section 2.3.2

based on two factors: *performance* and *data quality*, and is defined in $[0,1]$ as follows:

$$T_{DS} = \alpha * P + \beta * DQ \tag{5.1}$$

Where α, β are the weights attributed respectively to the performance and the data quality factor. These weights vary according to the importance that we give to each of these factors.

We recall that data quality is defined as in chapter 4 in $[0,1]$ as follows:

$$DataQuality = T_{DS} * T_{DB} \tag{5.2}$$

Where T_{DS} is data timeliness evaluated using equation 4.1, and T_{DSB} is database timeliness evaluated using equation 4.9.

The *Performance* describes the computing capacity of a data service defined by three metrics: availability, time efficiency, and task success ratio. The general idea behind the performance factor is that the data service is expected to be available when it is requested, it should stick to the response time announced by its provider, and it must systematically deliver data to consumers successfully.

The importance level assigned to each QoS parameter varies since QoS properties vary in units (i.e., response time is represented in milliseconds while cost is represented in cents). For example, a client that sets all weights to zero except for task success ratio indicates that our model should maximize task success ratio since it represents 100% significance to the client.

$$P = \sum W_j * Q_j \tag{5.3}$$

Where $Q_j = \{\text{availability, task success ratio, time efficiency}\}$ detailed hereafter. W_j: weight of the metric j, which varies according to the importance of a metric in a given context to the data service's consumer.

- **Availability (Av):** First, we want to be sure that the service remains up and functions without disruption. If the service is down, it won't be able to answer user's query. A data service is said unavailable when a request is denied [Man15]; [Ger91]. For instance, doctors in the e-health scenario must be sure that data services are available whenever they need to access Alice's data.

  Availability can be defined as in [Man15] as the degree to which a data service is operational and accessible when requested. Therefore, we define availability in $[0,1]$ as follows:

  $$Av = \frac{A_k}{N_k} \tag{5.4}$$

  Where A_k is the number of accepted requests by the data service k and N_k is the total number of requests submitted to the data service k during a period of time.

- **Task Success Ratio (TSR)**: Sometimes, data services may be available and accessible but are unable to deliver data to data consumers due to network failures, timeouts set by the consumer, etc. The task success ratio measures successful data delivery in response to accepted requests. Back to our scenario, a low task success ratio associated with a given data service indicates that there is a considerable risk of not getting the needed information.

$$TSR = \frac{S_k}{A_k} \tag{5.5}$$

  Where S_k is the number of successful requests where data arrived to destination by the data service k. TSR is in $[0, 1]$.

- **Time Efficiency (TE):** *TE* quantifies how well a data service meets the expected response time (ERt) promised by the service provider.

  Indeed, having a data service with a low response time violation on average w.r.t ERt indicates that one can rely on this service for urgent decision-making more than a service with a higher response time violation. Time efficiency metric is defined in $[0, 1]$ as follows:

$$TE = 1 - \frac{Rt}{ERt} \qquad \text{if} \qquad Rt < ERt \tag{5.6}$$

$$TE = 0 \qquad \text{if} \qquad Rt > ERt \tag{5.7}$$

  where Rt is the response time of the service on average.

In this section, we have defined three metrics including availability, time efficiency, and task success ratio, and how we use them to evaluate service performance. We have also defined a formal model for trust evaluation using service performance and data quality (defined in chapter 4) for black box stream data services. In the next section, we present DETECT that implements and helps validate the proposed trust evaluation model.

5.3 DETECT: black box Data sErvice Trust Evaluation arChitecTure

Figure 5.1 shows the general structure of *DETECT* that implements our trust evaluation model for black-box data services. The main objective of *DETECT* is enabling data requesters to select the most trustworthy data service according to our model providing a trust-sorted list of data services tagged with their up-to-date trust index.

DETECT is composed of three main modules: (i) the performance measuring module (PMM); (ii) the data quality measuring module (DQMM); and (iii) the trust measuring module (TMM). *PMM* monitors data service's performance, while *DQMM* observes and collects relevant metrics for evaluating the data quality provided by a data service, and *TMM* collects both data quality and performance measurements to compute data services trust scores. These modules are presented in the next section.

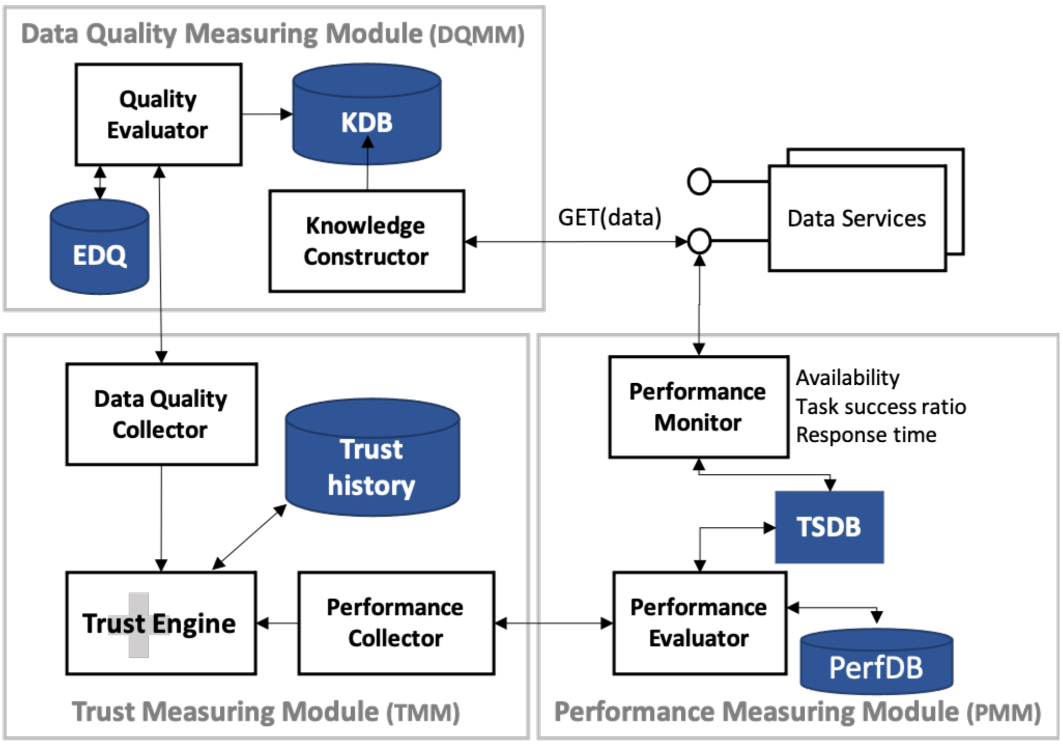


Figure 5.1: Data Service Trust Evaluation Architecture.

5.3.1 Performance Measuring Module

The *PMM* monitors the performance of candidate data services, and performs its operations in two steps: *observation* and *evaluation* described hereafter.

Observation step. It consists of observing and collecting the necessary evidence for the evaluation of service performance.

First, the *Performance Monitor* collects evidence related to data service response time, task success ratio and availability through continuously scraping performance metrics using the service's API. Several interesting tools for scraping and observing such performance metrics have been proposed in the literature [Li+18], and are available on-line (e.g., JMETER[3], Prometheus[4] etc.).

[3]`https://jmeter.apache.org`
[4]`https://prometheus.io`

Second, it records for a given instant (i.e., timestamp) their values. The *Performance Monitor* stores the time-stamped performance measurements in a time-series database (TSDB) which indicates the recorded response time of services in milliseconds, whether the service was available, and whether it successfully finished the job for each scraping timestamp. TSDB enables to store large volumes of timestamped measurements data in a format that allows fast insertions and fast retrieval to support the evaluation of the performance level using this data.

Evaluation step. It consists in evaluating the performance level of candidate services by the *Performance Evaluator* using the evidence stored in TSDB.

First, the performance measurements made within a specific time interval (including response times for the different timestamps, the number of requests made, the number of accepted requests, and the number of successful requests within this time interval) are collected.

Second, it measures performance metrics as defined in section 5.2 by computing (1) the response time on average for the corresponding time interval, (2) the availability and (3) the task success ratio for each candidate service.

Third, the *Performance Evaluator* evaluates the performance level of the corresponding service by applying equation 5.3. The *PerfDB* database stores the list of services tagged with their evaluated performance level along with the evaluation timestamp. This list is updated periodically each time the performance of the service is evaluated.

An API is made available to enable the trust measuring module to access the latest recorded performance level of candidate services.

5.3.2 Data Quality Measuring Module

The *DQMM* observes and evaluates the quality of the data accessed by every black box data service candidate. This evaluation is based on both data timeliness and database timeliness. DQMM implements our TUTOR-based data quality evaluation module presented in chapter 4.

To recall, the *knowledge constructor* observes the database states change using sampling techniques, computes the data sample timeliness, checks whether the database state has changed, and stores these observations in the *KDB* knowledge database. The *quality evaluator* evaluates database update frequency and thus, data quality level, and stores them in *EDQ* which contains a data quality-tagged list of candidate data services.

An API is made available to enable the trust measuring module to access the latest recorded data quality level of the candidate services.

5.3.3 Data Service Trust Measuring Module

The TMM evaluates the trust level of black-box data services using as input the *PMM* and *DQMM* output results. This module functions in two steps working according to a pub-sub model: the *collection* phase which collects performance and data quality levels, and the *evaluation* step which evaluates the trust level consuming the output of the collection step. Figure 5.2 presents the trust evaluation process.

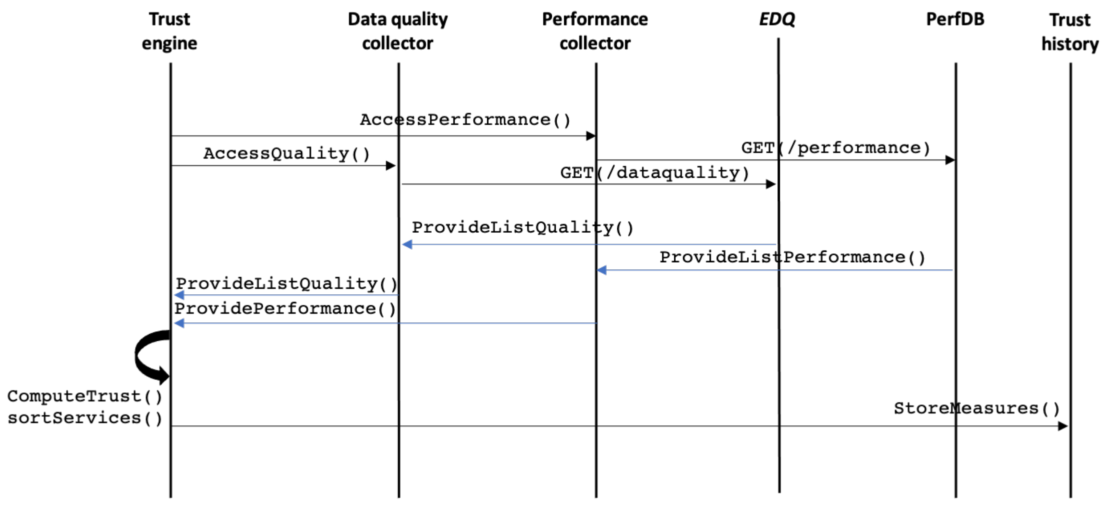


Figure 5.2: Trust Evaluation Process.

Collection step. Two collectors respectively the *performance collector* and the *data quality collector*, gather in parallel performance and data quality levels: the *performance collector* (respectively, the *data quality collector*) which gathers performance levels (respectively, data quality levels) of the candidate data services.

We assume that when receiving a data service request, the *Trust Engine* has as input the list of pre-selected data services that can semantically answer the user's query which is guided by the users' requirements. Indeed, this list is provided by another component (yet to be considered) including Rhone proposed in [Car+16] in the context of the SUMMIT project. Rhone is a services composition algorithm that semantically matches the available services with the users' requirements, and provides as output the list of candidate services. Therefore, once receiving a request (i.e., data service request), the trust engine requests the lists of service performance (*AccessPerformance()*) and data quality levels (*AccessQuality()*) of candidate services. Both these collectors access their corresponding databases (*EDQ* for *DQMM* and *PerfDB* for *PMM*) using REST APIS through the functions *GET(/dataquality)* and *GET(/performance)*. As a result, the trust engine receives back the requested lists through the *ProvideListQuality()* and *ProvidePerformance()* functions. The list accessed by the data quality collector contains candidate services IDs and their latest evaluated data quality levels. The list accessed by the performance collector contains candidate services IDs, and their latest evaluated performance levels.

Evaluation step. Using both these lists, the *Trust Engine* extracts the data quality level and performance level for each service ID, and evaluates the trust level of each candidate data service using equation 5.1 (*computeTrust()*). It further enables the data requester to specify the importance degree of trust factors (i.e., performance and data quality). A history of the evaluated trust levels of each service for the multiple requests is stored in the *trust history* which contains for every measuring timestamp the service ID, its data quality level, its performance level, and the corresponding data service trust level.

The proposed architecture *DETECT* has been implemented and validates our proposed trust evaluation model for black box data services as detailed in the next section.

5.4 DETECT: Implementation and Evaluation

This section demonstrates the applicability of our proposed trust evaluation model through a practical scenario. The objective being to demonstrate the effectiveness of our solution in ranking data services according to users' requirements, and its ability to satisfy these requirements. To do so, a proof of concept of the proposed model using the architecture *DETECT* has been developed, and experiments were conducted using the Alice e-health scenario. We remind that in such a scenario, Alice's doctor wants to select a trustworthy service among the available candidate services to access her medical records including her temperature readings (see section 1.1). Note that we use the same machine used in the experiments in chapter 4.

Two experiments are conducted: the first one has as objective to show the ability of our solution to rank data services using their trust levels according to users' requirements. Whereas the second has the objective to show the ability of our solution to satisfy users' requirements in terms of time efficiency and data timeliness, and demonstrates the added-value considering data quality factor in the data service trust evaluation model.

To this end, before presenting the experimental results, we will first present the common experimental settings of the two experiments.

5.4.1 Experimental Setting

Figure 5.3 presents the experimental setting of DETECT including PMM setting and DQEM setting presented in the next sections.

PMM setting

Three medical *HAPI FHIR* servers are deployed with multiple *FHIR* standards on docker containers: one simulating the hospital's server (HAPI FHIR$_2$), one simulating Alice's smartphone' server (HAPI FHIR$_1$) and the last one simulating the SOS server of the hospital (HAPI FHIR$_3$) (see figure 5.4). Each server has its independent database (respectively STU3, R4,

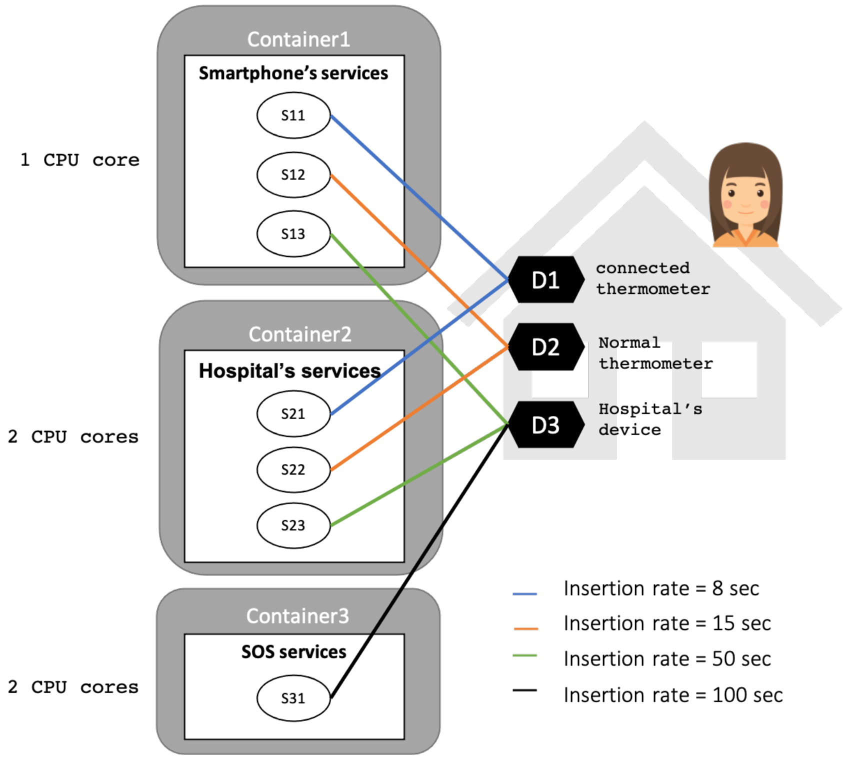


Figure 5.3: Experimental setting: Data Service Trust Evaluation.

and DSTU2) on the corresponding server, and is reachable through its URL. To give access to these servers, 7 data services are deployed on these servers each with its access point (API).

By default, a Docker container has no resource constraints and can use a given resource as much as the host's kernel scheduler allows. Therefore, for our experiment, we allocated a limited number of CPUs (equal to 7) of the host machine that Docker can use and allocate freely among the running containers. Indeed, it is important to not allow Docker to consume too much of the host machine's memory to avoid the risk of running out of memory. Docker also provides ways to control how many CPUs are allocated to each container.

Therefore, to simulate the variation in the performance of the candidate data services, a different number of CPU cores and services are allocated to the three *HAPI FHIR* servers as advised in[5]. The logic is: (1) The bigger the number of the allocated CPU resources for a server, the better the performance level since we have more resources for the server. Note that initially, every *HAPI FHIR* server where services run requires 2 CPU cores to function correctly. (2) The smaller the number of the deployed services on the same container, the better the performance since the available resources on the server are shared between a smaller number of services.

Accordingly, 1 CPU core is allocated to Alice's smartphone' server, 2 CPU cores are allocated to the hospital's server, and 2 CPU cores are allocated to the SOS server (see

[5]`https://docs.docker.com/config/containers/resource_constraints/`

figure5.3). Moreover, three data services give access to the three devices used by Alice are deployed on HAPI FHIR$_1$ and HAPI FHIR$_2$. Only one service gives access to HAPI FHIR$_3$. Indeed, by allocating a different number of CPUs and varying the number of services deployed per server, services have different performance levels.

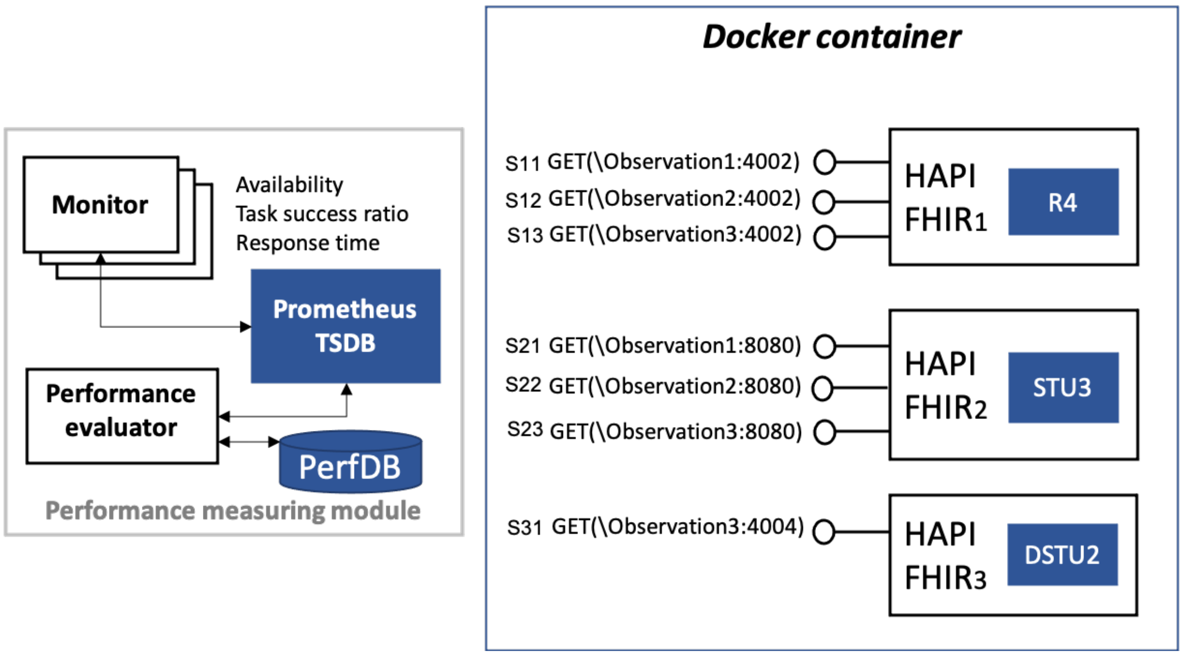

Monitor
Availability
Task success ratio
Response time
Prometheus TSDB
Performance evaluator
PerfDB
Performance measuring module
Docker container
S11 GET(\Observation1:4002)
S12 GET(\Observation2:4002)
S13 GET(\Observation3:4002)
HAPI FHIR1
R4
S21 GET(\Observation1:8080)
S22 GET(\Observation2:8080)
S23 GET(\Observation3:8080)
HAPI FHIR2
STU3
S31 GET(\Observation3:4004)
HAPI FHIR3
DSTU2

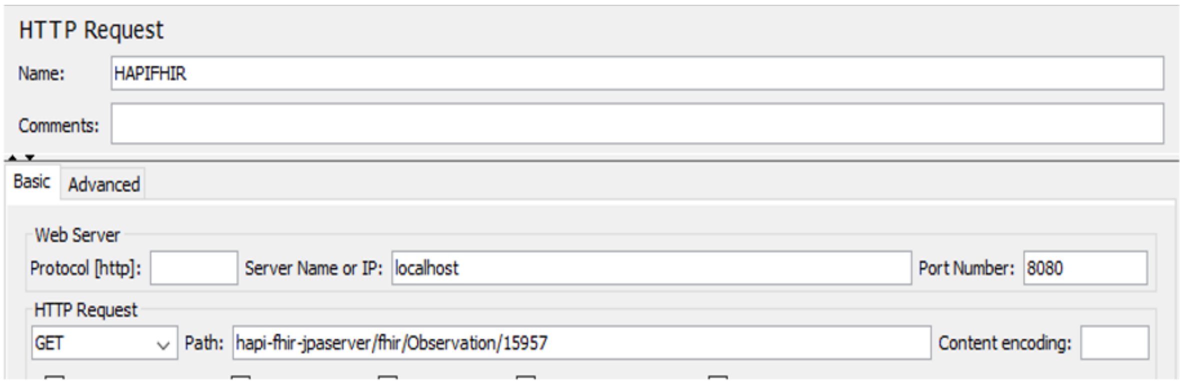
Figure 5.4: PMM Experimental Setting: Docker containers.

Following this logic, service S_{31} has the highest performance since it is deployed solely on the container with 2 CPU cores followed by the three services (S_{21}, S_{22}, S_{23}), which are deployed on the second container and also have 2 CPU cores, and finally, the three services (S_{11}, S_{12}, S_{13}) on the first container have the worst performance rate since they share only 1 CPU core compared to the second container.

The performance metrics are observed and measured using *JMETER*, and consumed by Prometheus for k candidate data services (k=7 in our setting) as described hereafter.

JMETER allows us to simulate various activity levels at the input of data services (i.e., a different number of incoming requests) to test their performance. This activity is simulated using a thread group thanks to a listener as shown in figure 5.5. Indeed, a thread group is a set of processes sending the same request in parallel to a given service API specified in the listener. Each process simulates a user request to a data service. *JMETER* enables to choose the number of processes per thread group. A thread group is referred to as a "post" and its processes as "jobs". A post can be configured to run with a (in)finite number of loops that can be specified. For instance, a first post that is configured with 100 jobs and 3 loops means that when executed, the post sends 100 jobs simultaneously to the target service three times in a row. A second post that is configured with 1 job and an infinite number of loops means that when executed, the post sends 1 job simultaneously to the target service infinitely.

For each thread group, JMETER observes and records several measures (i.e., KPIs) including the response time measurement of each service, and the HTTP response code for each post including its jobs.



HTTP Request

Name:	HAPIFHIR
Comments:	

Basic | Advanced

Web Server

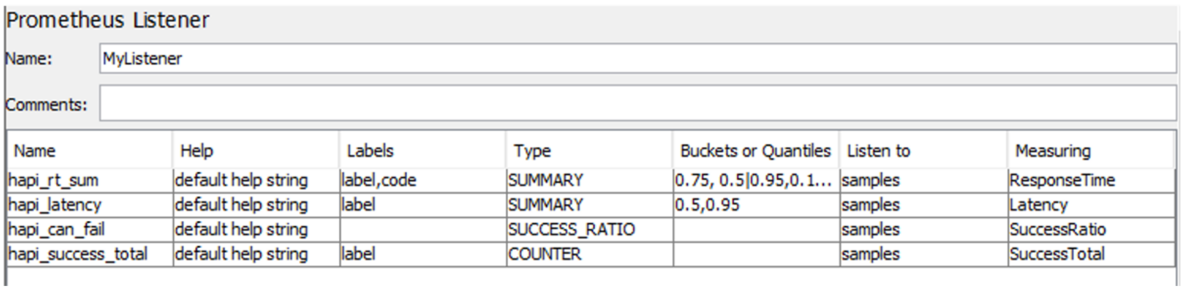
Protocol [http]:		Server Name or IP:	localhost	Port Number:	8080

HTTP Request

GET	Path:	hapi-fhir-jpaserver/fhir/Observation/15957		Content encoding:	

Figure 5.5: JMETER: Example of GET request.

JMETER enables Prometheus to scrape its measures through an exporter (see figure 5.6) by indicating what measures to scrape and their type. For instance, the first line in the table on the figure indicates to JMETER that Prometheus needs to export the response time measures for the different sent jobs as a summary. Therefore, JMETER makes use of the observed KPIS and transforms them into the desired metrics indicated by the exporter. For instance, the response codes are used to count the number of times a service accepted a job (code=202) or fails to respond to deliver data (code=404) over a time period.



Prometheus Listener

Name:	MyListener
Comments:	

Name	Help	Labels	Type	Buckets or Quantiles	Listen to	Measuring
hapi_rt_sum	default help string	label,code	SUMMARY	0.75, 0.5|0.95,0.1...	samples	ResponseTime
hapi_latency	default help string	label	SUMMARY	0.5,0.95	samples	Latency
hapi_can_fail	default help string		SUCCESS_RATIO		samples	SuccessRatio
hapi_success_total	default help string	label	COUNTER		samples	SuccessTotal

Figure 5.6: JMETER: Prometheus Listener.

Prometheus is configured to continuously scrape metrics every $15s$ ($\Delta t1$), and to store them in its *TSDB* for the different k candidate data services (see figure 5.7).

By default, observations related to the availability of services and their task success ratios (x_{k1}, x_{k2}) are of type Boolean, and thus, do not need normalization (0 if unavailable/unsuccessful, or 1 if available/successful job). However, the response time measure must be normalized in $[0, 1]$ to eliminate the time dimension of the observed data by JMETER. Therefore, the *performance evaluator* contains a process namely *EvalRT* that computes the average response time for the different k candidate data services in seconds, and normalizes it following a series of steps:

- *EvalRT* performs a GET request using *promQL* -Prometheus querying Language - every period $\Delta t2$ equal to $1m$ (i.e., minute) to access two time-series observations provided by Prometheus (x_{k3} that contains the summary of response times per period $\Delta t2$ in seconds, and x_{k4} that counts the number of jobs per $\Delta t2$). In fact, promQL requires the duration for accessing the summary of response times per minute, and thus, we selected the minimum ($1m$) to detect any sudden changes in the response time values. Concerning the choice of $\Delta t1$, it does not impact the results as long as it is smaller than $\Delta t2$ ($\Delta t2$=4*$\Delta t1$) because the values are going to be averaged per $\Delta t2$.

- For every service k, *EvalRT* normalizes the computed average response time (NRt$_1$ .. NRt$_k$) using the expected response time as in equation 5.7, and stores it in a separate database *RespDB* along with the measuring timestamps. Note that the expected response time *ERt* of these services is set to $300ms$ (average response time it takes to get a response using an API[6]).

At an instant t, the *Performance Monitor* accesses Prometheus *TSDB* along with *RespDB* every $200s$ ($\Delta t3$) to compute the performance metrics of the related data services for the last $200s$ using equations presented in section 5.2. Finally, the *Performance Evaluator* computes candidate service performance as in equation 5.3. Currently, we assume that availability, task success ratio, and response time are equally important, and thus, they have equal weights when computing performance level.

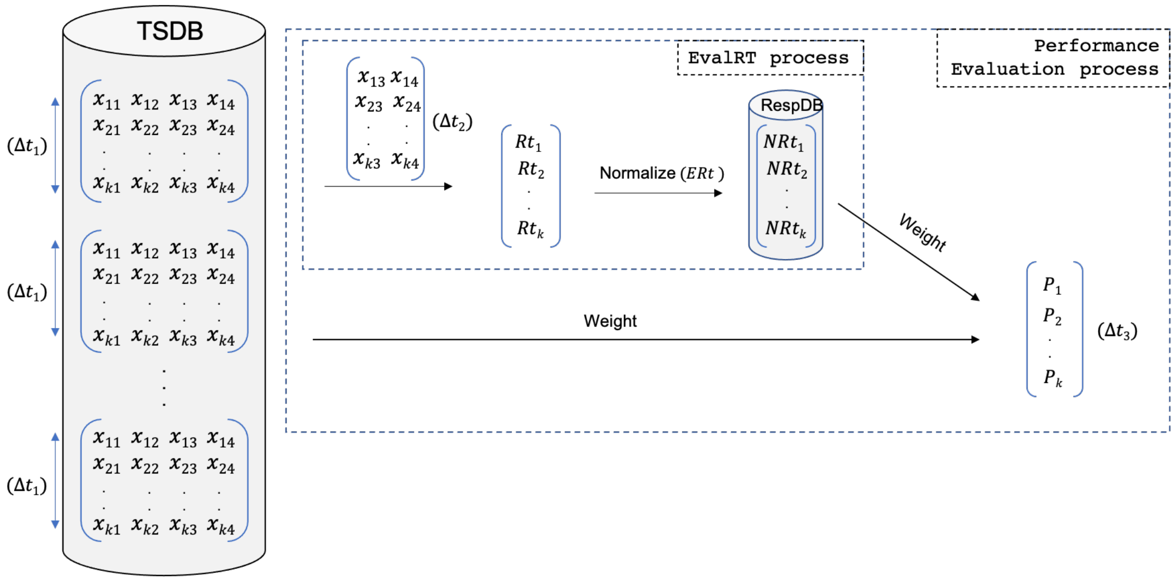


Figure 5.7: Performance Monitoring Process.

The choice of $\Delta t3$ ($\Delta t3 > 3^* \Delta t2$) is motivated by the need to compute response time on average using at least 3 observations stored in *RespDB*. In fact, we believe that the evaluation of the time efficiency should consider the most current response time value of the service and past values. For instance, a service may have on average a good response time but

[6]https://blog.avenuecode.com/how-to-determine-a-performance-baseline-for-a-web-api

unfortunately, the most current evaluated value shows that it took way longer than usual to respond due to a rare error. We don't want to be judging the performance of this service using only this one experience. Therefore, we considered the last measured response time at the instant t (i.e. the most current), and two response time measures at instants *t-1* and *t-2*.

DQEM Experimental Setting

In the absence of real-time real-world data services, the process of data production and data insertion are simulated for Alice's three devices as we did in chapter 4. We recall that the data validity interval T is pre-defined according to the application domain. As in chapter 4, T is set to $60s$, which means that data about temperature remain valid for only $60s$, and beyond this time, data are no longer considered fresh. Indeed, in our scenario, Alice's health state is critical and thus, the temperature recordings should be timely in terms of seconds.

Also, we have varied data production rates and data insertion rates for all devices in a way that they have different data quality levels w.r.t our experimental setting:

- (1) all data services have the same data production rate ($3s$),
- (2) data services S_{11} and S_{21} have the highest insertion rate equal to $8s$ (up to 6 insertions insertions per data validity interval),
- (3) followed by data services S_{12} and S_{22} with an insertion rate of $15s$ (up to 3 insertions per data validity interval),
- (4) data services S_{13} and S_{23} with an insertion rate of $50s$ (1 insertion per validity interval),
- and (5) last data service S_{31} has the lowest insertion rate equal to $100s$ (1/2 insertion per two validity intervals T).

The choice of these values is motivated by the need to deploy services which provide timely data with different levels of timeliness (i.e., insertion rate is faster than the validity duration) (S_{11}, S_{21}, S_{12}, S_{22}, S_{13}, S_{23}) and other services that provide data with low timeliness (i.e., insertion rate is slower than the validity duration) (S_{31}).

According to this configuration, it is expected that services S_{11} and S_{21} have higher data quality as they have the highest database update frequency, followed by S_{12} and S_{22}, followed by S_{13} and S_{23} and finally S_{31}.

Note that data insertion is performed using *API*s of the deployed data services, and data about Alice's body temperature are made accessible through the resource observation.

We recall that data timeliness and database timeliness, and thus, data freshness are evaluated as in equations defined in section 4.3.3. Therefore, data quality is evaluated as follows:

$$DataQuality_{ROP,Si} = T_{D,ROP,Si} \times T_{DB,ROP,Si} \tag{5.8}$$

Where $ROP{=}\Delta t_3$ for convenience reasons so that PMM and DQEM measure respectively performance and data quality for the same period.

To this end, table 5.1 shows the deployed services ranking according to the expected performance and data quality levels.

Table 5.1: Services Ranked According to Trust Factors

Performance	***Data Quality***
S_{31}	S_{11}, S_{21}
S_{21}, S_{22}, S_{23}	S_{12}, S_{22}
S_{11}, S_{12}, S_{13}	S_{13}, S_{23}
	S_{31}

This general experimental setting is used to launch two experiments presented in the next two sections.

5.4.2 Experiment 1: Ranking according to trust factors preferences

This section presents an experiment that aims to evaluate the ability of our solution to rank data services according to users' trust factors preferences. Users' trust preferences are expressed through weights assigned to both trust factors.

Trust Case Studies

Two case studies based on the eHealth scenario are presented to validate the trust model and *DETECT*'s applicability. Each case study is designed to target one or both trust factors. We assume that we have two kinds of services: on-demand services and pushed services. Note that we present two case studies but the process of trust evaluation is the same. The chosen case studies illustrate two different applications requirements for which performance and data quality will be weighted accordingly.

- **Case Study 1: Alerting**

  This case study addresses continuous monitoring of Alice's temperature. In the case of a significant variation in Alice's temperature, the devices' push service must immediately alert Alice's doctor. For instance, sending a message with her current temperature and location. Both performance and data freshness are essential since up-to-date data must be transmitted immediately when a sudden variation in temperature occurs. Since Alice's safety is at risk, data must be sent timely, and thus, the performance criterion for selecting services seems more critical than data freshness. Therefore,a bigger weight is assigned to the performance factor when evaluating the trust levels of the available and pre-selected services using the equation 5.1.

- **Case Study 2: Instant checkup by Alice's doctor**

  In this case study, we assume that Alice's doctor wants to perform from time to time an instant checkup on her health latest indicators using on-demand services. For example, she wants to have the latest recorded temperature, and thus, data must be as fresh as possible. As described in our scenario, temperature readings are done manually and digitally using various devices (e.g., hospital's devices, her connected thermometer, and her manual thermometer), and measures are taken at multiple frequencies and rates. This use case emphasises the data quality factor (i.e., data freshness produced by services managing the data produced by the various devices). Thus, one expects the system gives higher weight to data quality factor, when evaluating the services trust using the equation 5.1.

Configurations

Our experiment consists of evaluating the effectiveness of our solution by testing the effect of α and β on the data services ranking:

A request is performed to the *Trust Engine.* As a result, the *trust engine* outputs a ranked list of the available services according to their trust level. We recall that the objective is to test the effectiveness of our solution to rank data services for the presented case scenarios. Therefore, for fixed performance and data quality levels, α and β are varied to show the effect of the evaluated data quality level and performance level on the trust level evaluation of data services. To do so, the weight α is decreased from 1 to 0 while β is increased from 0 to 1 (see table 5.2).

Results and Discussion

Tests are performed using the configuration described above in order to verify our trust model. A request is sent to the *Trust Engine* and results are observed. The resulting ranked lists of services are presented in table 5.2.

According to table 5.2, we notice that:

- The service S_{31} has the highest trust level when we only give importance to service performance (α=1 and β=0); the less we give importance to performance, the more S_{31} is ranked lower in the list (5th place when α=0).

Table 5.2: Request output for different requirements

α=*1*,β=*0*	α=*0,7*,β=*0,3*	α=*0,5*,β=*0,5*	α=*0,3*,β=*0,7*	α=*0*,β=*1*
S_{31}	S_{22}	S_{21}	S_{21}	S_{21}
S_{23}	S_{21}	S_{22}	S_{22}	S_{11}
S_{22}	S_{31}	S_{31}	S_{31}	S_{22}
S_{21}	S_{23}	S_{23}	S_{23}	S_{12}
S_{13}	S_{11}	S_{11}	S_{11}	S_{31}
S_{12}	S_{22}	S_{12}	S_{12}	S_{13}
S_{11}	S_{13}	S_{13}	S_{13}	S_{23}

- The hospital's services (S_{21}, S_{22}, S_{23}) are ranked below the smartphone's services when α=1 and β=0. The more we give importance to the data quality factor, the better they are ranked and overpass the trust level of S_{31} excluding S_{23}.

- Services deployed on the first container (S_{11}, S_{12}, S_{13}) are ranked first in the list when α=1 and β=0. The more we give importance to the data quality factor, the better they are ranked and overpass the trust level of S_{31} excluding S_{13}. No matter the weight we attribute to trust factors, services deployed on the smartphone *HAPI FHIR* server are consistently ranked below the services deployed on the hospital's *HAPI FHIR* server.

Hereafter, we discuss the results for each case study.

- **Case study 1**: as aforementioned, in this case study, we are only interested in selecting data services with the highest performance level: α=1 and β=0.

  According to the above observations and as shown in table 5.1, experiments demonstrate the feasibility of our trust model and architecture since results provide the suitable trust-based ranking of data services as expected. Note that in our configurations, we control the performance of *HAPI FHIR* servers but not the performance of services deployed on the same server. These services run independently, and the allocation of resources among them depends on the docker containers' load balancing and scheduling method. However, for services deployed on the same container, we can perform some actions to diminish the performance of some of them, like sending more user requests.

- **Case study 2**: in this case study, we are interested in selecting data services with the highest data quality: α=0 and β=1.

  According to the above observations: (1) Services S_{21} and S_{11} have the best data quality. This result is not surprising because the configured data insertion rate for these two services is the highest. However, S_{21} is better than S_{11}. (2) As expected, S_{12} and S_{22} are ranked next since they have a lower insertion rate. Still, S_{22} is better than S_{21}. Services S_{13} and S_{23} followed them. (3) S_{31} was ranked before S_{13} and S_{23}.

These results can be explained by the correlation that exists between data quality and services' performance. The value of the data timeliness highly depends on the performance level of the corresponding service.

In fact, the characteristics of the infrastructure negatively influence data timeliness. An infrastructure configuration providing limited resources punishes the service's response time, and in consequence, the data timeliness suffers too.

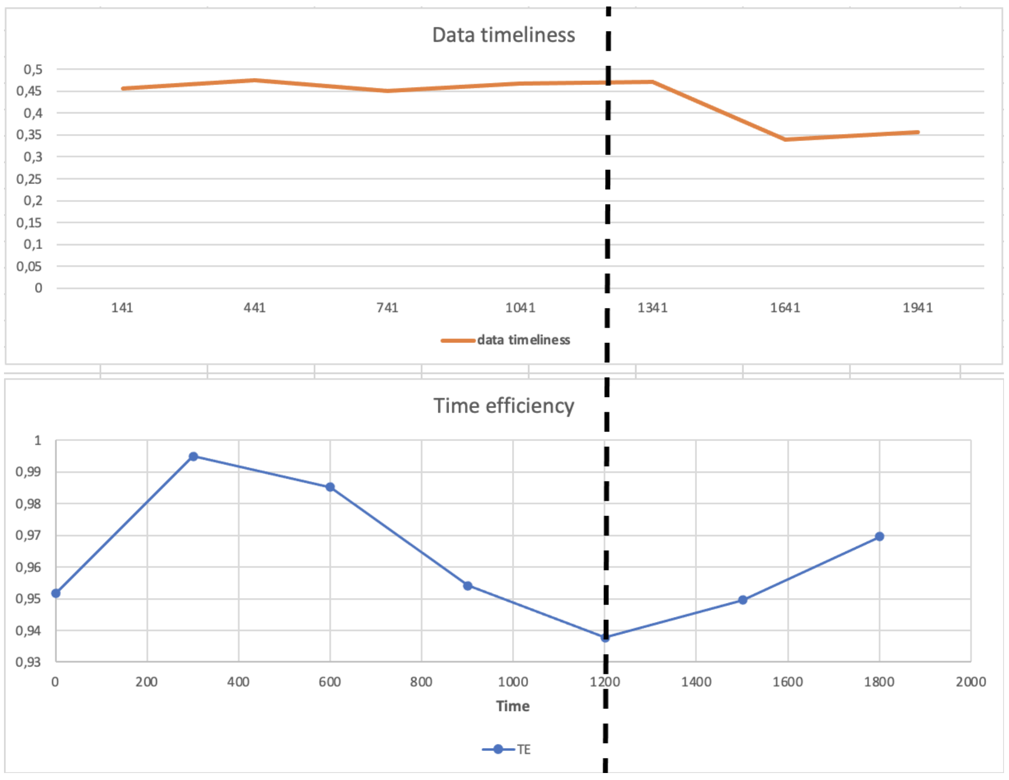


Figure 5.8: Correlation between time efficiency and data freshness.

Note that this correlation affects the data timeliness value because data in our scenario changes frequently and has a very small validity interval (in a matter of seconds). Thereby, data loses rapidly their freshness. Indeed, the longer the service takes to respond to the query, the worst its time efficiency, the worst the data timeliness, and thus, data freshness. For instance, services on the hospital's server are always better ranked than those deployed on the smartphones server even though their data production and insertion rates configuration is similar. That is because the performance configuration of the hospital's server gives access to more resources.

See figure 5.8 which represents the effect of time efficiency variation on the evaluated data timeliness. The horizontal axis represents the time when measurements were performed for each factor and the vertical axis the data timeliness level and time efficiency level. The vertical black line links the same time point for both figures. We can see that when time efficiency decreases, data timeliness level decreases accordingly.

This dependence should be represented through the data service trust evaluation model defined in equation 5.1. Nevertheless, we did not consider this dependence yet between the trust factors in our proposed trust model.

5.4.3 Experiment 2: User Satisfaction Evaluation

This section presents an experiment that aims to evaluate the effectiveness of our trust evaluation solution in satisfying users' requirements related to data freshness and response time. Note that we use the experimental setting presented in section 5.3.

Configurations

In this experiment, we assume that the user (Alice's doctor) requires Alice's temperature, with a maximum service response time (ERt) of 0,3s and the most current data captured (T) within the last 60s. In other words, the user expects the response time not exceed 0,3s and the data timeliness not exceed 60s. There are 7 candidate data services that can provide data under these QoS and QoD requirements.

A total of 5 independent posts are performed using JMETER with different thread group configurations N (i.e., the number of parallel incoming jobs per post) by increasing the number of jobs which ranges from 1 to 400 jobs with a step size of 100 jobs. The choice of the number of jobs per post is motivated by the need to increase the number of parallel incoming jobs as much as possible. Due to our setting's performance limitations, we stopped at 400 jobs[7]. Therefore, submissions are differentiated by a different and increasing numbers of jobs that would affect the time efficiency of the candidate services and by correlation, their data timeliness (as discussed in the previous section).

Each post is sent simultaneously for the 7 candidate data services (1 job per service to 400 jobs per service) and runs for a period of 50m. For instance, the first post consists of sending 1 job simultaneously for 50m, and the last job consists of sending 400 jobs simultaneously for 50m. During each post, data services are selected using three models: random model (RAND), performance based trust model (P-TEM), and performance and data quality based trust model (P, DQ-TEM) described hereafter.

RAND : This model operates without trust evaluation. For each request, a data service is selected randomly among the candidate ones to respond to the request. The objective is to have a basic selection solution to compare to our solution.

P-TEM : This model consists of evaluating the trust level of data services using only their performance level ($\alpha = 1$). When receiving a request, the *Trust Engine* provides a

[7]Doker, JMETER, and Prometheus are very resource-consuming applications. In our setting, they run on the same machine. The more we increase the number of jobs per post, the more JMETER (which sends requests) and Docker (where services that receive the requests are deployed) consume resources.

performance-tagged ranked list of data services. By focusing solely on the service performance factor, the objective is to demonstrate the ability of our solution to satisfy users' requirements w.r.t response time.

P, DQ-TEM : This model consists of evaluating the trust level of data services using both their performance and data quality levels with equal importance ($\alpha = \beta = 0,5$) as defined in section 5.2. The choice of these weights is motivated by our need to see the added value of the data quality factor to the performance one in the trust evaluation. In other words, the objective is to see the difference between considering only performance (P-TEM) and considering equally performance and data quality (P, DQ-TEM). For other ends, these weights can be varied accordingly.

A test script is developed that recurrently sends one request with users' requirements to the *Trust Engine* (figure 5.9) and the RAND solution. Therefore, two instances are deployed: one to send requests to the *Trust Engine* (i.e., P,DQ-TEM and P-TEM) and one to the RAND solution.

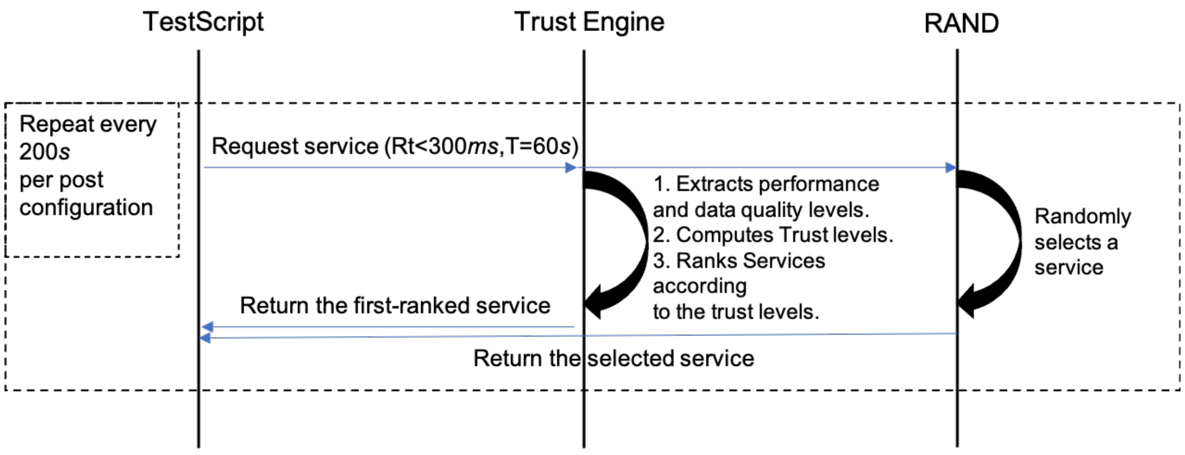

TestScript
Trust Engine
RAND
Repeat every 200s per post configuration
Request service (Rt<300ms,T=60s)
1. Extracts performance and data quality levels.
2. Computes Trust levels.
3. Ranks Services according to the trust levels.
Randomly selects a service
Return the first-ranked service
Return the selected service

Figure 5.9: Test script process: sending requests.

Since posts are launched independently, this test script is re-launched for every post and runs for $50m$ ($3000s$). For every post, a request is sent every $200s$. As a result, a request is sent 15 times (x) during a post. The choice of $200s$ is motivated by the fact that performance levels for candidate services and data quality levels change every $200s$ (see section 5.3).

We assume that the users' requirements remain the same for the different requests. Indeed, our goal is to observe the variation of user satisfaction level given the same requirements w.r.t the increase of the number of incoming jobs.

As a result, the *Trust Engine* (respectively, the RAND solution) outputs two ranked lists of the 7 candidate services according to their trust levels (one for α=1 and one for α=β=0,5) (respectively, one list for random ranking).

The effectiveness of the presented three solutions to reach user satisfaction w.r.t data timeliness level and time efficiency level is evaluated using metrics presented in the next section.

User Satisfaction Evaluation Metrics

Two metrics are defined for the evaluation of user (Alice's doctor) satisfaction w.r.t the results of a trust evaluation solution. The objective is to demonstrate that our solution succeeds in providing data services that adhere to users requirement related to response time and data timeliness. Assuming services are already available and succeed to deliver data, user satisfaction is evaluated as follows.

User satisfaction in the perceived time efficiency [Man15] of the selected service is measured using the metric US_{TE} which is evaluated in $[0, 1]$ as follows:

$$US_{TE} = 1 - \frac{Rt}{ERt} \quad \text{if} \quad Rt < ERt \tag{5.9}$$

$$US_{TE} = 0 \quad \text{if} \quad Rt > ERt \tag{5.10}$$

where Rt is the response time of the selected service post-usage and ERt is the expected response time.

User satisfaction in the perceived data timeliness of the selected service is measured using the metric US_{DT} which is evaluated as in equation 4.1 in $[0, 1]$ as follows:

$$US_{DT} = AVG(T_D) \tag{5.11}$$

Where T_D is the timeliness of data delivered to the data requester defined in $[0, 1]$ as follows:

$$T_D = 1 - \frac{t_R - t_P}{T} \quad \text{if} \quad t_R < t_{max} \tag{5.12}$$

$$T_D = 0 \quad \text{if} \quad t_R > t_{max} \tag{5.13}$$

We recall that t_R represents the request time (i.e., request sent to the service), t_P is the data production time, t_{max} is the maximum time for data to be considered as fresh and t_{min} $= t_P$.

The metrics US_{DT} and US_{TE} are measured automatically for every request to the selected service (as illustrated in figure 5.10). Indeed, the selected service with the highest trust value (respectively, the randomly selected) will be requested to execute a job by the first instance (respectively, the second instance). Subsequently, each instance evaluates the actual response time of the selected service (Rt) - time difference between invoking the selected service and receiving the data-, and thus, computes the user satisfaction level w.r.t time efficiency (if available and if successfully delivers data). Moreover, each instance evaluates data timeliness using the received data values and their production timestamps which allows to calculate the user satisfaction level w.r.t data timeliness.

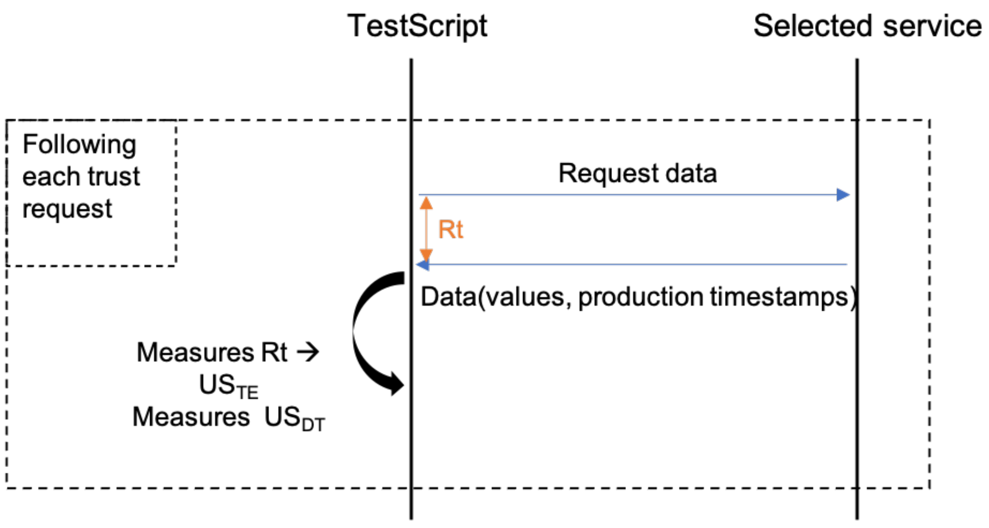


Figure 5.10: Test script process: evaluating user satisfaction.

User Satisfaction: Results & Discussion

Figures 5.12 and 5.11 depict the variation of user satisfaction level w.r.t time efficiency and data timeliness respectively for the first post configuration (1 job is sent per post). The objective is to observe the user satisfaction over time for a fixed number of incoming jobs (i.e., for the same post).

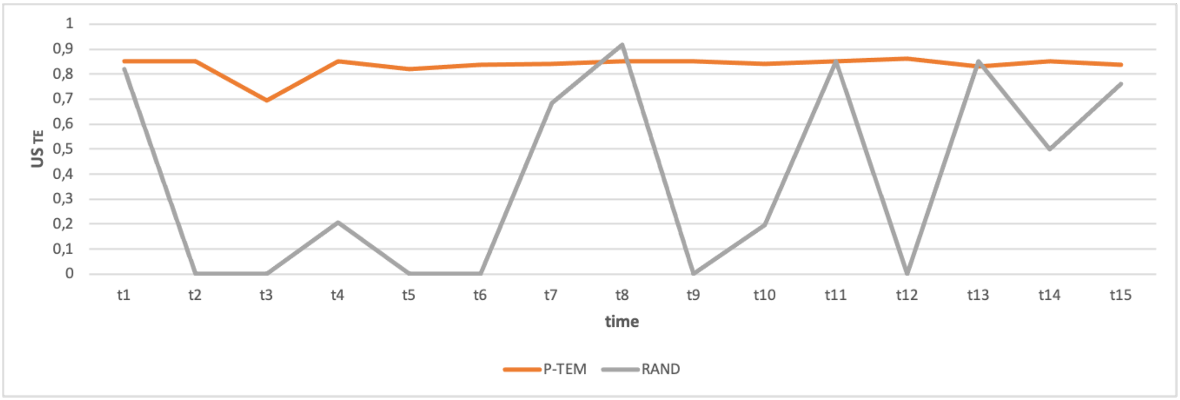


Figure 5.11: US_{TE} over time (1 job per post).

Figure 5.11 presents the variation of US_{TE} over time for 15 requests sent to P-TEM and RAND. The figure shows that the user satisfaction level w.r.t time efficiency US_{TE} is maintained throughout time using our solution and varies between 70% and 87%. However, the variation of US_{TE} is much more significant for the random solution which varies between 0% and 90%.

These results show that our solution outperforms RAND in satisfying the user w.r.t time efficiency when the number of jobs is maintained as it succeeds to maintain US_{TE} level.

Figure 5.12 presents the variation of US_{DT} over time for 15 requests sent to P,DQ-TEM and RAND. The figure shows that the user satisfaction level w.r.t data timeliness US_{DT} is somehow maintained throughout time using our solution and varies between 80% and 95%. However, the variation of US_{DT} is more significant for the random solution which is between 0% and 92%. These results show that our solution outperforms RAND in satisfying the user w.r.t data timeliness when the number of jobs is maintained as it succeeds to maintain US_{DT} level throughout time.

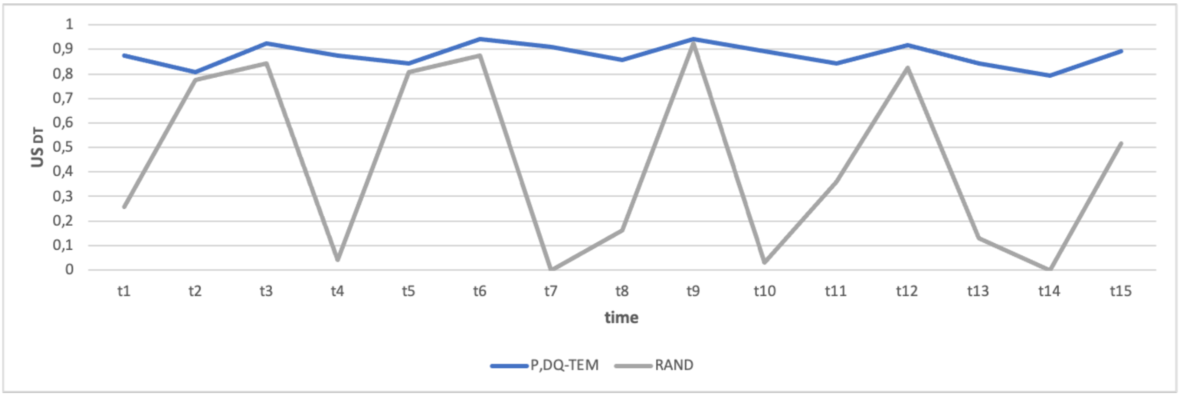


Figure 5.12: US_{DT} over time (1 job per post).

Figures 5.13 and 5.14 depict the variation of user satisfaction level on average for the various posts. The objective is to demonstrate the ability of our solution to evaluate the performance level and the added-value of the data quality factors to the trust evaluation model.

Figure 5.13 presents the user satisfaction w.r.t the perceived time efficiency level of data services selected using two trust evaluation models including RAND and P-TEM (PMM module). The horizontal axis presents the number of incoming requests to the candidate data services. Note that US_{TE} is measured automatically following every request, and averaged using x for every post configuration.

The figure shows that US_{TE} on average obtained using P-TEM that is deployed using DETECT ranges between 50% and 92% and outperforms RAND for all posts. Also, the figure shows that the level of the obtained US_{TE} on average decreases relatively with the incoming number of requests. This is because the response time of services are affected by the deployment conditions[8]: When increasing the load (i.e., number of incoming requests), Docker response time increases and thus, the probability to ensure a response time of 300ms for all services decreases. Moreover, it is worth noting that in our setting, the QoS of services are not scalable, and thus, can not always guarantee the response time. Indeed, no measures are taken when the service is overloaded with requests which need to be the case when using real services. Therefore, the obtained results can be better when ranking real data services. This lets us conclude that our solution is able to maintain user satisfaction w.r.t time efficiency.

[8]all DETECT components are deployed on the same machine and use softwares that are very resource-consuming. Therefore, the resources that Docker (where services are deployed) can consume from the host

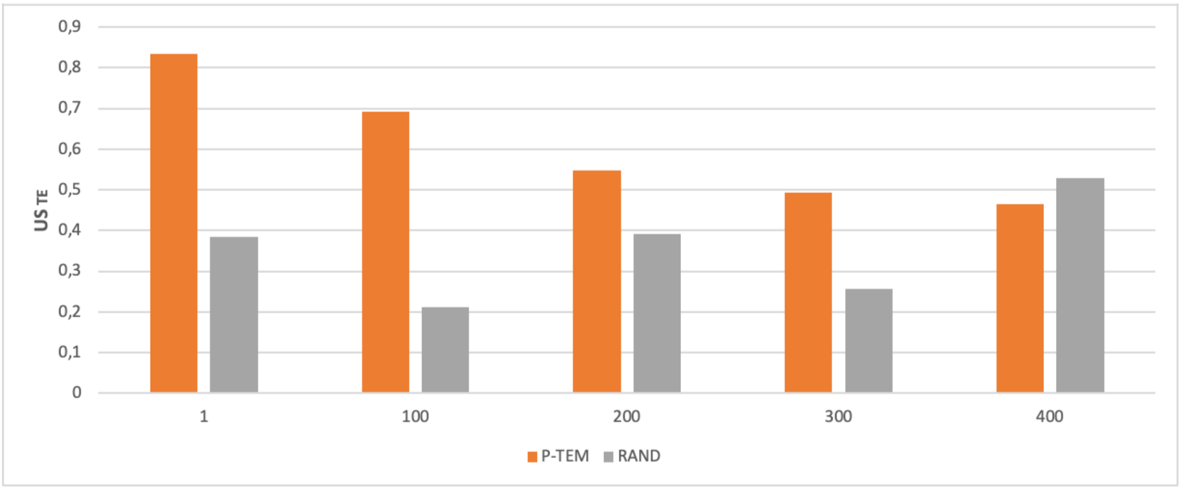


Figure 5.13: Comparison of Trust Evaluation Models (x = 15).

Figure 5.14 presents the user satisfaction w.r.t the perceived data timeliness level of data services selected using three trust evaluation models including RAND, P-TEM and P,DQ-TEM. Note that US_{DT} is measured automatically for every request, and averaged using x for every post Configuration.

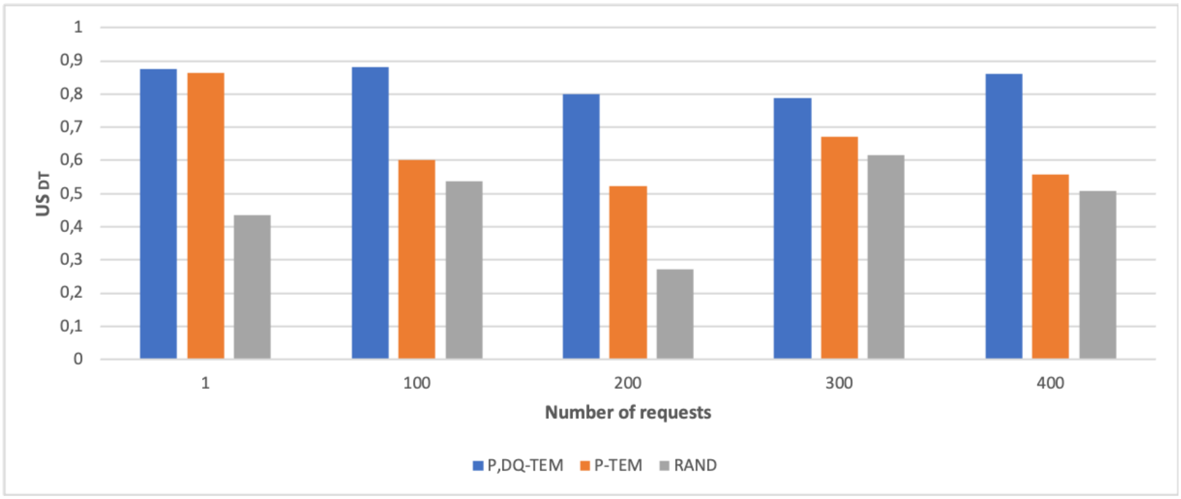


Figure 5.14: DT : Comparison of Trust Evaluation Models (x = 15).

The results show that the obtained US_{DT} level is higher for P,DQ-TEM which corresponds to our proposed trust evaluation solution than RAND and P-TEM which uses only the performance level to rank candidate services ($\alpha = 1$). This demonstrates that when data quality level focusing on data freshness is considered when evaluating the trust level of candidate services, the user (the doctor) satisfaction w.r.t data timeliness level is improved. This lets us conclude that the data quality factor adds value to the service trust evaluation and that our solution succeeds to maintain users' satisfaction w.r.t data timeliness.

machine are limited and fixed whatever the situation.

Now that our trust evaluation model is validated using DETECT, we move to testing the performance of our solution.

5.4.4 Computation Time Evaluation

In this section, we evaluate DETECT-based trust evaluation model in terms of computation time overhead. Therefore, we conducted an experiment to estimate the overhead induced by the *Trust Engine* functioning.

The processing time is defined as the time the *Trust Engine* takes to respond to a request: time elapsed from the instant the request is sent till the instant the user receives a response. This processing time is compared to the processing time needed to respond to a request by randomly selecting a service (i.e., without trust evaluation). For our experiments, we incrementally increase the number of the parallel incoming requests to both our selection solution P,DQ-TEM and the random-based selection solution RAND.

Using JMETER, the simulation is run for a small period of time during which an experiment is conducted which consists of sending 5 posts independently while incrementing the jobs per post (i.e., number of parallel incoming requests) from 1 to 400 with a step of 100 requests (table 5.3). Each post is run one time (1 loop), and then, the average response time for all jobs per post is measured by JMETER. For instance, for a post composed of 100 jobs, the average response time is measured as the average response time it took the Trust Engine (respectively, RAND) to respond to 100 jobs. The objective is to test and to compare the performance of both solutions when there is respectively a low and high number of incoming requests. We also stopped at 400 requests for performance constraints of the setting as in 5.4.3.

Table 5.3: Trust computation time overhead

N	RAND	P,DQ-TEM	Overhead
1	0,018	0,019	0,001
100	0,488	0,849	0,361
200	1,083	2,598	1,515
300	1,503	5,164	3,661
400	1,943	6,612	4,669

Table 5.3 presents computation time for two selection and ranking solutions including RAND and P,DQ-TEM. The results demonstrate that by increasing the number of incoming requests N, the time to send back a response given a data request for both solutions increases: RAND takes from 18ms when N=1 to 2s when N=400 while P,DQ-TEM takes from 19ms when N=1 to 6,6s when N=400. Moreover, the processing time of the *random-based selection* solution is always better than P,DQ-TEM generating an overhead that increases with the increase of the number of parallel incoming requests from 1ms to 4,6s. The overhead is

induced by P,DQ-TEM, running on a limited local execution environment, when accessing performance and data quality levels, computing the trust level of candidate services, and ranking them. This let us conclude that the increase of the number of incoming requests does not affect the performance of our DETECT-based solution.

This problem can be solved by improving the execution environment, in particular by adopting a distributed architecture logic for the deployment of DETECT.

5.5 Conclusion

5.5.1 Summary

This chapter answers the second research question of this thesis namely: "How to evaluate the trust level of black box data services?" for which we proposed a trust evaluation model for black-box stream data services using service performance and data quality as trust factors. This model enables the consumers to specify their requirements in terms of service performance and data quality level. Thus, trust is evaluated as the weighted sum of both these factors. Service performance is evaluated using three metrics including availability, time efficiency, and task success ratio. Data quality is evaluated using the model proposed in chapter 4. Moreover, we presented an architecture DETECT which helps validate our model. It is composed of three main components including a data quality measuring module (implementing the TUTOR-based data quality model, recurrently evaluating data freshness), a performance measuring module (recurrently evaluating performance metrics and thus, performance level), and a trust measuring module (evaluating trust level on demand).

Also, we presented the experimental setting and results of two experiments:

- The first was conducted to test the effectiveness of our solution in ranking data services according to users' preferences (different weights assigned to performance and data quality are attributed). Indeed, our solution succeeds for each importance level given to service performance and data quality factors in ranking the 7 candidate data services.

- The second experiment was conducted to (1) test the effectiveness of our solution to satisfy users' requirements w.r.t data quality and performance while increasing the number of incoming requests, and (2) demonstrate the added value of the data quality factor to the trust evaluation model. Indeed, when the number of jobs is maintained, our solution succeeds to maintain user satisfaction w.r.t data quality and performance throughout time with an average of 89% and 83% respectively. Moreover, the user satisfaction level is not affected by the increase in the number of incoming requests.

Finally, we evaluated the performance of our solution using DETECT in terms of processing time.

The above-described contributions and results have highlighted three types of limitations mainly concerning the trust evaluation model, users' requirements consideration, and scalability.

5.5.2 Limitations & Enhancement ideas

Trust evaluation model. Two limitations of our trust evaluation model has been identified and presented hereafter.

- Correlation between response time and data quality: as aforementioned, we noticed that there is a correlation that exists between these two trust factors. This correlation affected the currently obtained ranking results.

  Therefore, as future work, we plan to propose a mechanism to correct the bias induced by this correlation.

- Trust update: currently, trust is evaluated using only the last observations made about data quality and performance. However, it is commonly agreed in the literature of trust that the more your have cues about something, the better the trust judgment. Indeed, the judgment is poor when it is made only using the last observations, and it is consolidated when it considers past evaluations (i.e., gives more insights). As future work, we can propose an update function that evaluates current trust considering the most recent observations about performance and data quality and the last evaluated trust value. A possible model (to be studied and validated) for this evaluation is given in equation 5.14. Moreover, this update function can consider the time effect on the last evaluated trust value as in [Tan+16]; [LLL16]: any taken measure looses its value with time especially when judging trust.

  $$T_{DS} = \alpha * cT_{DS} + \beta * lT_{DS} \tag{5.14}$$

  Where cT_{DS} is the current trust level of a data service evaluated at the instant t, lT_{DS} is the last trust level of a data service evaluated at instant $t-1$, α and β weights attributed consecutively to these trust levels. They vary according to the importance that we give to each of these measures.

Users' requirements and expectations. Currently, the validity interval (T) for evaluating data freshness and the expected response time (ERt) for evaluating the time efficiency is fixed to $60s$ and $300ms$ respectively. Indeed, the data quality measuring module (respectively performance measuring module PMM) is programmed to evaluate data quality levels (respectively performance levels) independently from the incoming requests at the trust engine level. This limits DETECT and the possibility to perform more experiments for the validation of our solution.

As future work, DETECT must be developed into a framework that considers more requirements of data consumers. This framework should enable consumers to send requests with various requirements related to the data validity interval and the expected response time. These requirements include (1) the maximum data timeliness (T) and the expected response time value (ERt), and (2) the importance that is given to data quality and performance (i.e., the attributed weights α and β) in the trust model. Accordingly, the consumer receives a trust-ranked list of data services according to her requirements.

Scalability. At this time, we could not use any of the service datasets available for experiments since they relate solely to service performance. Therefore, data services are simulated and deployed using docker containers as a hosting environment. Docker limits the number of services that can be deployed due to performance constraints. Indeed, Docker is permitted to use a certain number of CPU cores of the host machine that it will share among the deployed servers and services. Moreover, in the second experiment, the number of parallel requests was limited to 400 requests due to performance constraints too. As future work, we must consider deploying the services on a different environment from *DETECT*'s hosting environment, or experiment on real-data services.

Chapter 6

Conclusion and Future Work

In recent years, data provisioning service environments have become popular. In contrast, data are produced under different conditions (e.g., production rate) and are inserted into the data services' database under other conditions (e.g., insertion rate). In addition, these data services are black boxes and present various QoS levels (e.g., response time, availability). Therefore, the process of selecting the most suitable data service given a user request guided by quality requirements is challenging. Indeed, multiple services can access the same type of data with heterogeneous offers related to data quality and QoS. Moreover, data are generally used for (more or less critical) decision making, and thus, data services must be trustworthy enough.

This thesis tackled essential research questions about trustworthy stream data services' selection challenge and resulted in two main contributions:

- A data quality evaluation model for black-box data services. We validate our model using *TUTOR*, a daTa qUaliTy Observability pRotocol.
- A trust evaluation model for black-box data services, validated using *DETECT*, a Data sErvice as a black box Trust Evaluation arChitecTure.

We validated these contributions in the context of medical data services related to the project SUMMIT funded by the region ARA.

6.1 Key Findings & Contributions

6.1.1 How to evaluate the quality of data focusing on their freshness for data accessed using black box data services?

Chapter 4 tackled this research question in detail by proposing a data quality evaluation model for black box data services. The proposed model selects the adequate service focusing on data freshness, which fits the requirements of the data service use in the scope of the SUMMIT project. Accordingly, multiple sub-problems were considered, namely:

"Which metrics to use for evaluating data freshness and how to compose them?": To answer the question, we proposed to measure data freshness using two timeliness metrics: *data timeliness*

and *database timeliness* because SUMMIT use cases' application requires timely data. We identified a correlation between these two metrics, and thus, data freshness is evaluated as the product of data timeliness and database timeliness.

"What meta-data (i.e., evidence) are available to evaluate these metrics?": We identified the information required to evaluate the timeliness metrics to answer the question. This information builds knowledge about the data service background, including data production timestamp and database update frequency. While data production timestamps are available, the database update frequency is unknown due to the black-box model.

"What method to adopt to collect evidence? When unavailable, is it possible to infer/estimate evidence indirectly (e.g., database update frequency)?": To answer the question: we have proposed a protocol named *TUTOR* that enables observing the timeliness metrics using random sampling techniques, including how many updates are detected for a service's database and data timeliness on average.

"How to evaluate data freshness metrics using the collected evidence?": To answer the question, we used the number of updates detected by TUTOR during an observation period to evaluate the database update frequency and data timeliness on average measures during an observation period to evaluate data timeliness. Thus, evaluating data quality for black-box data services.

"What is the cost versus the added-value ratio of our solution?": To answer the question, we have evaluated the efficiency and the effectiveness of TUTOR for different sampling techniques using its success rate and detection rate through experiments. The objective was to trade-off between having enough representative samples and reducing the cost of repeatedly accessing the data service and computing metrics. Therefore, the sampling technique that provided a good trade-off between the success and detection rates is selected to validate our TUTOR-based data quality evaluation model for black box data services.

6.1.2 How to evaluate the trust level of black-box data services?

Chapter 5 answers this question by proposing a trust evaluation model for black box data services using performance and data quality as trust factors.

"How to combine QoS and QoD to model the trust evaluation of black box data services while considering the user requirements and needs?": To answer the question, we have proposed to model the trust level evaluation as the weighted sum of these factors. Performance is evaluated using three metrics: time efficiency, availability, and task success ratio. Data quality is evaluated following the proposed evaluation model in our first contribution.

Our trust evaluation model has been implemented using *DETECT* architecture. Our experiments demonstrated our solution's ability to rank data services according to their trust levels, considering the different users' preferences and requirements (i.e., the importance given to trust factors). Moreover, we proved that our solution increased the user satisfaction level w.r.t the perceived QoS and data quality of the selected service via our trust-based solution. Finally, we evaluated the computation time overhead of our solution, comparing it to a random-based selection solution.

6.2 Future Work

These contributions help remarkably in the service selection challenge by differentiating between candidate services using their trust levels considering QoS and QoD. However, it is still possible to enhance the service selection process, and these potential improvements are detailed hereafter.

6.2.1 Modeling of the trust evaluation and its factors

We summarized certain limitations and perspectives in chapter 4 and chapter 5 and we summarise them as follows:

- **Evaluating the frequency of the data services' database update.** can be based on other estimators, including the Poisson estimator. However, to do so, we must prove that the databases' updates of data services follow a Poisson process.
- **Composing QoS and QoD.** Trust evaluation must be modeled considering the correlation between the service performance and data freshness discussed in chapter 5. As future work, this correlation should be studied in depth to learn the dependency between these two factors and how performance affects data freshness. This study should give insights and knowledge to adapt our trust evaluation model.
- Our trust evaluation model can be adapted to support other performance metrics and data quality dimensions.

6.2.2 Scalability

Chapter 4 and chapter 5 discussed the limitations related to the scalability of our approach. These limitations are due to the absence of real stream data services that we can use to experiment with our solutions. Therefore, the data services used in the various experiments are deployed locally with limited resources. Moreover, stream data accessed by these services are simulated, including data production and data insertion. This practical choice let us access real-time data and observe them on the fly. As future work, we will use real stream data services related to apnoea in the project SUMMIT.

Appendix A

Data Sampling

In this appendix, we present the existing sampling methods and technique.

Data sampling is a statistical method used to select and analyse a subset of data points (i.e. sample) in order to have an overview about the larger data set being analyzed. These elements are known as sample points or sampling or observations. Creating a sample is an efficient technique of conducting research. The process of deriving a sample is called a sampling method. A researcher usually chooses or selects a sample by using a pre-defined selection method. In most cases, it is impossible or costly and time-consuming to research the whole data set.

Sampling methods are regrouped into two distinct categories: probability sampling or random sampling methods and non-probability sampling or non-random sampling methods:

- Probability sampling is a method of deriving a sample where the data points are selected from the larger data set using the laws of probability. This sampling method includes every data point in the larger data set, and every data point has an equal chance of being selected.
- non-probability sampling method is derived mostly from the researcher's ability to get to this sample and and samples are selected based on his subjective judgment rather than random selection. This type of sampling is used for preliminary research where the primary objective is to derive a hypothesis about the topic in research and not every data point has an equal chance to be selected.

Probability sampling is used to make an accurate sample while non-probability sampling is used when the researcher does not intend to generate results that will generalize the entire data set. We are more interested in probability sampling method as we want to make an accurate data quality evaluation of the data source in question and samples are intended to generate observation and results that will generalize the entire data source state.

We identify four methods of probability sampling as depicted in figure.A.1:

- ***Simple random sampling***: each individual from a target population has an equal chance of being a part of the research study.

Example1: Selecting a data service from a service registry list full of 200 available services. In this example, the target data set (population) is the service registry list and the sample is random because each service has an equal chance of being selected

Example2: Accessing Alice's temperature via a data service at any given time from a given data source that captures and updates the temperature level on the fly. In this example, the target data set (population) is the data source and the sample is random because each captured temperature level has an equal chance of being selected at any moment

- ***Cluster sampling***: the correspondent population is divided into equal clusters. Clusters are indicative of homogeneous characteristics and have an equal chance of being a part of the sample. They are identified and included in a sample based on defining parameters such as data location, data type, etc.

  Example1: Selecting a data service from a service registry list full of 200 available services. Services can be categorized according to the location of their data source or the data type.

- ***Systematic sampling***: the researcher chooses individuals at equal intervals (i.e. system of intervals) from a population, the list of elements is "counted off". That is, every k*th* individual is taken. In this type of sampling, first we need to select the population, select a beginning number or time, select an interval (time or number) and gather samples at these chosen intervals.

  Example1: Accessing Alice's temperature via a data service every hour from a given data source that captures and updates the temperature level on the fly. In this example, the population is the data source and the sample is systematic because temperature level is selected every hour interval.

- ***Stratified random sampling***: dividing the respondent population into distinctive but pre-defined parameters in the research design phase.

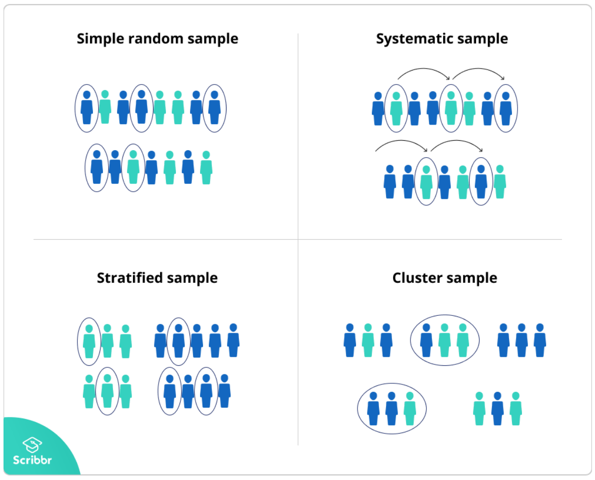


Figure A.1: Probability Sampling Methods

Bibliography

[AAT18] Osama AlFarraj, Ahmad AlZubi, and Amr Tolba. "Trust-based neighbor selection using activation function for secure routing in wireless sensor networks". In: *Journal of Ambient Intelligence and Humanized Computing* (2018), pp. 1–11 (cit. on p. 33).

[ABT17] Mohannad Alhanahnah, Peter Bertok, and Zahir Tari. "Trusting cloud service providers: trust phases and a taxonomy of trust factors". In: *IEEE Cloud Computing* 4.1 (2017), pp. 44–54 (cit. on pp. 33, 44).

[Ada+09] Eytan Adar et al. "The web changes everything: understanding the dynamics of web content". In: *Proceedings of the Second ACM International Conference on Web Search and Data Mining.* 2009, pp. 282–291 (cit. on p. 56).

[ADB18] Tooba Aamir, Hai Dong, and Athman Bouguettaya. "Trust in social-sensor cloud service". In: *2018 IEEE International Conference on Web Services (ICWS).* IEEE. 2018, pp. 359–362 (cit. on p. 23).

[ADC10a] Mohammed Alhamad, Tharam Dillon, and Elizabeth Chang. "Conceptual SLA framework for cloud computing". In: *Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on.* IEEE. 2010, pp. 606–610 (cit. on p. 42).

[ADC10b] Mohammed Alhamad, Tharam Dillon, and Elizabeth Chang. "Sla-based trust model for cloud computing". In: *2010 13th international conference on network-based information systems.* IEEE. 2010, pp. 321–324 (cit. on p. 18).

[AG07] Donovan Artz and Yolanda Gil. "A survey of trust in computer science and the semantic web". In: *Journal of Web Semantics* 5.2 (2007), pp. 58–71 (cit. on p. 30).

[Ahm+18] Waqas Ahmad et al. "Reputation-aware trust and privacy-preservation for mobile cloud computing". In: *IEEE Access* 6 (2018), pp. 46363–46381 (cit. on p. 31).

[Alh11] Mohammed Alhamad. "SLA-based trust model for secure cloud computing". PhD thesis. Curtin University, 2011 (cit. on p. 43).

[AMM07] Eyhab Al-Masri and Qusay H Mahmoud. "Qos-based discovery and ranking of web services". In: *2007 16th international conference on computer communications and networks.* IEEE. 2007, pp. 529–534 (cit. on pp. 22, 36).

[APT20] Konstantin Avrachenkov, Kishor Patil, and Gugan Thoppe. "Change rate estimation and optimal freshness in Web page crawling". In: *Proceedings of the 13th EAI International Conference on Performance Evaluation Methodologies and Tools.* 2020, pp. 3–10 (cit. on p. 37).

[Ard+18] Danilo Ardagna et al. "Context-aware data quality assessment for big data". In: *Future Generation Computer Systems* 89 (2018), pp. 548–562 (cit. on p. 23).

[Bao17a] Lihong Bao. "QoS-based trust computing scheme for SLA guarantee in cloud computing system". In: *2017 International Conference on Computing Intelligence and Information System (CIIS)*. IEEE. 2017, pp. 236–240 (cit. on pp. 36, 46).

[Bao17b] Lihong Bao. "QoS-based trust computing scheme for SLA guarantee in cloud computing system". In: *2017 International Conference on Computing Intelligence and Information System (CIIS)*. IEEE. 2017, pp. 236–240 (cit. on pp. 42, 43).

[Bar01] Jonathan Baron. "Measuring value tradeoffs: Problems and some solutions". In: *Conflict and tradeoffs in decision making: Essays in honor of Jane Beattie* (2001), pp. 231–259 (cit. on p. 72).

[Bau19] Paul C Bauer. "Conceptualizing trust and trustworthiness". In: (2019) (cit. on p. 30).

[BC00] Brian E Brewington and George Cybenko. "How dynamic is the Web?" In: *Computer Networks* 33.1-6 (2000), pp. 257–276 (cit. on p. 56).

[BEM05] Laure Berti-Equille and Fouzia Moussouni. "Quality-aware integration and warehousing of genomic data". In: *ICIQ'05-10th International Conference on Information Quality*. 2005, pp. 1–15 (cit. on p. 23).

[Ben+14] Nadia Bennani et al. "Sla-guided data integration on cloud environments". In: *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE. 2014, pp. 934–935 (cit. on p. 6).

[Ben+15] Nadia Bennani et al. "Towards a secure database integration using SLA in a multi-cloud context". In: *2015 IEEE 39th Annual Computer Software and Applications Conference*. Vol. 3. IEEE. 2015, pp. 4–9 (cit. on p. 6).

[Ben99] Paola Benassi. "TRUSTe: an online privacy seal program". In: *Communications of the ACM* 42.2 (1999), pp. 56–59 (cit. on p. 33).

[Ber03] Don M Berwick. "Improvement, trust, and the healthcare workforce". In: *BMJ Quality & Safety* 12.6 (2003), pp. 448–452 (cit. on p. 30).

[BFK98] Matt Blaze, Joan Feigenbaum, and Angelos D Keromytis. "KeyNote: Trust management for public-key infrastructures". In: *International Workshop on Security Protocols*. Springer. 1998, pp. 59–63 (cit. on p. 32).

[BL10] Elisa Bertino and Hyo-Sang Lim. "Assuring data trustworthiness-concepts and research challenges". In: *Workshop on Secure Data Management*. Springer. 2010, pp. 1–12 (cit. on pp. 39, 48).

[Bla01] Matt Blaze. *Using the KeyNote trust management system*. 2001 (cit. on p. 32).

[Bou04] Mokrane Bouzeghoub. "A framework for analysis of data freshness". In: *Proceedings of the 2004 international workshop on Information quality in information systems*. 2004, pp. 59–67 (cit. on p. 53).

[BP85] Donald P Ballou and Harold L Pazer. "Modeling data and process quality in multi-input, multi-output information systems". In: *Management science* 31.2 (1985), pp. 150–162 (cit. on p. 37).

[BR02] Laura Bright and Louiqa Raschid. “Using latency-recency profiles for data delivery on the web”. In: *VLDB’02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier. 2002, pp. 550–561 (cit. on p. 37).

[BVS13] Rajkumar Buyya, Christian Vecchiola, and S. Thamarai Selvi. “Chapter 1 - Introduction”. In: *Mastering Cloud Computing*. Ed. by Rajkumar Buyya, Christian Vecchiola, and S. Thamarai Selvi. Boston: Morgan Kaufmann, 2013, pp. 3–27 (cit. on p. 12).

[Cam+14] Ricardo Campos et al. “Survey of temporal information retrieval and related applications”. In: *ACM Computing Surveys (CSUR)* 47.2 (2014), pp. 1–41 (cit. on p. 56).

[Car+15] Daniel AS Carvalho et al. “Can Data Integration Quality Be Enhanced on Multi-cloud Using SLA?” In: *Database and Expert Systems Applications*. Springer. 2015, pp. 145–152 (cit. on p. 6).

[Car+16] Daniel AS Carvalho et al. “Rhone: A Quality-Based Query Rewriting Algorithm for Data Integration”. In: *East European Conference on Advances in Databases and Information Systems*. Springer. 2016, pp. 80–87 (cit. on pp. 6, 99).

[CB19] Soumi Chattopadhyay and Ansuman Banerjee. “QoS Value Prediction Using a Combination of Filtering Method and Neural Network Regression”. In: *International Conference on Service-Oriented Computing*. Springer. 2019, pp. 135–150 (cit. on p. 44).

[Ced+14] Priscila Cedillo et al. “Towards Monitoring Cloud Services using Models@ run. time.” In: *MoDELS@ Run. time*. 2014, pp. 31–40 (cit. on p. 40).

[CFP04] Cinzia Cappiello, Chiara Francalanci, and Barbara Pernici. “Data quality assessment from the user’s perspective”. In: *Proceedings of the 2004 international workshop on Information quality in information systems*. 2004, pp. 68–73 (cit. on pp. 23, 53).

[CGM03] Junghoo Cho and Hector Garcia-Molina. “Estimating frequency of change”. In: *ACM Transactions on Internet Technology (TOIT)* 3.3 (2003), pp. 256–290 (cit. on pp. 2, 91).

[CLC18] Ruo Jun Cai, Xue Jun Li, and Peter Han Joo Chong. “An evolutionary self-cooperative trust scheme against routing disruptions in MANETs”. In: *IEEE Transactions on Mobile Computing* 18.1 (2018), pp. 42–55 (cit. on p. 33).

[CN02] Junghoo Cho and Alexandros Ntoulas. “Effective change detection using sampling”. In: *VLDB’02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier. 2002, pp. 514–525 (cit. on pp. 56, 91).

[CN16] Matin Chiregi and Nima Jafari Navimipour. “A new method for trust and reputation evaluation in the cloud environments using the recommendations of opinion leaders’ entities and removing the effect of troll entities”. In: *Computers in Human Behavior* 60 (2016), pp. 280–292 (cit. on pp. 35, 36, 42, 46, 47).

[Cos03] Ana Cristina Costa. “Work team trust and effectiveness”. In: *Personnel review* (2003) (cit. on p. 30).

[CPB12] Oleksiy Chayka, Themis Palpanas, and Paolo Bouquet. “Defining and measuring data-driven quality dimension of staleness”. In: (2012) (cit. on pp. 23, 52, 53).

[CR12] Sudip Chakraborty and Krishnendu Roy. “An SLA-based framework for estimating trustworthiness of a cloud”. In: *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE. 2012, pp. 937–942 (cit. on pp. 36, 40, 47).

[CZ15] Li Cai and Yangyong Zhu. “The challenges of data quality and data quality assessment in the big data era”. In: *Data science journal* 14 (2015) (cit. on pp. 23, 52, 53).

[Dai+08] Chenyun Dai et al. “An approach to evaluate data trustworthiness based on data provenance”. In: *Workshop on Secure Data Management*. Springer. 2008, pp. 82–98 (cit. on pp. 39, 48).

[Dai+09] Chenyun Dai et al. “Assessing the trustworthiness of location data based on provenance”. In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2009, pp. 276–285 (cit. on pp. 39, 48).

[DBES09] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. “Truth discovery and copying detection in a dynamic world”. In: *Proceedings of the VLDB Endowment* 2.1 (2009), pp. 562–573 (cit. on p. 4).

[DF11] DAVID Dunning and Detlef Fetchenhauer. *Understanding the psychology of trust*. Psychology Press, 2011 (cit. on p. 30).

[DRP17] Gianni D’Angelo, Salvatore Rampone, and Francesco Palmieri. “Developing a trust model for pervasive computing based on Apriori association rules learning and Bayesian classification”. In: *Soft Computing* 21.21 (2017), pp. 6297–6315 (cit. on p. 31).

[Eva89] Paul Dennis Evans. “How business professionals decide to trust computer-based models developed by others”. PhD thesis. University of Michigan, 1989 (cit. on p. 31).

[Feh09] Ernst Fehr. “On the economics and biology of trust”. In: *Journal of the european economic association* 7.2-3 (2009), pp. 235–266 (cit. on p. 30).

[Fei98] Joan Feigenbaum. “Overview of the AT&T Labs trust-management project”. In: *International Workshop on Security Protocols*. Springer. 1998, pp. 51–58 (cit. on p. 32).

[FG06] Carlos Flavián and Miguel Guinalíu. “Consumer trust, perceived security and privacy policy: three basic elements of loyalty to a web site”. In: *Industrial management & data Systems* (2006) (cit. on p. 42).

[Fis95] Gregory W Fischer. “Range sensitivity of attribute weights in multiattribute value models”. In: *Organizational Behavior and Human Decision Processes* 62.3 (1995), pp. 252–266 (cit. on p. 72).

[Gal06] Stefania Galizia. “WSTO: A classification-based ontology for managing trust in semantic web services”. In: *European semantic web conference*. Springer. 2006, pp. 697–711 (cit. on p. 42).

[Ger91] Anne Geraci. *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. IEEE Press, 1991 (cit. on p. 95).

[GF08] Carrie Grimes and Daniel Ford. “Estimation of Web page change rates”. In: (2008) (cit. on p. 56).

[GFT08] Carrie Grimes, Daniel Ford, and Eric Tassone. “Keeping a search engine fresh: Risk and optimality in estimating refresh rates for web pages”. In: (2008) (cit. on p. 56).

[GGD07] Stefania Galizia, Alessio Gugliotta, and John Domingue. “A trust based methodology for web service selection”. In: *International conference on semantic computing (ICSC 2007)*. IEEE. 2007, pp. 193–200 (cit. on pp. 33, 43).

[Gid91] Anthony Giddens. *Modernity and self-identity: Self and society in the late modern age*. Stanford university press, 1991 (cit. on p. 30).

[Gol+18] Dagaen Golomb et al. “Data freshness over-engineering: Formulation and results”. In: *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE. 2018, pp. 174–183 (cit. on p. 37).

[Han+14] Guangjie Han et al. “Management and applications of trust in Wireless Sensor Networks: A survey”. In: *Journal of Computer and System Sciences* 80.3 (2014), pp. 602–617 (cit. on pp. 31, 33).

[Her+00] Amir Herzberg et al. “Access control meets public key infrastructure, or: Assigning roles to strangers”. In: *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*. IEEE. 2000, pp. 2–14 (cit. on p. 32).

[HHH14] Walayat Hussain, Farookh Khadeer Hussain, and Omar Khadeer Hussain. “Maintaining trust in cloud computing through SLA monitoring”. In: *International Conference on Neural Information Processing*. Springer. 2014, pp. 690–697 (cit. on p. 40).

[HMS02] Mark Hansen, Stuart Madnick, and Michael Siegel. “Data integration using web services”. In: *Efficiency and effectiveness of XML tools and techniques and data integration over the web*. Springer, 2002, pp. 165–182 (cit. on p. 21).

[HNT18] Florian Hawlitschek, Benedikt Notheisen, and Timm Teubner. “The limits of trust-free systems: A literature review on blockchain technology and trust in the sharing economy”. In: *Electronic commerce research and applications* 29 (2018), pp. 50–63 (cit. on p. 31).

[HRM11] Sheikh Mahbub Habib, Sebastian Ries, and Max Muhlhauser. “Towards a trust management system for cloud computing”. In: *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE. 2011, pp. 933–939 (cit. on p. 42).

[HS05] Michael N Huhns and Munindar P Singh. “Service-oriented computing: Key concepts and principles”. In: *IEEE Internet computing* 9.1 (2005), pp. 75–81 (cit. on p. 12).

[Jai+16] Sushma Jain et al. “A trust model in cloud computing based on fuzzy logic”. In: *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE. 2016, pp. 47–52 (cit. on p. 44).

[Jay+17] Upul Jayasinghe et al. “Data centric trust evaluation and prediction framework for IoT”. In: *2017 ITU Kaleidoscope: Challenges for a Data-Driven Society (ITU K)*. IEEE. 2017, pp. 1–7 (cit. on pp. 31, 32, 38).

[Jay+18] Upul Jayasinghe et al. “Machine learning based trust computational model for IoT services”. In: *IEEE Transactions on Sustainable Computing* 4.1 (2018), pp. 39–52 (cit. on p. 44).

[JG11] Wayne Jansen and Timothy Grance. “Draft NIST special publication guidelines on security and privacy in public Cloud computing”. In: *Computer Security, Jan* (2011) (cit. on pp. 22, 47).

[JHS14] Obed Jules, Abdelhakim Hafid, and Mohamed Adel Serhani. “Bayesian network, and probabilistic ontology driven trust model for sla management of cloud services”. In: *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*. IEEE. 2014, pp. 77–83 (cit. on p. 44).

[Jin+18] Hao Jin et al. “Full integrity and freshness for cloud data”. In: *Future Generation Computer Systems* 80 (2018), pp. 640–652 (cit. on pp. 23, 38, 40, 41, 91).

[JK02] Kalervo Järvelin and Jaana Kekäläinen. “Cumulated gain-based evaluation of IR techniques”. In: *ACM Transactions on Information Systems (TOIS)* 20.4 (2002), pp. 422–446 (cit. on p. 82).

[JM16] Yu Jin and Shaoying Min. “Stadam: A new SLA trust model based on anomaly detection and multi-cloud”. In: *2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI)*. IEEE. 2016, pp. 396–399 (cit. on pp. 43, 46).

[JoS+10] Anthony D JoSEP et al. “A view of cloud computing”. In: *Communications of the ACM* 53.4 (2010) (cit. on p. 47).

[Kas+15] Ubaidullah Alias Kashif et al. “Distributed trust protocol for IaaS cloud computing”. In: *2015 12th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. IEEE. 2015, pp. 275–279 (cit. on p. 33).

[Kha16] Fiaz Khan. “Service Level Agreement in Cloud Computing: A Survey”. In: *International Journal of Computer Science and Information Security (IJCSIS)* 14 (July 2016), pp. 324–330 (cit. on p. 18).

[Kim+20] Minsu Kim et al. “Ensuring Data Freshness for Blockchain-enabled Monitoring Networks”. In: *arXiv preprint arXiv:2011.07012* (2020) (cit. on p. 37).

[KL03] Alexander Keller and Heiko Ludwig. "The WSLA framework: Specifying and monitoring service level agreements for web services". In: *Journal of Network and Systems Management* 11.1 (2003), pp. 57–81 (cit. on p. 40).

[KT09] Damjan Kovač and Denis Trček. "Qualitative trust modeling in SOA". In: *Journal of systems architecture* 55.4 (2009), pp. 255–263 (cit. on pp. 31, 42, 46).

[Kub+18] Sylvain Kubler et al. "Comparison of metadata quality in open data portals using the Analytic Hierarchy Process". In: *Government Information Quarterly* 35.1 (2018), pp. 13–29 (cit. on p. 53).

[Lar98] Kent D Larson. "The role of service level agreements in IT service delivery". In: *Information Management & Computer Security* 6.3 (1998), pp. 128–132 (cit. on p. 18).

[LCRS96] Margaret Levi and Institut universitaire européen. Centre Robert Schuman. *A state of trust.* Citeseer, 1996 (cit. on p. 30).

[Li+14a] Changsong Li et al. "Trust evaluation model of cloud manufacturing service platform". In: *The international journal of advanced manufacturing technology* 75.1 (2014), pp. 489–501 (cit. on pp. 44–46).

[Li+14b] Xiaoyong Li et al. "Service operator-aware trust scheme for resource matchmaking across multiple clouds". In: *IEEE transactions on parallel and distributed systems* 26.5 (2014), pp. 1419–1429 (cit. on p. 42).

[Li+15] Xiaoyong Li et al. "Data-driven and feedback-enhanced trust computing pattern for large-scale multi-cloud collaborative services". In: *IEEE Transactions on Services Computing* 11.4 (2015), pp. 671–684 (cit. on p. 36).

[Li+18] Xiaoyong Li et al. "Fast and parallel trust computing scheme based on big data analysis for collaboration cloud service". In: *IEEE Transactions on Information Forensics and Security* 13.8 (2018), pp. 1917–1931 (cit. on pp. 22, 36, 40, 41, 47, 97).

[Li18] Xin Li. "Access Control Strategy Based on Trust under Cloud Computing Platform". In: *2018 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*. IEEE. 2018, pp. 327–330 (cit. on pp. 31, 32).

[Lin+14] Guoyuan Lin et al. "MTBAC: A mutual trust based access control model in cloud computing". In: *China Communications* 11.4 (2014), pp. 154–162 (cit. on pp. 32, 42).

[Lin+16] Sebastian Lins et al. "Dynamic certification of cloud services: Trust, but verify!" In: *IEEE Security & Privacy* 14.2 (2016), pp. 66–71 (cit. on p. 42).

[Liu+10] Yuhua Liu et al. "The incentive secure mechanism based on quality of service in P2P network". In: *Computers & Mathematics with Applications* 60.2 (2010), pp. 224–233 (cit. on p. 31).

[LLG09] Deyi Li, Changyu Liu, and Wenyan Gan. "A new cognitive model: Cloud model". In: *International journal of intelligent systems* 24.3 (2009), pp. 357–375 (cit. on p. 44).

[LLG+19] Guohui Li, Jianjun Li, Bing Guo, et al. "Maintaining data freshness in distributed cyber-physical systems". In: *IEEE Transactions on Computers* 68.7 (2019), pp. 1077–1090 (cit. on p. 37).

[LLL16] Li Liao, Bixin Li, and Chao Li. "A model to evaluate the credibility of service in cloud computing environment". In: *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech).* IEEE. 2016, pp. 294–301 (cit. on pp. 36, 118).

[LMB10] Hyo-Sang Lim, Yang-Sae Moon, and Elisa Bertino. "Provenance-based trustworthiness assessment in sensor networks". In: *Proceedings of the Seventh International Workshop on Data Management for Sensor Networks.* 2010, pp. 2–7 (cit. on pp. 39, 48).

[LMF07] Mikko O Lehtonen, Florian Michahelles, and Elgar Fleisch. "Trust and security in RFID-based product authentication systems". In: *IEEE Systems Journal* 1.2 (2007), pp. 129–144 (cit. on p. 31).

[LP09] Wenjuan Li and Lingdi Ping. "Trust model to enhance security and interoperability of cloud environment". In: *IEEE international conference on cloud computing.* Springer. 2009, pp. 69–79 (cit. on p. 42).

[LR03] Alexandros Labrinidis and Nick Roussopoulos. "Balancing performance and data freshness in web database servers". In: *Proceedings 2003 VLDB Conference.* Elsevier. 2003, pp. 393–404 (cit. on p. 37).

[LR04] Alexandros Labrinidis and Nick Roussopoulos. "Exploring the tradeoff between performance and data freshness in database-driven web servers". In: *The VLDB Journal* 13.3 (2004), pp. 240–255 (cit. on p. 23).

[LS07] Huaizhi Li and Mukesh Singhal. "Trust management in distributed systems". In: *Computer* 40.2 (2007), pp. 45–53 (cit. on p. 31).

[Lud+03] Heiko Ludwig et al. "Web service level agreement (WSLA) language specification". In: *Ibm corporation* (2003), pp. 815–824 (cit. on p. 18).

[Man15] Paul Manuel. "A trust model of cloud computing based on Quality of Service". In: *Annals of Operations Research* 233.1 (2015), pp. 281–292 (cit. on pp. 2, 22, 35, 41, 43, 45, 47, 95, 112).

[Mao+17] Chengying Mao et al. "Towards a trust prediction framework for cloud services based on PSO-driven neural network". In: *IEEE Access* 5 (2017), pp. 2187–2199 (cit. on p. 44).

[Mat05] Norman Matloff. "Estimation of internet file-access/modification rates from indirect data". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 15.3 (2005), pp. 233–253 (cit. on p. 56).

[May90] Frank L Mayer. “A brief comparison of two different environmental guidelines for determining’levels of trust’(computer security)”. In: *[1990] Proceedings of the Sixth Annual Computer Security Applications Conference*. IEEE. 1990, pp. 244–250 (cit. on pp. 31, 32).

[MDZ93] Christine Moorman, Rohit Deshpande, and Gerald Zaltman. “Factors affecting trust in market research relationships”. In: *Journal of marketing* 57.1 (1993), pp. 81–101 (cit. on p. 33).

[Men02] D.A. Menasce. “QoS issues in Web services”. In: *IEEE Internet Computing* 6.6 (2002), pp. 72–75 (cit. on p. 22).

[MG00] Maria Madsen and Shirley Gregor. “Measuring human-computer trust”. In: *11th australasian conference on information systems*. Vol. 53. Citeseer. 2000, pp. 6–8 (cit. on p. 31).

[MG+11] Peter Mell, Tim Grance, et al. “The NIST definition of cloud computing”. In: (2011) (cit. on p. 17).

[MMY09] Jian Meng, Shujun Mei, and Zhao Yan. “Restful web services: A solution for distributed data integration”. In: *2009 International Conference on Computational Intelligence and Software Engineering*. IEEE. 2009, pp. 1–4 (cit. on p. 16).

[MRR15] John Mitchell, Syed Rizvi, and Jungwoo Ryoo. “A fuzzy-logic approach for evaluating a cloud service provider”. In: *2015 1st International Conference on Software Security and Assurance (ICSSA)*. IEEE. 2015, pp. 19–24 (cit. on p. 44).

[MSAEB09] Paul D Manuel, S Thamarai Selvi, and Mostafa Ibrahim Abd-El Barr. “Trust management system for grid and cloud resources”. In: *2009 First International Conference on Advanced Computing*. IEEE. 2009, pp. 176–181 (cit. on p. 36).

[MSS17] Manel Mrabet, Yosra Ben Saied, and Leila Azouz Saidane. “Modeling correlation between QoS attributes for trust computation in cloud computing environments”. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2017, pp. 488–497 (cit. on pp. 35, 36).

[NCO04] Alexandros Ntoulas, Junghoo Cho, and Christopher Olston. “What’s new on the Web? The evolution of the Web from a search engine perspective”. In: *Proceedings of the 13th international conference on World Wide Web*. 2004, pp. 1–12 (cit. on p. 56).

[Ng06] Alex Ng. “Challenges to data transport in supporting heterogeneous Web Services protocol stacks”. PhD thesis. Mar. 2006 (cit. on p. 18).

[Noo+13] Talal H Noor et al. “Cloud armor: a platform for credibility-based trust management of cloud services”. In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. 2013, pp. 2509–2512 (cit. on pp. 42, 46).

[NU16] Sebastian Neumaier and Jürgen Umbrich. “Measures for assessing the data freshness in Open Data portals”. In: *2016 2nd International Conference on Open and Big Data (OBD)*. IEEE. 2016, pp. 17–24 (cit. on pp. 23, 38, 72).

[NV11] Aarthi Nagarajan and Vijay Varadharajan. “Dynamic trust enhanced security model for trusted platform based services”. In: *Future Generation Computer Systems* 27.5 (2011), pp. 564–573 (cit. on p. 42).

[Pap+08] Michael P Papazoglou et al. “Service-oriented computing: a research roadmap”. In: *International Journal of Cooperative Information Systems* 17.02 (2008), pp. 223–255 (cit. on p. 12).

[Pat+18] Muhammad Salman Pathan et al. “An efficient trust-based scheme for secure and quality of service routing in MANETs”. In: *Future Internet* 10.2 (2018), p. 16 (cit. on p. 42).

[Per+04] Verónika Peralta et al. “A Framework for Data Quality Evaluation in a Data Integration System.” In: *SBBD*. 2004, pp. 134–147 (cit. on pp. 1, 24, 37, 38, 41, 52, 53).

[Per06] Verónika Peralta. “Data quality evaluation in data integration systems”. PhD thesis. Université de Versailles-Saint Quentin en Yvelines; Université de la . . ., 2006 (cit. on pp. 41, 52).

[PLW02] Leo L Pipino, Yang W Lee, and Richard Y Wang. “Data quality assessment”. In: *Communications of the ACM* 45.4 (2002), pp. 211–218 (cit. on p. 23).

[Pow+11] Jason L Powell et al. “Towards a sociology of trust: community care and managing diversity”. In: *Sociology Mind* 1.02 (2011), p. 27 (cit. on p. 30).

[PSB20] Tao Peng, Sana Sellami, and Omar Boucelma. “Trust Assessment on Streaming Data: A Real Time Predictive Approach”. In: *International Workshop on Advanced Analytics and Learning on Temporal Data*. Springer. 2020, pp. 204–219 (cit. on pp. 38, 39).

[PWE13] WP Eureka Priyadarshani, Gihan N Wikramanayake, and EM Piyal Ekanayake. “Measuring Trust and Selecting CloudDatabase Services”. In: *Advances in Computer Science: an International Journal* 2 (2013), pp. 114–120 (cit. on p. 31).

[QB14] Chenhao Qu and Rajkumar Buyya. “A cloud trust evaluation system using hierarchical fuzzy inference system for service selection”. In: *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*. IEEE. 2014, pp. 850–857 (cit. on pp. 33, 44).

[QHC06] Daniele Quercia, Stephen Hailes, and Licia Capra. “B-trust: Bayesian trust framework for pervasive computing”. In: *International Conference on Trust Management*. Springer. 2006, pp. 298–312 (cit. on p. 31).

[QWO13] Lie Qu, Yan Wang, and Mehmet A Orgun. “Cloud service selection based on the aggregation of user feedback and quantitative performance assessment”. In: *2013 IEEE International Conference on Services Computing*. IEEE. 2013, pp. 152–159 (cit. on p. 42).

[RB13] Kira Radinsky and Paul N Bennett. “Predicting content change on the web”. In: *Proceedings of the sixth ACM international conference on Web search and data mining*. 2013, pp. 415–424 (cit. on p. 56).

[RB19] Sanjoy Roy and Suparna Biswas. "A novel trust evaluation model based on data freshness in WBAN". In: *Proceedings of International Ethical Hacking Conference 2018*. Springer. 2019, pp. 223–232 (cit. on p. 37).

[Ren19] Steffen Rendle. "Evaluation metrics for item recommendation under sampling". In: *arXiv preprint arXiv:1912.02263* (2019) (cit. on p. 75).

[RHJ04] Sarvapali D Ramchurn, Dong Huynh, and Nicholas R Jennings. "Trust in multi-agent systems". In: *The knowledge engineering review* 19.1 (2004), pp. 1–25 (cit. on p. 31).

[RMVS05] Yacine Rebahi, Vicente E Mujica-V, and Dorgham Sisalem. "A reputation-based trust mechanism for ad hoc networks". In: *10th IEEE Symposium on Computers and Communications (ISCC'05)*. IEEE. 2005, pp. 37–42 (cit. on p. 33).

[Rot67] Julian B Rotter. "A new scale for the measurement of interpersonal trust." In: *Journal of personality* (1967) (cit. on p. 30).

[Rou+98] Denise M Rousseau et al. "Not so different after all: A cross-discipline view of trust". In: *Academy of management review* 23.3 (1998), pp. 393–404 (cit. on pp. 30, 31).

[RPL17] V Ram Prabha and P Latha. "Enhanced multi-attribute trust protocol for malicious node detection in wireless sensor networks." In: *Sadhana* 42.2 (2017) (cit. on p. 33).

[RS09] Paul Resnick and Rahul Sami. "Sybilproof transitive trust protocols". In: *Proceedings of the 10th ACM conference on Electronic Commerce*. 2009, pp. 345–354 (cit. on p. 33).

[SAJM13] Mohd Izuan Mohd Saad, Kamarularifin Abd Jalil, and Mazani Manaf. "Data provenance trusted model in cloud computing". In: *2013 International Conference on Research and Innovation in Information Systems (ICRIIS)*. IEEE. 2013, pp. 257–262 (cit. on pp. 38, 48).

[Sal+05] Al F Salam et al. "Trust in e-commerce". In: *Communications of the ACM* 48.2 (2005), pp. 72–77 (cit. on p. 31).

[SC17] Ashish Singh and Kakali Chatterjee. "A multi-dimensional trust and reputation calculation model for cloud computing environments". In: *2017 ISEA Asia Security and Privacy (ISEASP)*. IEEE. 2017, pp. 1–8 (cit. on pp. 33, 46).

[SD18] Archana B Saxena and Meenu Dawe. "IAAS trust in public domain: evaluative framework for service provider". In: *2018 IEEE 18th International Conference on Advanced Learning Technologies (ICALT)*. IEEE. 2018, pp. 458–460 (cit. on p. 36).

[Ser+16] Damián Serrano et al. "SLA guarantees for cloud services". In: *Future Generation Computer Systems* 54 (2016), pp. 233–246 (cit. on p. 42).

[SG05] Ren Saint-Germain. "Information security management best practice based on ISO". In: *IEC* 17799 (2005), pp. 60–66 (cit. on p. 22).

[Sha00] Umesh Maheshwari Radek Vingralek William Shapiro. "How to build a trusted database system on untrusted storage". In: (2000) (cit. on p. 33).

[Sha+08] Riaz Ahmed Shaikh et al. “Group-based trust management scheme for clustered wireless sensor networks”. In: *IEEE transactions on parallel and distributed systems* 20.11 (2008), pp. 1698–1712 (cit. on p. 33).

[Sin07] Sanasam Ranbir Singh. “Estimating the Rate of Web Page Updates.” In: *IJCAI.* Vol. 7. 2007, pp. 2874–2879 (cit. on pp. 37, 56).

[Som+18] Nivethitha Somu et al. “A trust centric optimal service ranking approach for cloud service selection”. In: *Future Generation Computer Systems* 86 (2018), pp. 234–252 (cit. on p. 33).

[SP12] Jane Siegel and Jeff Perdue. “Cloud services measures for global use: the service measurement index (SMI)”. In: *2012 Annual SRII global conference.* IEEE. 2012, pp. 411–415 (cit. on p. 22).

[SS17a] Jagpreet Sidhu and Sarbjeet Singh. “Design and comparative analysis of MCDM-based multi-dimensional trust evaluation schemes for determining trustworthiness of cloud service providers”. In: *Journal of Grid Computing* 15.2 (2017), pp. 197–218 (cit. on p. 44).

[SS17b] Sarbjeet Singh and Jagpreet Sidhu. “Compliance-based multi-dimensional trust evaluation system for determining trustworthiness of cloud service providers”. In: *Future Generation Computer Systems* 67 (2017), pp. 109–132 (cit. on p. 36).

[Tam18] Larysa Tamilina. “A brief overview of approaches to defining social trust”. In: (2018) (cit. on p. 30).

[Tan+16] Zhenhua Tan et al. “A novel trust model based on sla and behavior evaluation for clouds”. In: *2016 14th Annual Conference on Privacy, Security and Trust (PST).* IEEE. 2016, pp. 581–587 (cit. on pp. 35, 47, 118).

[Tan+17] Zhen-Hua Tan et al. “MCTModel: A Multi-clouds Trust Model Based on SLA in Cloud Computing”. In: *Journal of Computers* 28.6 (2017), pp. 236–245 (cit. on pp. 35, 40, 41, 47).

[TE06] Electra Tamani and Paraskevas Evripidou. “Applying trust mechanisms in an agent-based p2p network of service providers and requestors”. In: *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID’06).* Vol. 2. IEEE. 2006, pp. 13–13 (cit. on p. 31).

[Tia+20] Sen Tian et al. “Random Sampling-Arithmetic Mean: A Simple Method of Meteorological Data Quality Control Based on Random Observation Thought”. In: *IEEE Access* (2020) (cit. on p. 91).

[Trc11] Denis Trcek. “Trust management in the pervasive computing era”. In: *IEEE security & Privacy* 9.4 (2011), pp. 52–55 (cit. on p. 32).

[VHA05] Le-Hung Vu, Manfred Hauswirth, and Karl Aberer. “QoS-based service selection and ranking with trust and reputation management”. In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems".* Springer. 2005, pp. 466–483 (cit. on pp. 33, 42, 46).

[Wan+08] Shouxin Wang et al. "An evaluation approach of subjective trust based on cloud model". In: *2008 international conference on computer science and software engineering*. Vol. 3. IEEE. 2008, pp. 1062–1068 (cit. on p. 44).

[Wan+11] Shangguang Wang et al. "Cloud model for service selection". In: *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*. IEEE. 2011, pp. 666–671 (cit. on p. 33).

[Wan+19] Tian Wang et al. "Crowdsourcing mechanism for trust evaluation in CPCS based on intelligent mobile edge computing". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.6 (2019), pp. 1–19 (cit. on p. 31).

[Wan+20] Tian Wang et al. "A novel trust mechanism based on fog computing in sensor–cloud system". In: *Future Generation Computer Systems* 109 (2020), pp. 573–582 (cit. on p. 33).

[Wer18] Kevin Werbach. "Trust, but verify: Why the blockchain needs the law". In: *Berkeley Tech. LJ* 33 (2018), p. 487 (cit. on p. 31).

[WZ16] Zhengping Wu and Yu Zhou. "Service trustworthiness evaluation using neural network and fuzzy logic". In: *2016 IEEE International Conference on Services Computing (SCC)*. IEEE. 2016, pp. 563–570 (cit. on p. 44).

[WZM08] Shouxin Wang, Li Zhang, and Na Ma. "A quantitative measurement for reputation of Web Service and providers based on cloud model". In: *2008 International Conference on Computational Intelligence for Modelling Control & Automation*. IEEE. 2008, pp. 500–505 (cit. on p. 31).

[YJ05] Geng Yang and Sirkka L Jarvenpaa. "Trust and radio frequency identification (RFID) adoption within an alliance". In: *Proceedings of the 38th annual Hawaii international conference on system sciences*. IEEE. 2005, 208a–208a (cit. on p. 31).

[YW03] Ting Yu and Marianne Winslett. "A unified scheme for resource protection in automated trust negotiation". In: *2003 Symposium on Security and Privacy, 2003*. IEEE. 2003, pp. 110–122 (cit. on p. 33).

[YZV14] Zheng Yan, Peng Zhang, and Athanasios V Vasilakos. "A survey on trust management for Internet of Things". In: *Journal of network and computer applications* 42 (2014), pp. 120–134 (cit. on p. 31).

[Zha+15] Xin Zhao et al. "Toward SLA-constrained service composition: An approach based on a fuzzy linguistic preference model and an evolutionary algorithm". In: *Information Sciences* 316 (2015), pp. 370–396 (cit. on p. 44).

[Zhe+12] Zibin Zheng et al. "QoS ranking prediction for cloud services". In: *IEEE transactions on parallel and distributed systems* 24.6 (2012), pp. 1213–1222 (cit. on pp. 40, 41).

[ZHI18] Shams Zawoad, Ragib Hasan, and Kamrul Islam. "Secprov: Trustworthy and efficient provenance management in the cloud". In: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE. 2018, pp. 1241–1249 (cit. on pp. 38, 48).

[Zho+20] Zimeng Zhou et al. “Maintaining Real-Time Data Freshness in Wireless Powered Communication Networks”. In: *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE. 2020, pp. 166–177 (cit. on p. 37).

[Zhu+14] Chunsheng Zhu et al. “A trust and reputation management system for cloud and sensor networks integration”. In: *2014 IEEE International Conference on Communications (ICC)*. IEEE. 2014, pp. 557–562 (cit. on p. 46).

[Zim10] H-J Zimmermann. “Fuzzy set theory”. In: *Wiley Interdisciplinary Reviews: Computational Statistics* 2.3 (2010), pp. 317–332 (cit. on p. 44).

[ZKZ18] PeiYun Zhang, Yang Kong, and MengChu Zhou. “A domain partition-based trust model for unreliable clouds”. In: *IEEE Transactions on Information Forensics and Security* 13.9 (2018), pp. 2167–2178 (cit. on p. 31).

[ZWJ17] Wei Zong, Feng Wu, and Zhengrui Jiang. “A Markov-based update policy for constantly changing database systems”. In: *IEEE Transactions on Engineering Management* 64.3 (2017), pp. 287–300 (cit. on p. 37).

[ZYS18] Jing Zhong, Roy D Yates, and Emina Soljanin. “Two freshness metrics for local cache refresh”. In: *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2018, pp. 1924–1928 (cit. on p. 37).

[ZZF18] PeiYun Zhang, MengChu Zhou, and Giancarlo Fortino. “Security and trust issues in Fog computing: A survey”. In: *Future Generation Computer Systems* 88 (2018), pp. 16–27 (cit. on p. 31).

Résumé :

1. Contexte générale et problématiques de recherche :

Ces dernières années ont été marquées par une croissance exponentielle de services de données en flux continu, issues du monde physique. Cette mouvance a accru la difficulté de leur sélection en réponse à des requêtes complexes, qui soit en adéquation avec les attentes et les exigences qualitatives des consommateurs. En effet, lesdits services permettent l'accès et le recueil de données en temps réel souvent collectées dans différentes conditions en termes de fraîcheur, provenance, sécurité, performance du service, etc.

En outre, les données obtenues grâce à ces services sont généralement utilisées pour prendre des décisions importantes voire critiques, exigeant de ce fait la sélection de services fiables (de confiance). Un service de flux de données est dit fiable dès lors qu'il est conforme aux conditions QoS promises par son fournisseur et donne accès à des données actualisées (à jour). Cependant, les services sont souvent fournis et déployés dans divers environnements en adoptant le modèle de la boîte noire. Ce dernier modèle crée des obstacles supplémentaires dans la mesure où ces services n'exposent ni n'exportent de (méta)-données sur les conditions dans lesquelles ils recueillent ces données, ni sur la qualité desdites données fournies.

Partant de ce constat, l'objectif de la présente thèse est de proposer une solution répondant aux défis sous-jacents à la sélection de services de flux de données fiables. Plus précisément, étant donnée une requête utilisateur, cette solution doit permettre de : (1) calculer la fiabilité des services de flux de données en utilisant leur performance et la qualité des données qu'ils fournissent, et (2) classer les services en fonction de leur niveau de fiabilité.

A cette fin, nos travaux de recherche se sont focalisés sur trois problématiques complémentaires illustrées par la Figure 1, et portant sur : (1) la définition d'un modèle d'évaluation de la qualité des données pour les services de flux de données, (2) compte tenu du caractère boîte noire des services, la proposition de protocoles et de stratégies pour la recueil des informations nécessaires pour l'évaluation, et la façon dont elles sont utilisées pour cette évaluation, et (3) la définition d'un modèle d'évaluation de la fiabilité pour les services de données de type « boîte noire » alliant performance de services et qualité des données.

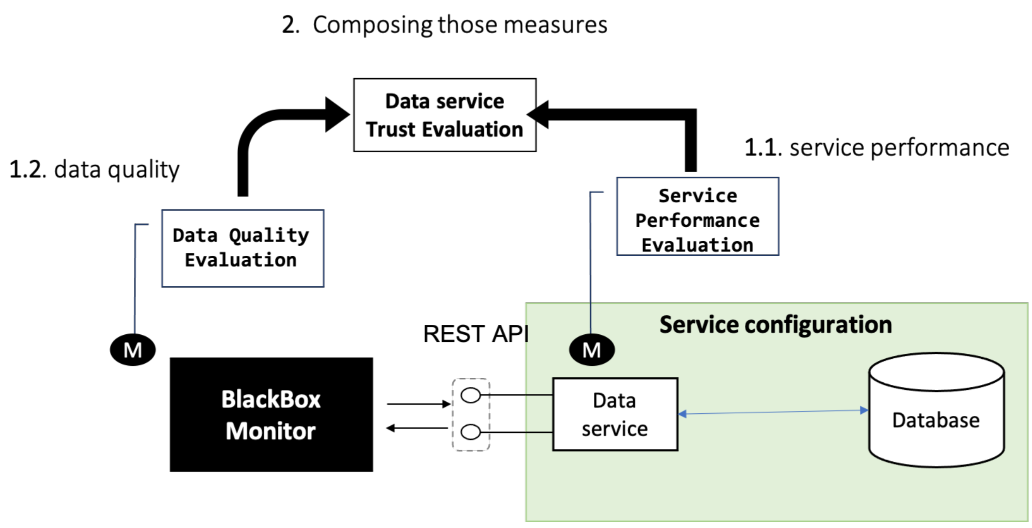


Figure 1 : Problématiques de recherche

2. Contributions :

2.1. Modèle d'évaluation de la qualité des données

En réponse à ces problématiques, nous avons proposé dans un **premier lieu** un modèle d'évaluation de la qualité des données pour les services de flux de données axé principalement sur la fraîcheur des données. Ainsi, la connaissance de la qualité des données fournies par les services pour une requête est nécessaire pour déterminer la fiabilité desdites données et si elles sont adaptées à une tâche donnée. Néanmoins, le processus d'évaluation de la qualité des données est difficile lorsque cette évaluation est effectuée dans des environnements IoT, Big Data et Cloud Computing qui introduisent l'hétérogénéité de la qualité des données en raison du croissement du nombre de producteurs de données. De plus, comme mentionné précédemment, les services de flux de données dans ces environnements produisent des données en continu dans différentes conditions de qualité de fraîcheur, sécurité, confidentialité, provenance, etc. En conséquence, les données sont fournies sous différents niveaux de qualité.

L'évaluation de la qualité des données dépend en général du contexte et peut être réalisée en utilisant différentes dimensions de qualité, notamment l'exactitude, la fraîcheur, etc. Selon l'environnement (entre autres), l'application et les besoins des utilisateurs cibles, la définition de la qualité des données et son modèle d'évaluation diffèrent. Ainsi, le processus à adopter est le suivant : (1) définir et connaître les exigences et les besoins des utilisateurs, le contexte environnemental, etc., (2) sélectionner les dimensions de qualité en conséquence, et (3) utiliser ces dimensions de qualité pour définir le modèle d'évaluation de la qualité des données.

Le cadre de notre étude s'inscrivant dans le domaine médical, les résultats sont utilisés par des applications pour lesquelles des données continues se doivent d'être à jour (p. ex., les applications de e-santé). Ainsi, nous modélisons la qualité des données en utilisant uniquement la dimension temporelle axée sur la fraîcheur des données. Par définition, la fraîcheur des données indique dans quelle mesure les données sont à jour en ce sens qu'elles soient utiles d'une application cible. Le principe sous-jacent est que les données fraîches sont précieuses et dignes de confiance contrairement aux données périmées qui perdent leur valeur au fil du temps et peuvent donc affecter négativement les décisions (critiques) prises en les utilisant. Par conséquent, la fraîcheur des données est liée au degré d'actualité de données qu'un service peut garantir.

L'évaluation de l'actualité peut être réalisée à partir des connaissances que nous avons sur la configuration des sources de données (p. ex., le taux de production) et les conditions dans lesquelles les services de données recueillent et stockent continuellement ces données dans des bases de données (p. ex., le taux d'insertion, etc.). Cependant, les services de données sont des boîtes noires et ne partagent pas ces informations sur la configuration des données. Par conséquent, il est essentiel de (1) définir un modèle d'évaluation de la qualité des données en utilisant la fraîcheur des données pour les services de données continus, et (2) définir un protocole d'observabilité pour la collecte des preuves nécessaires à cette évaluation.

Les solutions existantes liées à l'évaluation de la qualité des données cibleraient une application ou un domaine à la fois, et à notre connaissance, aucun modèle d'évaluation de la qualité des données n'a été proposé pour les services de données continus en mode boîte noire. Toutes les solutions supposent que les métadonnées sont disponibles en ligne ou fournies par le fournisseur de données.
Nous avons subséquemment proposé un modèle d'évaluation de la qualité des données des services de boîtes noires en utilisant ladite fraîcheur des données. Aussi, nous abordons les questions liées à l'absence de données probantes nécessaires pour cette évaluation au moyen du modèle. L'objectif de notre solution est de classer les services de données en fonction de leurs niveaux de qualité de données.

La fraîcheur des données est évaluée à l’aide de deux métriques d'actualité, y compris l'actualité des données et l'actualité de la base de données. L’actualité des données révèle à quel point les données recueillies sont à jour. L’actualité de la base de données révèle quant à elle à quel point la base de données du service est à jour.

Ensuite, nous avons proposé TUTOR, un protocole d’observabilité de qualité de données pour les services de données boîtes noires, permettant de recueillir les preuves nécessaires à l’aide de techniques d’échantillonnage pour le calcul des métriques d’actualité et par conséquent, le niveau de fraîcheur des données. Les services sont ainsi étiquetés avec un niveau de qualité des données à jour.

Notre protocole TUTOR se base sur 3 étapes primordiales, à savoir :

(1) Les connaissances sont construites grâce à l’historique des changements de données pour un service de données candidat, ce qui permet d’obtenir un aperçu de la mesure de l’actualité. Ces connaissances sont stockées dans une base de connaissances. L’échantillonnage constituant un mécanisme d’aide pour effectuer des inférences statistiques sur la qualité des données de la source de données, en particulier la fraîcheur des données. Par conséquent, afin d’obtenir et de constituer de bonnes observations, il est crucial de choisir les bonnes caractéristiques d’échantillonnage (informatives et utiles) pour TUTOR. Une caractéristique d’échantillonnage est une propriété mesurable individuelle. La collecte de toute l’information pour l’évaluation de la qualité des données peut s’avérer longue et coûteuse compte tenu de nombre de demandes d’échantillonnage nécessaires. Par conséquent, les données doivent être échantillonnées pour trouver un compromis entre la collecte d’un nombre suffisant d’échantillons représentatifs de la source de données en question pour la mesure des mesures de rapidité et la réduction des coûts induits par l’accès répété aux services de données et l’extraction d’échantillons de données. Ainsi, le choix de la technique et de la fréquence d’échantillonnage est crucial car il indique à quelle fréquence TUTOR accède à un service de données. Pour choisir la méthode d’échantillonnage adéquate, nous avons mené des expérimentations qui consistent à observer la variation en moyenne de l’actualité de données et le pouvoir de TUTOR à détecter les mises à jour des bases de données cibles pour des fréquences d’échantillonnage différentes. Les résultats de ces expérimentations nous ont permis de valider TUTOR ainsi que de montrer que l’échantillonnage avec une fréquence aléatoire est meilleur permettant d’obtenir un équilibre entre la collecte de connaissance

suffisante sur les sources de données et la réduction du coût lié à l'échantillonnage.

(2) Les observations effectuées sont périodiquement utilisées comme données d'entrée pour le processus d'observation de l'actualité de la base de données, qui évalue à quelle fréquence une base de données d'un service de données candidat est mise à jour au cours d'une période d'observation, en l'occurrence la durée de la dernière évaluation.

(3) Les observations accomplies au cours des étapes précédentes sont périodiquement utilisées comme données d'entrée pour le processus d'évaluation de la qualité des données. Ceci permet ainsi d'évaluer le niveau de fraîcheur possible des données de chaque service de données candidat.

2.2. Modèle d'évaluation de la fiabilité pour les services de flux de données

Dans un **second lieu**, nous avons proposé un modèle d'évaluation de la fiabilité pour les services de flux de données.

Les fournisseurs de services proposent des services de données dans des conditions de qualité de service hétérogènes pour répondre aux besoins des différents utilisateurs en matière d'évolutivité, de rapidité de réponse, de disponibilité, etc. La qualité de services (QoS) varie d'un fournisseur de services à l'autre, d'un même prestataire à plusieurs moments (par exemple, au jour le jour), voire de différentes applications selon le domaine d'application. Par conséquent, les services offrent différents niveaux de qualité de service qui peuvent fluctuer au fil du temps et, ainsi, ne permettent pas de se fier à la qualité convenue fournie dans leur contrat SLA. Le SLA c'est un contrat qui stipule les conditions d'utilisation d'un service ainsi que son niveau de QoS. De ce fait, la mesure et le suivi de la qualité de service contribuent à choisir un service en adéquation et répondant aux besoins des utilisateurs.

De plus, les services de données accèdent à diverses sources de données qui sont continuellement mises à jour par de nouvelles données à chaque fois que celles-ci évoluent de façon dynamique dans le monde réel. En effet, les services de données de flux mettent à jour leurs données avec des taux différents qui influencent la fraîcheur des données accédées par un service donné. Par exemple, étant donné un utilisateur dont le besoin porte sur des données saisies au cours des 60 dernières secondes, un service dont la base de données est mise à jour chaque 10s est plus susceptible de fournir des données plus récentes qu'un service dont la base de données est mise à jour chaque 60s. Par conséquent, il est nécessaire de fournir également une mesure permettant de déterminer dans quelle mesure les données consultées par un service donné sont à jour. À cette fin, lorsque nous sollicitons et invoquons un service de données à un fournisseur, nous tenons compte de deux facteurs. Tout d'abord, les expériences passées avec le service de données permettant d'obtenir un aperçu de sa performance. Deuxièmement, nous examinons la qualité des données fournies par le service de données en nous concentrant sur la fraîcheur des données.

Ainsi, notre modèle d'évaluation de la fiabilité pour les services de flux de données repose simultanément sur les aspects fonctionnels et non fonctionnels du service. Autrement dit, sur les aspects techniques du service et les aspects qualité des données fournies. A cette fon, une série d'étapes a été suivie en vue de définir ce modèle d'évaluation de la fiabilité : premièrement, la définition des métriques pour l'évaluation des performances des services, y compris la disponibilité, l'efficacité du temps et le taux de réussite des tâches. Deuxièmement, la définition d'une méthode alliant la qualité des données et la performance afin de calculer la fiabilité des services de données. Les services sont ainsi étiquetés avec un niveau de confiance à jour.

Notre modèle d'évaluation de la fiabilité est implémenté à l'aide d'une architecture DETECT (voir figure 2). Cette dernière se compose de trois modules principaux : (i) un module de mesure de la performance (PMM), (ii) un module de mesure de la qualité des données (DQMM), et (iii) un module de mesure de fiabilité (TMM). DETECT permet aux consommateurs de données de sélectionner les services de données les plus fiables en fonction de leurs besoins et de leurs préférences. Aussi, les consommateurs recherchent des services et DETECT renvoie une liste de services de données classés selon leur niveau de fiabilité. Par ailleurs, DETECT permet de mettre constamment à jour le niveau de confiance des services en surveillant leurs comportements, en calculant et en mettant à jour leur mesure de confiance, et en étiquetant les services avec cette valeur.

De ce fait, le *PMM* mesure la performance du service de données. Il exécute cette tâche en deux étapes : l'observation et l'évaluation. Au cours de l'étape de l'observation, PMM recueille les mesures de performance en continu à l'aide de l'API du service et stocke les mesures de performance dans une base de données

chronologique nommée « TSDB » (Time Series DataBase). Les séries chronologiques indiquent le temps de réponse enregistré des services en millisecondes, si le service était disponible et s'il avait terminé la requête avec succès. Pendant l'étape de l'évaluation, le PMM crée une base de niveau de performance. Tout d'abord, il recueille des mesures de rendement à partir de la TSDB effectuées dans un intervalle de temps précis. Deuxièmement, il calcule les indicateurs de performance. Troisièmement, le PMM évalue le niveau de performance du service correspondant et stocke la liste des services étiquetés avec leur plus récent niveau de performance évalué avec le temps d'évaluation dans une base de données.
Le *DQMM* quant à lui, observe et évalue la qualité des données consultées par le service de données correspondant en implémentant TUTOR.

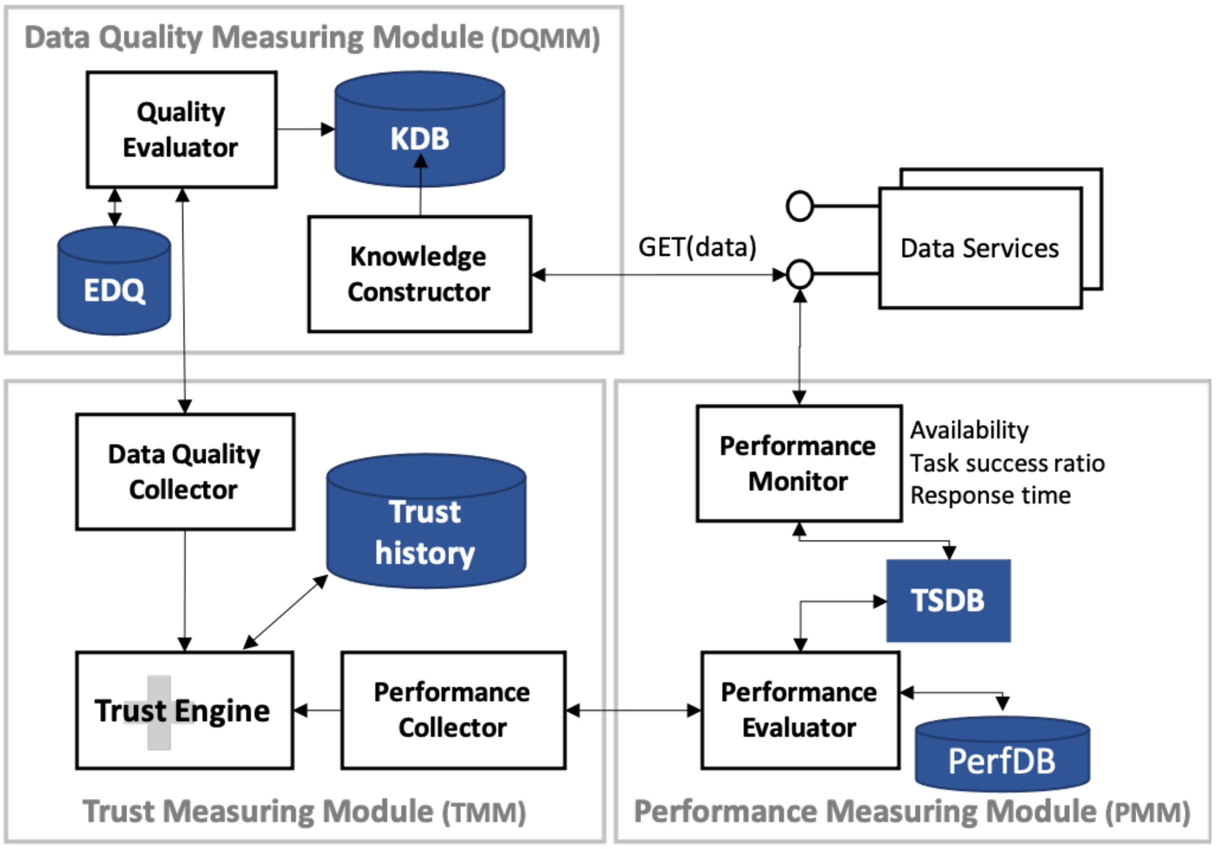


Figure 2 : l'architecture DETECT

Enfin, le *TMM* évalue le niveau de fiabilité des services de données boîtes noires en utilisant comme entrée les résultats de sortie des deux modules PMM et DQMM. Ce module fonctionne en deux étapes également fonctionnant selon un modèle pub-sub [1]: la phase *collection* qui collecte les niveaux de performance et de qualité des données, et l'étape *évaluation* qui évalue le niveau de confiance en consommant les résultats de l'étape de collecte.

2.3. Évaluation :

Enfin, nous avons implémenté ces deux propositions et les avons validées séparément en utilisant des services de données du domaine médicale portant sur l'apnée du sommeil dans le cadre du projet SUMMIT dans lequel s'inscrit la présente thèse, financé par la région Auvergne Rhône Alpes.

Les expérimentations du modèle de l'évaluation de la qualité de données nous ont permis de valider notre solution et de montrer son efficacité à classer 7 services selon leur niveau de qualité de données.

Les expérimentations du modèle d'évaluation de la fiabilité nous ont permis de valider notre solution à travers deux expérimentations. La première a été menée pour tester l'efficacité de notre solution dans le classement des services de données selon les préférences des utilisateurs (différentes pondérations sont attribuées à la performance et à la qualité des données). En effet, notre solution réussit pour chaque niveau d'importance accordé à la performance du service et à la qualité des données dans le classement de 7 services de données candidats. La deuxième expérience a été menée pour (1) tester l'efficacité de notre solution à satisfaire les exigences des utilisateurs en termes de qualité des données et de performance, tout en augmentant le nombre de demandes entrantes, et (2) démontrer la valeur ajoutée du facteur de la qualité des données au modèle d'évaluation de la fiabilité. En effet, lorsque le nombre de demandes entrantes est maintenu, notre solution réussit à maintenir la satisfaction des utilisateurs en termes de qualité des données et performance tout au long du temps avec respectivement une moyenne de 89% et 83%. De plus, le niveau de satisfaction des utilisateurs n'est pas affecté par l'augmentation du nombre de demandes entrantes.

Les contributions et les résultats décrits ci-dessus ont mis en évidence des questionnements liés à l'extensibilité (*scalability*) de notre approche. Ces limites sont dues principalement à l'absence actuelle de services de flux de données réels en lien avec le domaine médical et plus spécifiquement à l'apnée du sommeil sujet du projet SUMMIT et que nous pouvons utiliser pour

[1] Pub-sub : c'est un modèle qui permet à un émetteur (Publisher) de pousser, en temps réel, des messages au client qui s'est abonné (Subscriber)

expérimenter avec nos solutions. Ainsi, les services de données utilisés dans les diverses expérimentations sont déployés localement avec des ressources limitées. De surcroît, et pour les mêmes raisons, les données accessibles par ces services sont simulées, y compris la production et l’insertion de données. Ce choix pratique nous permet d’accéder à des données en temps réel et de les observer à la volée. Ces questionnements feront l'objet de nos travaux futurs, principalement autour de l'usage desdits services de données de flux réels.

Mots clés : Fiabilité, services de flux de données, performance, qualité de données.