https://doi.org/10.35662/unine-thesis-3068

Université de Neuchâtel
Faculté des sciences
Institut d'informatique

Using multiperspective observations to improve data quality in distributed systems

par

Panagiotis Gkikopoulos

Thèse

présentée à la Faculté des sciences
pour l'obtention du grade de Docteur ès sciences

Acceptée sur proposition du jury:

Dr. Valerio Schiavoni, Directeur de thèse
Université de Neuchâtel, Suisse

Prof. Peter Kropf, Co-Directeur de thèse
Université de Neuchâtel, Suisse

Dr.-Ing. habil. Josef Spillner, Co-Directeur de thèse
Zürcher Hochschule für Angewandte Wissenschaften, Suisse

Prof. Pascal Felber
Université de Neuchâtel, Suisse

Prof. Lydia Chen
Université de Neuchâtel, Suisse

Prof. Miguel Matos
IST, U. Lisboa & INESC-ID, Portugal

Prof. Rafael Tolosana-Calasanz
Universidad de Zaragoza, Espagne

Soutenue le 28 Septembre 2023

unine
Université de Neuchâtel
Faculté des sciences

Rue Emile-Argand 11
CH-2000 Neuchâtel
doctorat.sciences@unine.ch

IMPRIMATUR POUR THESE DE DOCTORAT

La Faculté des sciences de l'Université de Neuchâtel autorise l'impression de la présente thèse soutenue par

Monsieur Panagiotis GKIKOPOULOS

Titre :

"Using multiperspective observations to improve data quality in distributed systems"

sur le rapport des membres du jury composé comme suit :

- Dr Valerio Schiavoni, co-directeur de thèse, Université de Neuchâtel, Suisse
- Prof. émérite Peter Kropf, co-directeur de thèse, Université de Neuchâtel, Suisse
- Dr-Ing. habil. Josef Spillner, co-directeur de thèse, ZHAW, Suisse
- Prof. Pascal Felber, Université de Neuchâtel, Suisse
- Prof. Lydia Chen, Université de Neuchâtel, Suisse
- Prof. Miguel Matos, IST, Universidade de Lisboa, Portugal
- Prof. Rafael Tolosana-Calasanz, Universidade de Zaragoza, Espagne

Neuchâtel, le 19 octobre 2023

Le Doyen, Prof. R. Bshary

ACKNOWLEDGMENTS

Doing a PhD is an exciting and rewarding journey. We get to work at the cutting edge of our field, interact with leading scientists of our discipline, travel the world and present our work to our peers. All while striving to push the state-of-the-art that all-important inch forward.

It is also a very challenging one. One has to quickly learn to deal with rejection from reviewers, unpredictable workloads and the constant need to be flexible and available at all times. A collaborative PhD between institutions also brings its own set of challenges. Administrative misunderstandings, incompatible schedules, the strain of lots of remote communication and the always complicated matter of explaining my affiliation during conferences and other events. And of course, as with all my fellow PhD students who started their work during the pandemic, we were robbed of much of the joy of travelling to share our work with our colleagues abroad.

No road worth travelling is ever an easy one though, and I could never ask for better support than the one I had during my journey. I would so like to thank Valerio Schiavoni, for all of his guidance and feedback, and for his patience in helping me navigate the cross-institute administration of my studies. I would also like to thank Peter Kropf, for supporting and helping me even into his retirement. Last but not least, I would like to thank Josef Spillner, for being the first to believe in me, and for going above and beyond to fully realise his mentoring role during my PhD study.

I would also like to thank all of the collaborators, colleagues and friends I made along the way, the true reward of the journey, as they say. I wish I could list everyone, but I would especially like to thank all of the co-contributors of this work, in more-or-less chronological order, Oleksii Serhienko, Piyush Harsh, Alina Buzachis, Massimo Villari, Cristian Mateos, Alfredo Teyseyre, Pamela Delgado, Cristine Choirat, Oliver Cvetkovski, Mehmet Cihan Sakman, Francesco Martella and Cristina Abad Robalino.

Θα ήθελα επίσης να ευχαριστήσω την οικογένειά μου, τους γονείς μου, Σπύρο και Ξανθιάννα, που πάντα πιστεύουν σε εμένα και με στηρίζουν, ακόμα και όταν αποφάσισα να φύγω χιλιάδες χιλιόμετρα μακριά ακολουθόντας τα όνειρά μου. Τον αδερφό μου Γιάννη, που με το χιούμορ και τη διάθεσή του πάντα καταφέρνει να "διορθώσει" και τη δική μου.

Τέλος, περισσότερο από όλους, θα ήθελα να ευχαριστήσω τη Μαρία μου, που πάντα είναι κοντά μου, όσο δύσκολες κι αν είναι οι μέρες, και πάντα κάνει όλες τις δυσκολίες να εξαφανιστούν. Τίποτα από όλα αυτά δεν θα συνέβαινε χωρίς εσένα.

RÉSUMÉ

Les systèmes pilotés par les données deviennent rapidement un paradigme de premier plan, l'avènement de l'IA et des systèmes cyber-physiques intelligents devenant une caractéristique déterminante de l'époque moderne. On dit souvent que la qualité des données qui entrent dans ces systèmes est le principal déterminant du comportement et des décisions qu'ils produisent. Il est donc primordial de fournir de meilleures stratégies de gestion et d'amélioration de la qualité des données pour les systèmes autonomes.
Un autre attribut de l'infrastructure moderne basée sur les données est sa nature hautement distribuée. En effet, les déploiements de cloud, d'IoT et de continuum/fog sont presque omniprésents dans la pratique actuelle. Dans de nombreux cas, les systèmes pilotés par les données susmentionnés sont déployés sur ce type d'infrastructure en premier lieu. Nous voyons une opportunité de tirer parti de l'ubiquité des ressources de calcul distribuées pour ajouter une couche d'assurance qualité aux données.
Notre travail s'inspire de deux sources principales. D'une part, la nécessité de collecter de manière cohérente des données brutes de haute qualité et des informations et mesures dérivées, en présence d'erreurs, de problèmes et d'autres interférences. D'autre part, nous nous inspirons de la fusion de données, une méthodologie couramment utilisée pour combiner des données provenant de différentes sources afin d'obtenir des informations de meilleure qualité que la somme de leurs parties. Nous envisageons une généralisation de la fusion de données à tous les formats d'ensembles de données, en particulier lorsqu'elles sont obtenues en doublons redondants par des observateurs indépendants. Nous appelons ce type d'observation une "observation multiperspective".
Notre méthodologie de base consiste à concevoir, mettre en œuvre et évaluer ce concept d'observation multi-perspective. La première partie est un système d'observateurs indépendants, représentés comme des nœuds dans une architecture distribuée, des collaborateurs dans un projet crowdsourcé ou même simplement des capteurs matériels dans une configuration traditionnelle de fusion de capteurs.
Nous commençons par présenter la première partie de cette stratégie d'observation, l'acquisition de données. Nous présentons notre premier scénario motivant, qui consiste à suivre l'évolution de l'écosystème cloud-native dans un observatoire distribué "démocratique". Nous fournissons ensuite notre implémentation de cet observatoire et présentons son utilisation pour comprendre et améliorer le support matériel dans les images Docker. En outre, nous discutons de l'intégration de notre système d'acquisition de données avec des outils de reproductibilité et de preuve des données centrés sur la science des données. Ce travail nous permet également d'étudier les limites et les défis liés à l'obtention de données de la part d'observateurs indépendants. Nous utilisons nos résultats pour développer nos méthodologies dans la partie suivante de ce travail.
Nous présentons ensuite la solution que nous proposons pour résoudre les problèmes de qualité des données découverts : Consensus centré sur les données (DCC). En utilisant notre système d'acquisition de données et les données que nous avons obtenues, nous développons une architecture de système pour fusionner les observations en une vue commune de la vérité, sur laquelle tous les observateurs sont d'accord. Nous étudions ensuite les algorithmes que nous pouvons utiliser pour y parvenir, ainsi que les implications de notre système en termes de performances.

Enfin, nous nous concentrons sur les algorithmes eux-mêmes et présentons notre propre contribution à l'espace des électeurs définis par logiciel, l'algorithme de vote AVOC, et VDX, une spécification générique pour décrire les électeurs définis par logiciel. Nous évaluons notre contribution à la fois par rapport et en conjonction avec l'état de l'art dans un exemple de fusion de capteurs pour montrer que les algorithmes de vote peuvent en effet augmenter à la fois la qualité des résultats et la performance lorsqu'ils sont correctement combinés avec des approches de fusion de données standard.
Dans l'ensemble, cette thèse présente la stratégie de l'observation multi-perspective dans une approche de bout en bout, de l'acquisition des données à la réconciliation des conflits et à la détermination des gains de qualité des données. Nous montrons où cette méthodologie est applicable et fournissons une mise en œuvre du modèle ainsi qu'une évaluation de ses performances et de ses limites.

Mots-clés: Fusion de Données, Systèmes Distribués, Science des Données, Électeurs Logiciels

ABSTRACT

Data-driven systems are quickly becoming a prominent paradigm, with the advent of AI and smart cyber-physical systems becoming a defining characteristic of the modern day. It is often stated that the quality of data going into such systems is the primary determinant of the behaviour and decisions they produce. It is thus paramount to provide better strategies for managing and improving data quality for autonomic systems. Another attribute of modern data-driven infrastructure is its highly distributed nature. Indeed cloud, IoT and continuum/fog deployments are almost ubiquitous in current practice. In many cases, the above-mentioned data-driven systems are deployed to such infrastructure in the first place. We see an opportunity to leverage the ubiquity of distributed compute resources to add a layer of quality assurance to data. Our work is inspired by two main sources. On one hand, the need to consistently collect high-quality raw data and derived insights and metrics, in the presence of faults, issues and other interference. On the other hand, we draw inspiration from data fusion, a methodology commonly used to combine data from different sources to obtain insights of higher quality than the sum of its parts. We envision a generalisation of data fusion to all formats of datasets, particularly when obtained in redundant duplicates from independent observers. We call such an observation, a 'multiperspective observation'. Our core methodology is to design, implement and evaluate this concept of multiperspective observation. The first part is a system of independent observers, represented as nodes in a distributed architecture, collaborators in a crowdsourced project or even just hardware sensors in a traditional sensor-fusion setup. We begin by presenting the first part of this observation strategy, data acquisition. We show our first motivating scenario, tracking the evolution of the cloud-native ecosystem in a 'democratic' distributed observatory. We then provide our implementation of this observatory and present its use in understanding and improving hardware support in Docker images. Further, we discuss the integration of our data acquisition system with data science-centric reproducibility and data provenance tooling. This work also serves to help us study the limitations and challenges of obtaining data from independent observers. We use our findings to develop our methodologies for the next part of this work. Then, we introduce our proposed solution to the discovered issues in data quality: Data-Centric Consensus (DCC). Using our data acquisition system and the data we obtained, we develop a system architecture to merge the observations into a common view of the truth, that all observers agree to. We then investigate the algorithms we can use to achieve this, and the performance implications of our system. Finally, we focus on the algorithms themselves, and present our own contribution to the space of software-defined voters, the AVOC voting algorithm, and VDX, a generic specification for describing software-defined voters. We evaluate our contribution both against and in conjunction with the state-of-the-art in a sensor fusion example to show that voting algorithms can indeed augment both output quality and performance when correctly combined with standard data fusion approaches. All in all, this thesis presents the strategy of multiperspective observation in an end-to-end approach, from acquiring data, to reconciling conflicts to determining the gains in data quality. We show where this methodology is applicable and provide model implementation and an evaluation of both its performance and limitations.

Keywords: Data Fusion, Distributed Systems, Data Science, Software Voters

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

Distributed architectures are widely used in systems with a certain criticality concerning life, environment or business continuity and with data-intensive or data-driven workloads. The complexity of the data as well as the overall system is ever-increasing. However, with the rise of automated decision-making and AI-driven systems, the quality of the input data utilized is gaining increased importance, as exemplified by the data-centric AI movement[1]. It is often claimed that the output of such systems is only as good as the input. As such, the importance of high-quality input data is increasing, and handling key deficiencies in data quality is a key factor in delivering high-quality decisions.

Deficiencies in data quality, such as missing or incomplete values, data corruption and various types of biases threaten the integrity of the underlying systems and reduce the reliability of the decisions made. Missing data can occur from incomplete execution or errors in data acquisition pipelines, as well as misconfiguration and incomplete instrumentation. Similarly, data can be corrupted by runtime errors or misconfiguration, such as in the case of scraping a public data source that may change its interface. In the case of sensor data, interference effects, such as sunlight glare on a visual sensor, can also skew or corrupt data. Lastly, biases can be introduced in data that is collected by a single authority, even if that authority utilizes a distributed architecture. However, distributed architectures also present an opportunity for a solution to these concerns. An observation system that is designed around independent observations from multiple perspectives can be used to mitigate the negative impact of these deficiencies.

This work investigates the potential and applications of what we will refer to as *multiperspective observations*. We define this as an umbrella term for observations of the same subject from different points of view in the context of distributed systems, either by independent authorities, independent instrumentation (e. g., redundant sensor readings) or different observation means (e. g., acquisition methodology or preprocessing algorithms). These independent observers can be homogeneous (i. e., using the same hardware or software) and thus ensuring easier reproducibility at the expense of propagating implementation errors (such as software bugs and manufacturing deficiencies). They can also be heterogeneous, which would make comparisons more difficult, as data formats and timing would have to be reconciled, but provide the benefit of not replicating each other's errors, as they follow different solutions to the same problem. This distinction can be thought of as the difference between N-Module Redundancy and N-Variant Programming, but generalized so that both can apply to either software or hardware. We identify a multiperspective system as capable of performing two key tasks: The first task is reproducible data collection from independent

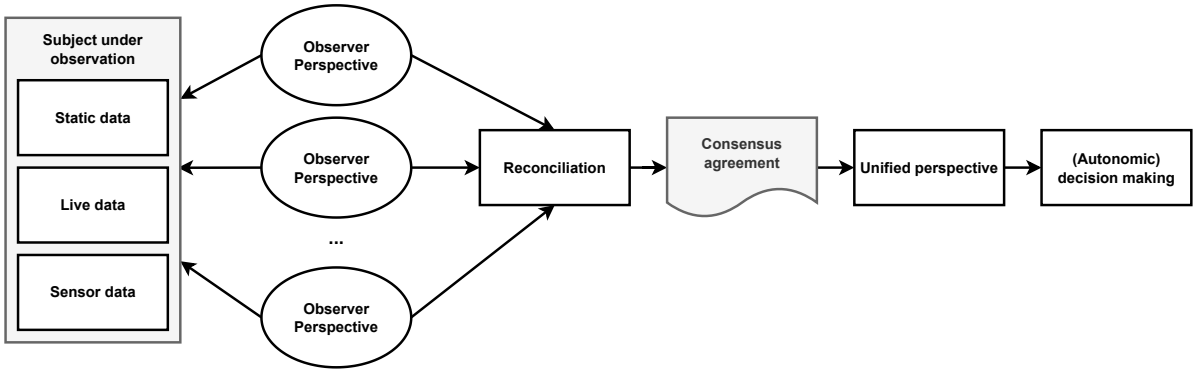

Figure 1.1: High-level overview of multiperspective observations

perspectives, with the usage of portable software or redundant hardware. The second task is the capability to reconcile, merge or otherwise aggregate the data from these observations into a dataset, stream, or other suitable output representation. The second task is key to the improvement of data quality, as it allows the construction of a dataset that is more complete and less biased than the individual observations. In addition, the reproducible data acquisition means that the input data production process is more transparent and wrong decisions can be traced back more easily to misconfiguration, broken data sources etc. A high-level overview of the concept is shown in Figure 1.1.

Multiperspective observations can be applied to several domains. In particular, we have identified reproducible science[2, 3], crowdsourced or collaborative data collection projects, independent quality review of public artefacts, sensor fusion systems, particularly in the context of smart cities[4] and other large cyber-physical systems[5] and quality control of AI input datasets, particularly those with crowdsourced labels[6].

Several of these cases have been investigated as part of this work, with related software and datasets available in addition to the related publications. The applied sciences setting of this doctoral work has enabled us to work closely with partners in the industry across a variety of projects, ensuring the broader applicability of our results. Namely, we have investigated the reproducibility scenario, collaborative data collection and sensor fusion in IoT-centric smart retail and building settings with redundant sensors. Concretely, this work has focused on investigating the feasibility and applicability of this methodology to different domains as well as on the design and experimental evaluation of systems that apply the principles discussed. A particular focus of this work has been the application of containerized software as a vehicle for delivering software to the different observers, as well as the development of voting-based methods for merging the observations. The intent is to evaluate these tools and methodologies in terms of their effect on data quality and their feasibility in practice.

We can summarize the goals of this work with the following research questions:

- RQ 1: How can multiperspective observation be utilized to obtain multiple perspectives on data?
- RQ 2: How can crowdsourced or decentrally provided insights be utilized in practice?
- RQ 3: Can multiperspective data be aggregated/merged to resolve conflicts, restore data loss and/or improve the reliability of a dataset?
- RQ 4: Can multiperspective data reconciliation be adapted to the needs of real-time systems?

1.2 Contributions

The contributions made over the course of this work are on a spectrum from theory to applied science, being driven by the particular domains of application. They can be summarized as follows:

- Distributed metrics observation. A core part of multiperspective observation, is that the means of deriving observations need to be easily available to all observers. In the case of hardware, that means sensors capable of producing comparable measurements. In the case of software, that means portable software and reusable workflows. To this end, DiMOS (Distributed Metrics Observation System) was developed and evaluated. DiMOS (formerly called MAO, Microservice Artefact Observatory), is a tool used to wrap the execution of reproducible software and manage their input and outputs in collaboration with other users. It allows the execution of workflows implemented as containers with specific branches of repositories mounted to them as data volumes. This allows each collaborator to manage their own copy of a dataset easily. The tool also includes a voting system that uses the VDX (see below) specification to merge and reconcile data from the aforementioned branches and aggregate the combined result from all collaborators to a 'ground-truth' branch. This enables operators to make independent observations on the same facts, thus improving the quality of the reconciled consensus output.
- An integrated data pipeline combining MAO and Renku[7] is designed and implemented, as a study of integrating multiple observation and reconciliation mechanisms. Data generated using one approach is placed in repositories managed by the other framework. This allows the results of a collaborative data collection workflow to serve as the input to a reproducible data science workflow. This method allows the extension of our method to include an end-to-end data science workflow.
- A study of hardware support in public Docker images is conducted using the data obtained from MAO. Several repositories with public Docker images that supported hardware features like Intel SGX and Nvidia CUDA were analysed to generate insights on the state of support for heterogeneous hardware in the Docker ecosystem. The insights were used in the development of HDocker. This study is also a proof of concept for the ability of multiperspective observations to drive decision-making, in this case for the development of tooling.
- HDocker: Hardware Docker, a tool that was developed using insights on hardware-compatible docker images obtained from the first deployment of DiMOS. The tool informs the users on what hardware components (if any) a container image requires and provides them with advice on how to configure pass-through from the host to the container. This tool was developed to demonstrate and evaluate the utilization of our metrics to produce actionable advice.
- DCC: Data-Centric Consensus, a method of reconciling observation data from different perspectives using voting algorithms and aggregation. DCC is the second key component of the multiperspective observation methodology. It allows granular comparison, aggregation and verification of dataset views from the different observers, that are then merged using customizable rules to produce the final agreement dataset. DCC is built into the DiMOS tool and implemented using our VDX system.

Table 1.1: Publications and artefacts related to this dissertation

Chapter	Keyword	Section	Publication	Code & Data	Venue
2	MAO-Renku	2.3	[8]	[9]	SoftwareX'22
	Docker Hardware	2.2	[10]	[11, 12]	DAIS'21
3	DCC	-	[13]	[14–16]	TPDS'22
4	VDX	-	[17]	[18]	Middleware'22

- VDX: Voting Definition eXtended, a generic specification for designing software-defined voters. The system allows the usage of state-of-the-art voting algorithms as well as our own AVOC voting strategy. Voting is used as the key strategy for merging diverging views in granular data. The data-driven voting strategy employed by DCC requires each field in the dataset to be verified separately, thus allowing a granular rule-based verification of a large dataset, that would normally be time-consuming and costly to verify manually. VDX is specialized in numeric voting methods, used to reconcile numeric values using voting algorithms, though generic majority voting on arbitrary data types is also supported.
- AVOC: Accurate VOting with Clustering, is an extension of the state-of-the-art history-based voting algorithm by adding a clustering-based bootstrapping procedure to increase the accuracy early in the operation of the system (before the historical record is reliable enough) or during transient errors (when the historical record becomes unreliable). This was developed based on the observation that history-based voting systems require a lengthy history to be reliable, thus early in a system's life cycle, a fallback mechanism is needed to increase accuracy beyond what is possible in a history-based weighted average algorithm, whose accuracy falls back to that of an unweighted average if the weights are not yet tuned. AVOC replaces this standard fallback with the use of a clustering-based voting mechanism, leveraging the outlier detection capabilities of clustering algorithms to improve accuracy in these situations.

These contributions were published in conference proceedings and journal papers, co-authored by myself and my supervisors, Josef Spillner and Valerio Schiavoni. Other international collaborators contributed to individual publications over the course of this work. A summary of publications completed over the course of this work, along with venues, published datasets and code artefacts where applicable, is presented in Table 1.1. A detailed list of publications authored over the duration of this work is available in the Appendix. The text and findings in this thesis qualify my own intellectual contributions to these works.

1.3 Background

Managing data in the context of a distributed system involves many challenges, but its broad applicability makes it a topic of significant interest, with several approaches to several aspects of the issue and at different levels of maturity[19]. Individual aspects of the state of the art and their relation to this work are discussed within the individual chapters, but it is important to establish key concepts and the general positioning of this work within this topic area. This section aims to briefly present some of the key challenges of distributed data management, well-established approaches to these challenges and the position of this work within this context. In addition, key aspects of data reconciliation and their relationship to the application domains we mentioned above are also briefly presented here.

1.3.1 Challenges in data quality

The reliability and usefulness of data are subject to several quality issues, with some of them becoming more disruptive if multiple independent observations are involved. Data gaps can range from single fields missing due to incompatible interfaces to the entire execution of a scraping job being skipped due to errors or a power outage in the infrastructure. These gaps, in turn, can create incomplete views of the subject under observation, which can present increased biases in the behaviour of the system that uses the data. Another potential issue is data corruption, commonly caused during the transfer or transformation of data from one format to another, which can have the same effect.

Bias in the data and subsequently on the system behaviour is a very pressing problem in the field of AI[20] and an incomplete view of the subject can lead to such biases. This is a strong argument for incorporating multiple perspectives and collaborative data acquisition, which is why crowdsourcing (especially in the context of labels)[6] is a common practice of data-centric AI.

Lastly, data can be manipulated by malicious actors to introduce bias beneficial to them or to omit data that could be detrimental to their purposes. This work is not concretely focused on the security features needed to prevent such attacks, but methods to mitigate the impact of falsified data will be discussed in the context of data reconciliation.

To better understand the impact of data deficiencies, it is important to examine them in the context of the different indicators of data quality, as defined by the ISO 25012 standard[21]. These indicators are:

- Accuracy: the degree to which the data is free of errors and omissions.
- Completeness: the degree to which the data is free of gaps.
- Consistency: the degree to which the data is free of contradictions.
- Timeliness: the degree to which the data is free of delays.
- Uniqueness: the degree to which the data is free of duplicates.
- Validity: the degree to which the data is free of falsifications.

In the context of this work, we are primarily concerned with the first three indicators, as they are the most relevant to the data acquisition and reconciliation processes. In particular, our methodology to compare and combine the perspectives of multiple observers that provide complementary or redundant data can increase the accuracy of data by resolving errors with voting and data fusion algorithms. It can increase completeness by providing redundancy, where if an observer fails the other observers will provide data to fill the gap. Consistency (in our context, which is different from database consistency that relates to transactions) is achieved by resolving conflicts between observers, with the use of voting algorithms. The last three indicators are more relevant to the data storage and retrieval processes, which are not the focus of this work. Timeliness and uniqueness are not attributes where multiperspective observation can aid, and though validity can be verified in the conflict resolution process, in practice malicious skewing of observations or attacks that affect the majority of observers are not accounted for in our techniques, as they focus entirely on the data.

1.3.2 Data management in distributed systems

Data-intensive distributed systems are increasingly common with the advent of the big data[22] and AI fields. Several aspects of our desired data management strategy can be seen

Table 1.2: Data management solutions in distributed systems

System	Data distribution	Fault tolerance	Multiple views	Merging
Distributed database	✓	✓	✗	✗
Blockchain	✓	✓	✗	✗
BitTorrent	✓	✓	✗	✗
Distributed version control	✓	✗	✓	partial

in the state of the art, though as we will discuss, the intent and thus the capabilities do not match our requirements to the point that we could adapt existing systems or architectures.

Replicated data can be seen in distributed databases[23] and in particular, federated databases[24]. However, the intent of such systems is to provide a single, unified view of the data to the users, which is not the case in our scenario. Still, such a system could provide a usable back-end for our method, but that is purely an engineering consideration.

Collaborative data management naturally invokes the idea of peer-to-peer systems, such as BitTorrent[25], blockchains[26] (and in particular, consortium blockchains[27]) and distributed version control systems[28]. With regards to BitTorrent, once again the focus is on a unified version of the data being available to peers. Blockchain systems are especially interesting due to their use of consensus algorithms, however in this case, the consensus is not a granular data-centric procedure but rather a binary security feature, not able to provide nuances on data values. (Blocks are either accepted or not accepted into the chain.) Nevertheless, a hypothetical data-centric blockchain could be explored, though this was not the direction followed in this work. Still, our early MAO platform prototype was built on a peer-to-peer architecture, as described in Chapter 3.

Distributed version control systems such as git fit our criteria a lot closer, due to the support for multiple views of the files being tracked and the built-in tools for merging and branching. This was a close enough fit for us to choose such systems as our backend for both the DCC experiment and the subsequent DiMOS tool. However, it should be noted that this is not a one-size-fits-all solution. Distributed version control systems are not meant for the storage of large files and datasets, but once again, as this is an implementation-specific restriction, it was still a viable option for us. What is then concretely missing from version control systems as a solution to our needs is the ability to merge multiple versions of the data at once, and not from the perspective of files being identical, but rather with a granular rule-based reconciliation strategy for any conflicts. This is in broad terms what DCC aims to achieve and what we demonstrate with the DiMOS prototype. A summary of the key features of distributed data management systems is presented in Table 1.2.

1.3.3 Workflow and scheduling systems

Workflow or pipeline execution is a common component of data management systems in data science and AI. Several systems have the capability of executing a pre-defined workflow, such as Airflow[29], Dagster[30], Kubeflow[31] and other systems that can execute tasks defined as Directed Acyclic Graphs (DAGs). For the purposes of this work, we investigate integrating the workflow component in two ways. First, we demonstrate the inclusion of a workflow-based

Table 1.3: State-of-the-art voting algorithms and their status in our VDX specification

Algorithm class	**Multi-dimensional**	**Weighted**	**VDX**
Majority based	✗	✗	✓
Weighted averaging	✗	✓	✓
Tensor voting[33]	✓	✗	✗
Genetic voting[34]	✓	✓	partial
Clustering[35]	✓	✓	partial
Maximum likelihood[36]	✗	✓	partial

system, in our case, the SDSC's Renku platform and how it integrated with our method. This integration is not specific to the software, but rather an investigation of how our methodology integrates with the well-established practices of data science. However, for the majority of the work, we focus more on the processing of data after its acquisition, and for that purpose we assume the workflow is encapsulated within the portable software solution (in our case the containers managed by the DiMOS tool).

Early in this work, the focus was also on scheduled execution of the data acquisition workflows. Systems based on cron were used in early prototypes, and MAO, the precursor to DiMOS, featured a custom cron-like scheduler to trigger the execution of the tasks. Ultimately, this was not pursued further, as the focus shifted to the data reconciliation and verification aspects of the work. The existing solutions were deemed sufficient for us to use, and building our prototypes around them introduced complexity that was not necessary for the scope of this work.

1.3.4 Reconciling and verifying distributed data

The core focus of this work is on improving the quality of data, in the presence of multiple independent perspectives on the subject. While the motivation and methodology are novel, the verification and reconciliation aspect extends several existing approaches from the state of the art. One of the main methods of data reconciliation discussed in this work is using voting algorithms to decide the optimal values to retain within the dataset. In particular, we focus on numeric voting algorithms, as the imprecise nature of numeric data (e. g., in the results of an experiment) made voting a greater challenge. A detailed investigation of numeric voting algorithms, as well as our addition to the field, AVOC, is presented in Chapter 4.

Voting however, is not limited to numeric values. Majority voting is commonly used to compare and verify data collected by crowdsourcing[32], such as labels from AI datasets[6]. As majority voting is a powerful tool in voting for the correct value in arbitrary data that cannot be compared numerically, it was also incorporated into our generic voting specification, VDX, which we detail in Chapter 4. Other classes of voting algorithms are present in the state of the art, such as tensor voting[33] and genetic voting[34], however, these did not fit our topic domains and were not investigated in the context of VDX. In particular, our scenarios did not include higher dimensional data that would benefit from tensor voting techniques, and the concept of mutating and combining outputs in genetic voting was not a good fit for the way we implemented our generic voting, where each individual value is treated as one voting round. An overview of the general classes of voting algorithms, their high-level features and their inclusion or not in VDX are presented in Table 1.3.

We list genetic and Maximum Likelihood voting as partially included, as VDX includes their voting mechanisms (weighted majority voting and weighted average respectively), but

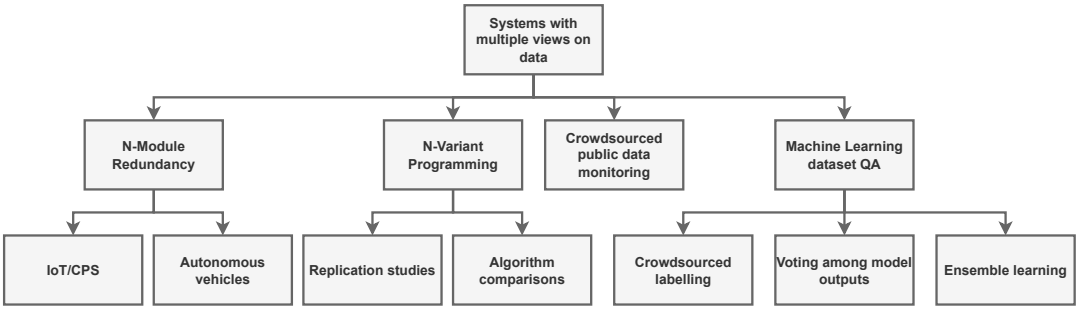

Figure 1.2: Overview of application domains

does not include in the specification the method the vote weights are derived, meaning they would need to be computed separately and passed to VDX. Clustering is also partially included, as VDX includes a one-dimensional clustering algorithm, but does not include the multi-dimensional clustering algorithms that are present in the state-of-the-art. Our reasoning and approach to multi-dimensional data is detailed in Chapter 4.

Concepts used in voting systems are also present in consensus algorithms. It is important to highlight that in our context, consensus is not strictly speaking a security feature (though it can mitigate some malicious data modifications), and as such is mostly unrelated to the consensus protocols as present in blockchain systems. However, the key aspects of fault tolerance, byzantine agreement[37] and leader elections (as seen in the Paxos[38] and Raft[39] algorithms) are relevant and are discussed in greater detail in Chapter 3, where we detail the considerations involved in the DCC methodology.

Finally, the process of data cleansing[40] is a key component in data science, and our voting-based verification is a form of data cleansing in the presence of data redundancy. With regards to data cleansing (or data cleaning), several other methods are combined into our work in various ways, such as Kalman Filtering, outlier detection and clustering. These are discussed further in Chapters 3 and 4.

1.3.5 Application domains

Multiperspective data verification is a term we use to describe a generalisation of multi-view data fusion[41] that includes combining any form of data, from sensor reading to large datasets. It is already employed in several domains. It is encountered in data crowdsourcing, such as in the context of AI, where majority voting is often employed to perform quality control on data labels[6]. Another area of AI where voting is encountered is ensemble learning, where voting techniques are used among the classifiers used[42]. Furthermore, voting can also be applied in scenarios where multiple models are predicting the same output and voting is employed, weighed by their confidence interval. In addition, the recent effort for more reproducible science depends on results being independently verifiable[3]. Though a great deal of work has been performed to aid the reproducible execution of research artefacts, the verification that results match is still commonly performed manually or using custom purpose-built scripts. We identify this as a gap that can benefit greatly from automated solutions.

There are scenarios where observations may be hindered by abnormal conditions. One such example is firewall restrictions preventing the querying of APIs or the download of datasets from certain institutions or locations. Another might be hardware or network failures in the middle of the experiment or during one of many scheduled runs of a scraping tool. If experiments were to be done once, such issues may have been mistaken for the norm and

there would be no way of recovering data lost, while a reproduction across multiple observers would allow them to be singled out as outliers and the redundancy allows gaps and corrupted data to be recovered. We tackle the domain of reproducibility and decentralized acquisition of batch datasets in Chapter 2.

Another angle for multiperspective data is that collected by redundant hardware or sensor fusion systems. This is already a mature field for several applications, such as avionics[43], where redundant sensors are typically employed alongside hardware voters. However, with sensor fusion gaining ground in consumer cyberphysical systems, such as in the context of autonomous vehicles[44], the ability to standardize and develop tools for data-centric voting is becoming relevant again. We have studies applying our VDX specification in this context in IoT-based use cases, which we present in Chapter 4. An overview of the identified application domains is presented in Figure 1.2.

1.3.6 System assumptions

In this work, multiperspective observation is defined as a data-centric methodology, and does not regard the system pipeline as a whole, but rather its resulting data. As such, a set of assumptions must be made for the rest of the system model.

As the input of the verification step is the output of the data acquisition, and the verification system has no control over the acquisition, the input of the data acquisition is assumed to be the same for all observers. If that is not the case, then the output would not be comparable, and any conflict resolution would not have value. As such, considerations needed to ensure the alignment of input data among observers (such as timing) need to be addressed by the designer of the acquisition workflow.

Security is not a concern of the verification step. As such, if the data acquisition step is compromised on one or more observers, the system has no way of detecting it. As a caveat, due to the reliance on majority voting-derived techniques for the verification step, the system can detect and filter out falsified data from a minority of nodes. This is however a side effect of voting, and the system has no way of differentiating falsified data from a malicious node versus erroneous data from a faulty one.

Finally, the executor of the verification (the arbiter node) is assumed to be trustworthy. Considerations regarding this assumption and ways to address it are further discussed in Chapter 3.

1.4 Outline

This thesis is organised into five chapters, with this introduction being the first one. The remaining chapters are organised as follows:

Chapter 2 presents the data acquisition aspect of multiperspective observation. It introduces the distributed metrics observation methodology, and presents the MAO design as a model implementation to verify the method in practice. Our design is presented in conjunction with the Renku data science platform developed by the Swiss Data Science Center, as an integration effort between the two was investigated. The chapter continues with the application of our data acquisition methodology to a study of heterogeneous hardware support in publicly available Docker images. This study employed an iteration of the DiMOS tool in a prototype collaborative data collection context. The results of the study were used to develop HDocker, a tool that provides advice on how to configure hardware pass-through

for Docker containers, as a way to demonstrate the end-to-end process of deriving actionable metrics from our process.

Chapter 3 focuses on the data aggregation and reconciliation aspects of this work. It introduces the challenges that arise in data-intensive autonomic systems and presents a deep dive into the considerations of developing a consensus system that can tackle the issues present. An overview of the requirements and pre-processing needed to enable granular data-centric consensus is described and process of DCC itself is presented, along with options considered and challenges faced. The method is evaluated by building a prototype implementation of DCC and applying it to DiMOS datasets as well as simulated data. Both performance and data quality aspects are evaluated to demonstrate the capabilities of the strategy.

Chapter 4 introduces VDX, a generic specification for designing software-defined voters. VDX is a component of the core DCC strategy of data reconciliation, a way to define the rules needed to merge diverging numerical values within an observation. It also presents AVOC, an algorithm for numeric voting that extends the current state-of-the-art software voters for numeric data. VDX and AVOC are then evaluated using a set of IoT-based testbeds, demonstrating a different view of the DCC strategy, then applied to scenarios needing real-time reconciliation of data, rather than the batch processing of entire datasets by DiMOS.

Finally, Chapter 5 is the conclusion to this work, where the contributions are summarized, the answers to the research questions posed are briefly revisited and perspectives for future work are presented.

Chapter 2

Multiperspective Data Acquisition

The first core aspect in a multiperspective-based data-driven system, is the acquisition of data. In such a system, we must be able to collect data from multiple independent sources. Those sources may be nodes in a cluster/cloud or peer-to-peer system, collaborators in a data science project, or redundant/overlapping sensor setups. The observers could belong to one or multiple authorities, provided their acquisition of data operates independently. The reconciled data could then range from logs, metrics and traces to sensor readings. Our observation concept, as presented in our introduction, can be further expanded as shown in Figure 2.1.

Our multiperspective observation system consists of N independent observers, who observe a common target. The method or tool used for observation may vary, provided that the outputs can be described in a common format and be directly comparable. In our use cases, the observation tools and methods (software or hardware) were shared among observers, and the reproducibility of observations was guaranteed either through the use of portable software in a reproducible environment, as described in this chapter, or with the use of identical hardware, as demonstrated in Chapter 4. Observations can be synchronized or not, depending on the needs of the application. The resulting observations are then subjected to a consensus mechanism to resolve conflicts among observers. The result of this mechanism is a single observation, representing the collective, agreed-upon view of the truth among observers. This observation is the closest approximation of the ground truth that the observers can create, as in real conditions, the ground truth cannot be known in advance.

In other words, given an observation subject S and N observers O, the observation is simply:

$$S_{\text{observed}} = \text{Consensus}(O_1(T), O_2(T), \ldots, O_n(T))$$

Where Consensus can be any from a number of mechanisms, further discussed in Chapters 3 and 4.

This chapter focuses on the first part of this process, the acquisition of data from independent sources. We will motivate our example use-case, the Microservice Artefact Observatory (MAO), and present the use of this framework to study the support of heterogeneous hardware in Docker images. Afterwards, we present our work on integrating MAO with the Renku platform, and how this integration can be used to create a reproducible environment for data acquisition.

It is important to define, that in the context of this work, when we mention reproducible acquisition, we mean that the means of acquisition is reproducible. In other words, the software components need to be identical and identically configured, and the workflow followed needs to be the same. In such case, if the input is also identical among observers, then the

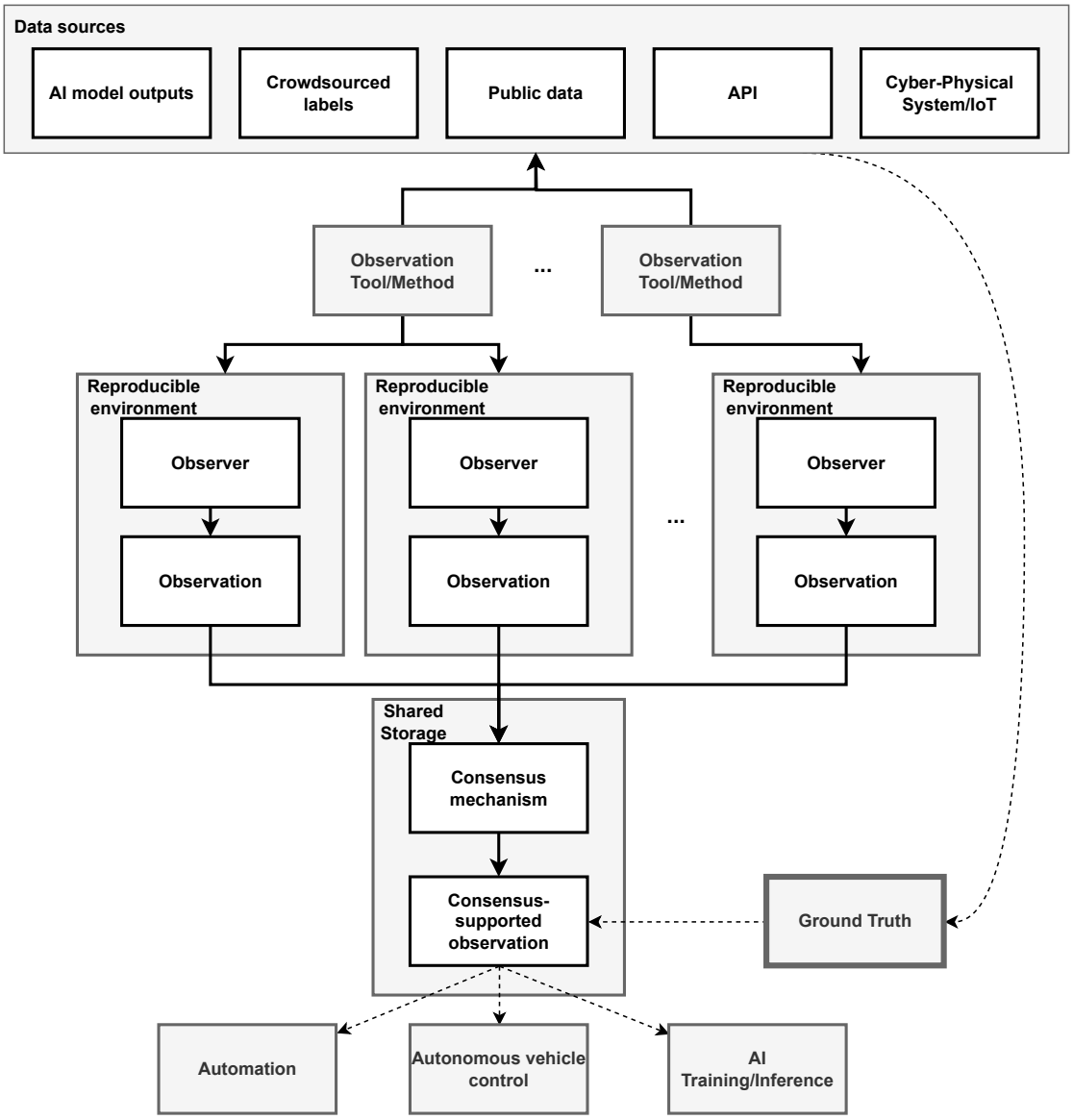

Figure 2.1: The conceptual architecture of a multiperspective observation system. A common target is observed. Tools may vary among observers. In the case of shared tools, using a reproducible execution/deployment environment ensures the reproducibility of the observation. The resulting observations are then collected in a shared storage, and subjected to a consensus mechanism to produce a single observation. That observation is our best approximation of the ground truth, which is not known in real situations, and can be used to drive automated systems or as input data for AI.

output is expected to be identical, and thus any conflicts are what our system is meant to resolve, using the methodologies presented in later chapters. This is a distinction from the typical definition of reproducibility[45], in which the output should already be identical (often within a specific margin of error) for the reproduction of a workflow to be considered successful. With this in mind, our system as discussed further, serves to resolve imperfections in the reproduction of a data acquisition workflow.

These works helped us observe the behaviors of independent observation tools in practice, and to identify the issues and challenges in producing a data-centric consensus mechanism. Briefly, these issues can be categorized as individual and distributed issues,

to reflect on data issues affecting one observer, or issues only apparent when obtaining an agreement among observers. Of particular interest to us are conflicts between observers. This brief categorization is illustrated in Figure 2.2 These issues and how we address them will be discussed in Chapters 3 and 4.

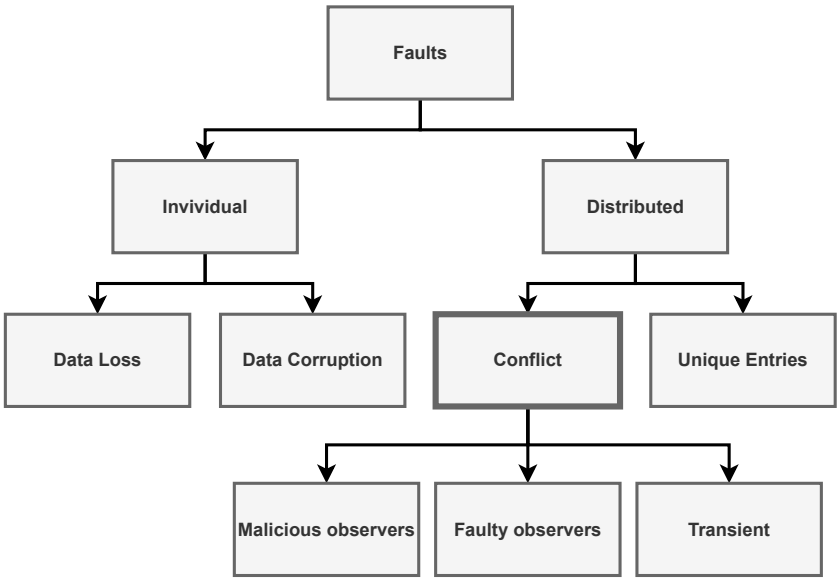

Figure 2.2: Individual and distributed issues in multiperspective observation.

2.1 The Microservice Artefact Observatory

2.1.1 Motivation

Microservices are becoming a prevalent model for modern software development as technology is headed more toward cloud-native systems. As is usually the case with new developments, new ways of implementing and exploiting the technology are now being released at a rapid pace. One area of particular interest is the emerging marketplaces for microservice software artefacts. In this context the term artefact entails any pre-made service or even set of services made publicly available by its developer as a package to be cloned and deployed through a marketplace. More and more such packages are made available from multiple vendors, with leading examples being Amazon's AWS, Kubernetes and Docker. However, as with any such new trending technology, the early steps of development are usually somewhat chaotic. Types of ready made deployment blocks like these are appearing fast and sometimes even failing before receiving traction and even for established marketplaces quality control is sometimes problematic or impossible. There is currently little effort to systematically monitor these hubs and marketplaces and their contents. However, this endeavour is scaling up in requirements fast. Where there used to be a handful of vendors and a few hundred artefacts to check, there are now several thousand in many cases.

For this reason the MAO informal consortium was envisioned, as a scientific community effort to monitor and document these artefact marketplaces, to provide insight on them through a combination of metadata checks, code quality control and dynamic testing. MAO was envisioned as a distributed observatory, with members choosing which monitoring projects and experiments to join and the ability to pool and compare data to improve the output quality. This concept is visualized in Figure 2.3

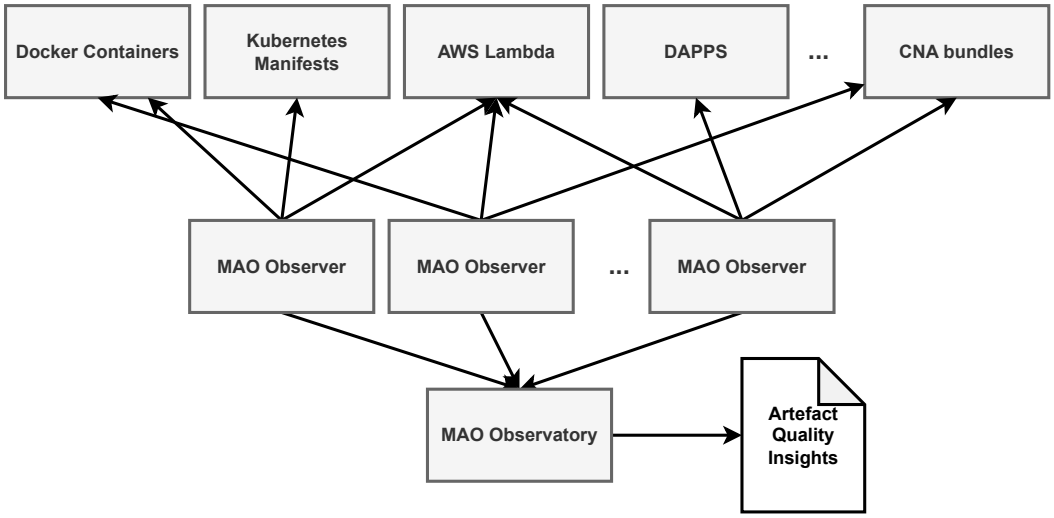

Figure 2.3: MAO: A distributed observatory for monitoring the quality of microservice artefact ecosystems. Each member can voluntarily join ongoing experiments and data is pooled to create insights.

2.1.2 Background

Cloud applications based on novel microservice architectures are drawing ever closer to becoming the norm in many sectors of modern software engineering. Microservices allow for a complex application to be developed and operated in a distributed way as well as increasing its resilience and scalability. Additionally, teams can select the languages and technologies to be used for each component service with more freedom, afforded to them by the loosened coupling between services. These services are loosely connected and typically interact with each other within their specific architecture using Representational State Transfer (REST) Application Programming Interfaces (APIs)[46], message queues or the newer service meshes. A completely separate team of developers can work independently on individual component services of a much larger application[47] and some of the resulting software artefacts can be reusable and thus placed in a hub or marketplace for other developers to adapt and integrate to other projects. Hubs, such as Docker Hub[48], Kube Apps Hub and the AWS Serverless Application Repository, offer a growing collection of available deployment-ready packages. These repositories of microservice artefacts are still relatively young, but already contain a wealth of data both in terms of metadata and runtime characteristics. Some of this data is versioned, but some can be lost through change, causing an irreversible loss of valuable data on the evolution of microservices.

Our motivation is exploring mining and exploitation techniques for taking advantage of this data, as well as providing a medium for data distribution and storage, that in unison would act as a catalyst for collaboration and further research. The main inspiration for this endeavour are existing global observatories in other fields. Such observatories are the primary reference point for the entire field, with access to the latest scientific developments at a glance, as well as a wealth of documents and data. Such an observatory is needed to preserve, study and further the rapidly expanding landscape of microservice architecture.

2.1.3 State-of-the-Art

Related Work

Various works have delved into the monitoring, testing or benchmarking of various aspects of microservices and the application architectures they can be used in. Some approaches focus on the metadata and logs generated by the microservices[49], some others are built to benchmark specific architectures or test dependencies and service-to-service interactions[50] and others on runtime monitoring of microservice-based applications. More recently, attempts are being made for benchmarking architectural models based on microservices to study the implications of the design pattern on real-world applications[51]. Work has been done to extract usable metrics for developers and corporate managers from the data obtained by monitoring these software artefacts. Effort has also been made to set a standard set of requirements for the orchestration of microservices. From a slightly different perspective, some surveys have been made to detect the trends in microservice development[52].

MAO and Predecessors

What was missing from prior works is attempts at systematic monitoring of the microservice-related marketplaces. Preceding this research is an existing collaboration of researchers in an effort to develop and operate research infrastructure to assess microservice artefacts, named the Microservice Artefact Observatory (MAO). Researchers at the Distributed Systems Research Group of the Zurich University of Applied Sciences (ZHAW) have made attempts to crawl through the Helm charts of the Kube Apps Hub[53] and the Serverless Application Repository[54] for QA and preservation purposes.

However, such a method of data crawling can never be made truly effective, as there will be breaks in its execution and data will inevitably be lost. Consequently, what would be needed is a truly distributed and decentralized architecture, that would allow the experiment to experience no downtime and grant it fault tolerance.

Though such a task has not been undertaken before in this sub-field, some examples of similar infrastructure are already there. Distributed architecture examples can be seen in Planetlab, BOINC, Cloudlab, the EGI Federated Cloud, GENI or EOSC. The existence of Data Management Platforms and the many Mining Software Repository papers show an insight into the data mining aspect of this work, though in this case what would be needed is a continuous stream of data. Experiments in the field include industry benchmarks like YCSB as well as others like CloudSim and DockerAnalyser.

2.2 Analysis and Improvement of Heterogeneous Hardware Support in Docker Images

We continue by presenting our work in using MAO in a real context, to collect data on publicly available Docker images. We used this data to study the use of hardware features in Docker images, and to propose a new metadata standard for Docker images, along with a wrapper software for applying it in practice.

2.2.1 Introduction

Portability is a desirable property in cloud computing[55, 56][57]. Virtual machine images, container images and other executable artefacts ought to be flexibly deployed across clouds[58], migrate from the cloud to the edge[59][60], or be confined within differentiated cloud security zones[61]. Nevertheless, limitations to portability arise from the differences in the underlying heterogeneous hardware. Apart from targeting the basic processor architecture, new cloud service paradigms such as accelerated computing or secure computing demand support for further hardware extensions abstracted as shared resources[62]. Any hardware thus requires support in the boot process (BIOS/EFI), operating system, virtualisation layers and application execution stacks, as well as in the applications and their constituent artefacts (e. g., images). Among the hardware features of interest to cloud application engineers are FPGAs, GPUs and nested virtualisation[63–65], as well as security-increasing hardware.

Secure (confidential) cloud computing is one of the emerging concepts affected by reduced portability: tenants offload computation to third-party untrusted providers without having to worry about security issues typically associated with cloud offloading[66]. Over the years, this concept has been mapped to concrete secure architectures involving monitoring and intrusion detection[67], algorithmic proofs on data, trusted boot[68], physical security including hardware security tokens and modules[69] and policy languages[70]. Recently, hardware-based trusted execution environments (TEEs) enabled outsourcing the processing of confidential data with high privacy requirements[71] towards untrusted clouds.

Mapping this concept to hardware requires support for special CPU instructions as well as appropriate configuration of hypervisors and other execution parts. Several vendor-specific instruction sets exist nowadays in processors available in the consumer market, as well as those offered by cloud providers[72, 73]. Specifically, we mention Intel Software Guard Extensions (SGX)[74], AMD Secure Enhanced Virtualisation (SEV)[75], or ARM TrustZone[76, 77].

In most cloud environments, Docker containers become the standard *de facto*, as application developers migrate towards integrated platform services, e. g., IBM Code Engine[78], Amazon ECS[79], Azure Container Service[80] and Google Container Engine[81]. Research on heterogeneous hardware support for container execution has been limited to few aspects, and in particular has not covered TEEs. For highly security- and privacy-sensitive software applications, such as those in the domains of e-health, banks or smart metering, the use of TEEs offers many benefits. To drive their adoption, it is necessary to encapsulate the use in containerised form. Yet container runtime support for heterogeneous hardware in public clouds is still lacking, in particular for secure execution. This is surprising given that certain features are today available to providers of cloud infrastructure-as-a-service. For instance, all major cloud providers offer instances with SGX-/SEV-capable CPUs, nevertheless access to consumers is either limited or disabled. More specifically, Azure offers SGX/SEV capabilities with their Confidential Compute instances, while IBM or Alibaba have them enabled only for their bare metal instances. AWS and Google do not provide SGX-enabled instances, as the feature is disabled in their systems' BIOS.[1] In general, no cloud provider natively supports a container runtime that makes use of SGX. We believe this situation is going to fundamentally change over the next few years, and in that perspective, the research focus will shift to provide efficient support in the application stacks and artefact management tools, nowadays largely container-based.

[1] Google just recently announced SEV-enabled instances[72], while AWS is introducing Nitro Enclaves, heavily inspired by Intel SGX[82].

To allow a Docker container to utilise a hardware-specific feature of the host system, the relevant device driver or kernel module needs to be explicitly mounted to the container at runtime. Some features, such as SGX, can be configured to run in simulation mode in the absence of supporting hardware. Base images that offer the relevant development tools for these features are sometimes provided by the hardware vendors themselves (e. g., Nvidia Docker[83]), other companies (e. g., SCONE[84, 85]) or by members of the developer or research community. For the SGX use-case, the Graphene Secure Container Environment (GSCE)[86] aims to wrap the standard Docker engine to allow standard images to run their applications within SGX enclaves, although this is in preliminary stages of development.

While our analysis shows that only few Docker images require specialised hardware at the time of writing, this is expected to change in the future. Already now, the discoverability of hardware in the form of CPU architectures per tag is limited, leading to issues in mixed environments. Making conscious use of the hardware features requires knowledge about them, which is presently absent from container images and handling tools and motivates our research. By consciously leveraging specific hardware types, augmented images can be invaluable to certain sectors, e. g., Highly Information Sensitive Computing (HISC), High Performance Computing (HPC) and FPGA-hosted blockchain-as-a-service[87, 88], enabling the integration of containers to their niche workloads.

The contributions of this paper are fourfold. We introduce a *systematic analysis* of the content of Docker images with emphasis on heterogeneous hardware, especially on TEEs for security and GPUs for acceleration. In contrast to existing empirical works, our analysis follows a three-staged process: automated monitoring for metadata inspection, static image inspection, and manual runtime inspection. We also contribute *augmented metadata* that reflects the analysis results, and an improved *hardware-aware Docker executor* `hdocker` that parses the metadata to increase upfront assessment about the likelihood of hardware-dependent execution failure. We follow an *open science* approach. Our tools and datasets are released and available from `https://doi.org/10.5281/zenodo.4531794`.

2.2.2 Background

Processors and Hardware Heterogeneity

At the most fundamental level, hardware support means the ability to execute on a given processor architecture. Public clouds typically include `amd64` (`x86-64`) and `arm64`, with minority occurrences of `i386`, `ppc64el` (IBM) or `mipsel` (Loongson). However, discussions of heterogeneous hardware refer to a broader set of features, such as specific instruction sets. When supported by BIOS/EFI, hypervisor and operating system, applications can typically rely on more than 100 unique CPU flags, e. g., `sse3` for streaming operations or `avx` for vectoring. This also includes hardware-assisted TEEs, i. e., Intel SGX and AMD SEV or hardware-assisted virtualisation technology (ie, Intel VT-x, AMD-V, HVM, etc). Similarly, another group captures support for expansion devices or cards. This became increasingly relevant in recent years, by the usage of GPUs for high-performance computing, machine learning and simulations. We include in this group support for fast/low-latency disks (e. g., NVMe non-volatile memories) and network links (e. g., InfiniBand).

Docker Images

Since its introduction in 2013, Docker emerged as one the most popular cloud-native software artefacts and as such an integral part of modern DevOps operations Several frameworks and

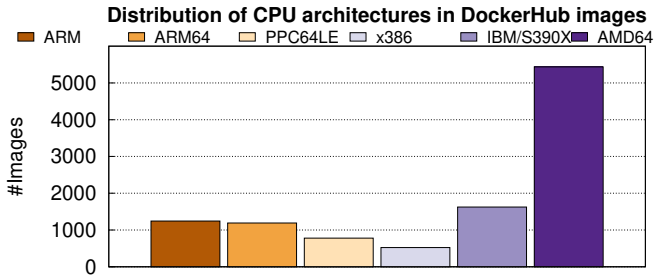

Figure 2.4: Distribution of image types in an ∼11'000 images non-random sample collected as of February 4^{th}, 2021. X-axis: processor architecture.

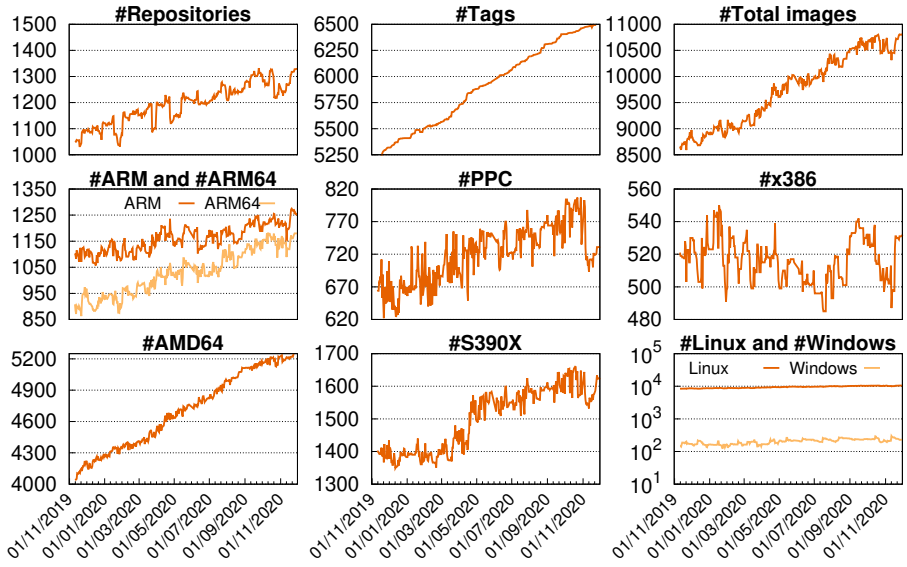

Figure 2.5: Monitoring metrics for Nov 11, 2019 to Feb 4, 2021

development environments are now available as drop-in solutions using Docker, significantly reducing the overhead of both setting them up and orchestrating them.

Images can be distributed in two ways. The first largely adopted method is using a container registry, the largest one being DockerHub.[2] There are private registries, curated by specific vendors[3], which incorporate stricter standards, i. e., built-in security scanning of all images motivated by several security issues found on public registries[89]. The second distribution method is to include the Dockerfile in a publicly accessible code repository (e. g., GitHub), for users to build the image themselves[90].

Docker has support for multiple CPU architectures, namely `x86`, `x86-64`, `ARM`, `ARM64`, `PowerPC 64 LE` and IBM's POWER and Z architectures. We fetched and analyzed a sample of the publicly accessible Docker images on DockerHub, constituted by approximately 11'000 images (as we detail later in Section 2.2.3). We report these results in Figure 2.4 (also released to the research community). Unsurprisingly, `x86-64` leads by a large margin with ∼48% of the images. Interestingly, IBM Z (`S390X` in Figure 2.4) is second most popular (with ∼15%). We observe that `ARM64` is the fastest trending image type, after `x86-64`, showing a 1.7% monthly increase (Figure 2.5). Docker has recently introduced an experimental multi-architecture builder in which several versions of an image, compatible with multiple

[2]Docker Hub: `https://hub.docker.com/`
[3]Red Hat Registry: `http://quay.io`, Tenable: `http://tenable.io`

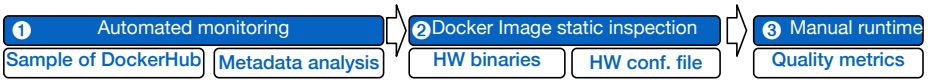

Figure 2.6: Our methodology to gather and validate DockerHub's image data/metadata.

processor architectures are built simultaneously, thus speeding up the process of introducing compatibility for different architectures.

Metadata available from the official registry provides basic overview metrics, such as supported architectures and OS versions, as well as the number of images and tags within a repository. However, global statistics cannot currently be obtained from DockerHub, as the API endpoint for the global catalog metadata is disabled. Utilization of Docker images in contexts that require specialized hardware is now an emergent use-case, varying from Nvidia GPUs to Intel SGX-compatible CPUs, though adoption appears to be at an early stage, judging from the number of repositories and their omission from the list of official images. They require the host to have the correct hardware and driver support installed, and are thus limited in their reach to end users, but provide the ability to leverage Docker's advantages for applications that require specialized hardware features, such as secure execution and GPU-accelerated HPC. Still, metadata does not reflect hardware dependencies, and that makes such repositories difficult to detect with automated means, which is why we searched by specific vendors (namely Nvidia and Intel) and hardware features (CUDA and SGX) heuristically to create the sample for the evaluation of hardware support.

2.2.3 Methodology

The methodology followed to assess the hardware support in Docker images is divided into three steps. The first one (Figure 2.6-❶) consists of automatic monitoring of the main DockerHub repository to retrieve a representative sample of the most used images. The metadata from the sample images is then stored and analysed for information on any specific hardware they support or require. The second step (Figure 2.6-❷) dives deeper into the images and attempts to find any traces of hardware-specific binaries or configuration files. Finally, the third one (Figure 2.6-❸) performs basic manual runtime testing of a targeted selection of Docker images from DockerHub that use specific hardware features, to assess their quality and maturity.

The automated sampling of DockerHub metadata uses a two-tier approach. The search starts with the official Docker images, collected in a private *library* repository. The metadata from this repository is collected and added to the dataset. For each entry in the resulting data, we assume each image to correspond to an organisation, which maintains its own repository. For example, the image `nginx` is controlled and maintained by a corresponding `nginx` organization. We thus repeat the search for each of these possible organisations, adding the data returned (if any) to the dataset. This process essentially creates a shallow family tree of two levels, stemming from the official images. This serves to effectively increase the sample size by an order of magnitude, granting us a higher number of images to investigate. This selection process is scripted and both it and the metadata analysis runs nightly on a scheduled monitoring setup.

Compared to typical random sampling, this method has the advantage of specifically targeting repositories within DockerHub that are considered of high-quality/high-profile, as their maintainers are either verified or certified publishers. It should be noted that the goal of this sampling is to feed the next steps with potential targets, not to obtain global metadata statistics for Docker Hub as a whole. As such, the statistics are meant as an indication of

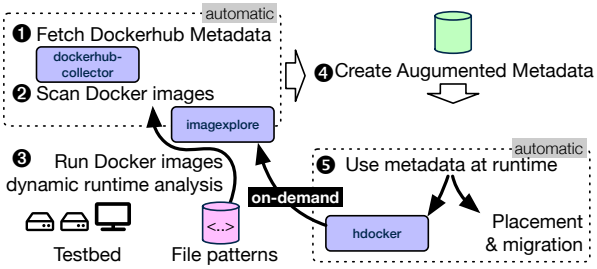

Figure 2.7: Data-driven process: from experiment (steps ❶,❷,❸) to augmented metadata usage (❹,❺); `dockerhub-collector`, `imagexplore`, and `hdocker` are open-source.

hardware support in the official and related repositories and do not represent a study of Docker Hub as an ecosystem.

Currently, the metadata provided by DockerHub's API does not provide any insight on whether or not an image requires specific hardware to be present on the host in order to function. A possible *hardware features* field could rectify this. In the present state of the metadata, the only way to select images with specific hardware requirements beyond CPU architecture is to rely on the web front-end's text search function, a rather cumbersome and unreliable approach. For example, a search of the keyword 'SGX' with the intention of discovering images that utilise Intel SGX, will also yield images submitted by users with the characters 'sgx' in their username. Naturally, such a search will exclude any images that use the hardware but don't include it in their name or description.

However, due to the relatively small number of images using targeted hardware technology, it is still possible to hand-pick a sample and assess them heuristically for typical quality metrics, such as compatibility, documentation and best practices. For the sake of illustration, we restrict the choice of the target technologies to: *(1)* Intel SGX, as a secondary processor architecture feature with high significance for security-conscious use cases, and *(2)* support for Nvidia GPUs, arguably the most common example of heterogeneous computing power. For the former, adoption is still rather limited. We believe this to change as SGX drivers have just been up-streamed into the mainline Linux kernel[4]. For the latter, adoption is more wide-spread, with many individual user-contributed repositories, so the search is narrowed down to the images contributed by the hardware vendor themselves, and restricted to static analysis.

2.2.4 Evaluation

We ran three separate experiments, one for each of the aforementioned methodic steps. Figure 2.7 gives an overview of the experiments, the resulting augmented metadata and its use in specific container and cloud management tools.

Automated monitoring. We setup automated long-term monitoring and tracking of Docker container images via the global Microservices Artefact Observatory.[5] The observatory creates execution schedules for each registered monitoring tool, and shares these tools with other nodes within a federated cluster architecture for highly reliable observation and metadata retrieval. The `dockerhub-collector` tool periodically requests DockerHub's public API to retrieve metadata on the 'library' repository. This repository is unique: it is curated by DockerHub itself and contains 'official images', i. e., the most useful, popular and high-quality

[4]42nd rev `https://lore.kernel.org/lkml/20201214114200.GD26358@zn.tnic/`
[5]MAO: `https://mao-mao-research.github.io/`

images available. Using this data as a starting point, we identify 'sister repositories', i. e., attempt to see if the developer of each image maintains their own repository, for which we request metadata. This selection process is integrated into the tool and automated. It increases the sample from the 164 repositories of the official set to ∼1200 repositories.

The continuous metadata collection has been running on each machine in the cluster daily, starting on November 11, 2019. The data snapshots obtained by each machine are compared for inconsistencies and then aggregated into a single dataset to analyse the hardware trends. The images with SGX and Nvidia support are selected manually in irregular intervals, based on a search of the repository for the specific hardware feature as explained before. The final lists describing the samples, after filtering out duplicates, images without any version and irrelevant repositories, are included in the dataset to ease reproducibility. Thus, we curated multiple lists of images as February 2021 snapshot for the evaluation: full library and associated images (∼11'000), strict library subset (164), SGX (67) and Nvidia (36). We applied static scanning to the last two, and dynamic assessment to the last.

Static image inspection. To gain further insights into the hardware-related aspects of container images, we developed `imagexplore`, a tool to uncompress all layers of an image and recursively search for the occurrence of certain files, directory and content patterns. These come from a knowledge base created with file glob patterns as well as text and binary symbols hinting at the presence of hardware-specific features. Matches result from the installation of the SGX toolkits, Nvidia device drivers, USB devices or other hardware-related software leaving traces in the system. For example, an SGX base image needs to contain the SGX SDK. Thus, toolkit files can be detected in a predictable manner to confirm the presence of the SDK. Extending the knowledge base with additional patterns for different hardware devices enables the detection of images that utilize them, giving information on potential deployment issues due to hardware incompatibility. Evidently, this experiment leads to suspicions but not always to verified cases due to false positives. Hence, our method encompasses the manual verification of any edge cases. Furthermore, the scanning encompasses all files in the `/dev` file system, as those are not supposed to be part of any container image but their accidental addition may reveal further insights. This scanning phase is performed for all the automatically retrieved library and SGX images.

Manual runtime inspection. Beyond the static testing, we conduct dynamic runtime invocation on the SGX images to find out the actual use of any TEE-related operation. This test is run on an SGX machine using an Intel Core i7-8650U. We categorised the SGX images into *(1)* base images and *(2)* those for applications or tools. We assessed them on the presence of documentation, instructions and the ability to initialise on an SGX-compatible machine. It is important at this stage to define what 'initialise' means in this context. For an application or tool, this is trivial: we expect some program output or the successful initialisation of a service. For a base image, we expect to setup a functional development environment when we run the image in interactive mode. This was achieved by inspecting the image to ensure the SDK and relevant dependencies are present. In the case of curated images, code samples are provided in the documentation that can be used to test the image. Due to the small number of SGX images, we allowed a minimal troubleshooting, which we define as being able to run the images by just following the instructions (if any). In case of errors, we see if it is straightforward to resolve, e. g., mounting a file in a volume or specifying a command-line argument. If not, the image is deemed as not working, although its usability mainly depends on incomplete (or lack of) documentation. The minimal configuration possible for an image without instructions or other forms of documentation is to run it with no options besides mounting the SGX-related virtual devices from the host (i. e., `/dev/isgx` and `/dev/mei0`).

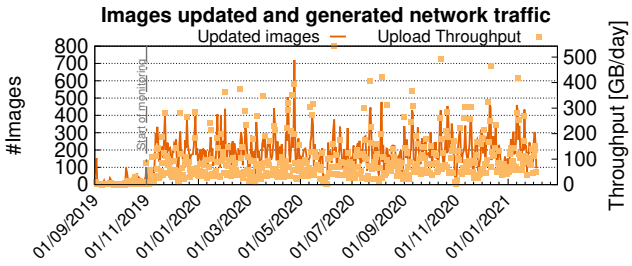

Figure 2.8: Images updated and data uploaded from Sept'19 until Feb'21; `y2axis` (right side): inbound network traffic originated by updates of the images on the `y1axis`.

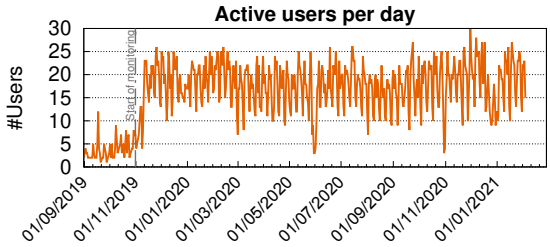

Figure 2.9: Active users by day from Sept'19 until Feb'21.

For base images, this proved sufficient on the majority of occasions, as the images contained all the development tools they needed except for the hardware driver component.

Results

Automated monitoring results. The automated monitoring of Docker Hub's public metadata gives the basic overview of processor architecture and operating system support discussed above. While not novel in itself, we further consider the amount of activity on the chosen sample during the same time period. Figure 2.8 and Figure 2.9 present respectively the amount of image updates and total data volume per day, as well as the number of unique usernames that appeared in each day's data, extracted from the same dataset. The goal of this observation is to ensure that our sample does not come from an inactive subset of Docker Hub, and thus reflects recent trends. Despite this being a small subset of approximately 1200 repositories, it gives a sense of the activity and data throughput per repository in Docker Hub. In the observed period, the total amount of data pushed to these repositories was ∼23TB.

Static image inspection results. We run the `imagexplore` tool on three sets of container images: *(1)* 67 SGX candidates, *(2)* 36 Nvidia candidates, and *(3)* 164 official library images as subset of the sample whose metadata we continuously retrieved over multiple months. Due to the absence of a `:latest` tag or non-public images, not all pulls succeed, thus these numbers are slightly reduced especially for less maintained non-official images for which non-default tags have therefore been recorded. All layers of all images are unpacked on disk for the analysis. We expect the tool to indicate 100% SGX and Nvidia dependencies in the first two sets (minus false positives), and 0% any hardware dependency in the third set (minus false negatives). Table 2.1 compares the results. Among the SGX candidates, four use the name coincidentally but do not contain any SGX-related code. Similarly, among the suspected Nvidia images, four contain code to run the Kubernetes-based EGX AI platform that is offered by Nvidia but do not contain any hardware-related code themselves. Among the library images, three are operating system distributions with support for USB devices

Table 2.1: Automated static analysis of the SGX, Nvidia and library container samples.

	SGX images	**Nvidia images**	**Library images**
Count	67	36	164
Successfully retrieved	62 (93%)	14 (39%)	156 (95%)
Cumulative disk size	61 GB	15 GB	75 GB
Hardware dep detection correctness	58 (94%)	10 (71%)	152 (98%)
Detection time	1876.7 s	211.0 s	1080.4 s
Exceptions	False pos.	False pos.	False neg.
Excepted images	sgx-django, sgx-bootstrap, sgx-cosmos, payload-ethash	egx-*, eac	odoo, crux, sourcemage

Table 2.2: Device files in the SGX and library container samples.

	SGX images	**Nvidia images**	**Library images**
Average /dev layers	2.1	1.0	1.0
Multiple /dev layers	10 (15%)	0 (0%)	8 (5%)
Shadow /dev files	27..36 (53%)	1..2 (14%)	6..11 (7%)
Further /dev files	27 (40%)	1 (7%)	3..8 (5%)

that are successfully detected, whereas the vast remainder are pure application containers. The average scanning time per image already present locally is 7.0 s for library images, 15.1 s for Nvidia images and 30.3 s for SGX images on a server with Xeon E3-1270 CPU @ 3.80GHz. The main contributor to the significant difference is that most library images are rather tidy base images whereas especially the SGX images contain many layers with thousands of added files.

Concerning the `/dev` file system, the official images show better quality compared to the SGX images. However, in both sets, a lot of superfluous files were found. The Nvidia images are quality-wise in between with still a few of such files. Most of them mirror the standard device files added automatically by Docker, whereas almost as many consist of various other files to access audio devices, RAM disks, shared memory and other hardware. We suspect careless creation of images by snapshotting running container instances without proper post-execution cleanup of device files is the main reason. This hypothesis is supported by the occasional leftover of temporary files (`/tmp`, `/var/tmp`) in both sets of images. Table 2.2 compares the device file findings, with the optimal metric being 0 in all rows. The table shows how many images had multiple `/dev` layers, and how many layers on average those images had.

Manual runtime inspection results. The mentioned categories yield 33 base images (49.25% of total), 29 apps/tools (43.28%) and 5 images (7.46%) whose functionality could not be determined due to lack of documentation. The latter ones are excluded from the percentage calculations that follow. As for quality metrics, 47 of the images (75.81%) initialised successfully when tested and only 14 of them (22.58%) had any documentation or instructions.

Considering Docker Hub's limitations on making images private, the bulk of the undocumented images are likely experiments that were not meant to be shared. Even within these 67 images, a certain degree of deduplication was possible. Upon manual inspection 28 of

the images were derivatives of others in the same set, of which 18, based on their functionality, were duplicated of earlier versions of other images. Examples include several variations of the SGX base image, which provides the SGX SDK and Platform Software (PSW) and an example 'hello world' application, cementing the observation that many images available were simply the results of users experimenting with the technology, or following tutorials. The lack of documentation and the amount of 'hello world' duplicates in the sample is a clear indication that the majority of images that use the SGX technology are experimental, and thus adoption of the technology is in its infancy in the Docker ecosystem.

In terms of observed errors among the 15 failures, there were image initialisation errors for 4 images and an enclave initialisation for 1. Others were application-specific errors. It is important to mention that as with any heuristic evaluation, the testing methodology can affect the final result and as such, the 15 failures have to be treated as an upper bound. As a secondary experiment, we attempted to run the tools that ran successfully, but without mounting the SGX devices. For them, 8 out of 15 ran successfully in simulation mode, with only two 'hello world' applications and an SGX-specific stress testing tool failing due to enclave initialisation errors. The 4 remaining failures were instances where the output of the program made it difficult to understand if initialisation was successful, ie possibly false-negatives.

The high success rate of the base images, even those with no documentation, also has a simple explanation. These images usually only include the SGX dependencies (the SDK and PSW) and typically some language specific dependencies for development. Thus, generically running them interactively by simply mounting the SGX device drivers is expected to yield a high degree of success. The spreadsheet used to track this heuristic evaluation is available in the publicly released dataset.

Images with support for Nvidia GPUs are far more numerous, potentially up to 1411 image repositories in our latest snapshot. We note that the company maintains 2 distinct organisation accounts (`nvidia` and `nvidiak8s`, respectively with 22 and 12 repositories), the latter focused on Kubernetes-related images. A common characteristic of both is the almost complete absence of documentation within Docker Hub itself. Instead, all relevant documentation is available in Nvidia's Github repositories, where the source code for these images can also be found. The absence of links from the Docker Hub repositories to the ones in Github however, means that the only reliable source of information on these images and their usage is Nvidia's Github. This is by no means an oddity, as typically Docker Hub is used to store images for public distribution and serves as a medium for users that wish to use a specific tool without building from source code manually, one of the main advantages of Docker. In that context, only having minimal documentation on Docker Hub (usually related to differences in using the image compared to running a tool natively) makes sense, as the main source code repo is responsible for housing the full documentation.

Augmented Metadata

As mentioned earlier, the existing metadata schema for Docker, a sample of which can be seen in Listing 2.1, does not contain hardware details beyond the CPU architecture. As such, Docker images built to utilize specialized hardware can benefit from augmented metadata to include hardware requirements. Such images are still a relatively small niche within the ecosystem. However, additional metadata can benefit users attempting to deploy them, as a clearer picture can be obtained by the Docker client on whether the image can be expected to function in the target system. To this end, we propose hardware-specific fields be added to the

Listing 2.1: Existing Docker metadata schema sample.

```
{
  "library/mongo": {
      "count": 635,
      "next": "https://hub.docker.com/v2/repositories/library/mongo/tags?page=2",
      "previous": null,
      "results": [
          {
              "name": "3.6.15-xenial",
              "full_size": 165560728,
              "images": [
                  {
                      "size": 165560728,
                      "digest": "sha256:e224e342ed62274df41a7f57e6c1a221abd7a58ead788a5fb124eb5f15fd2ea6",
                      "architecture": "amd64",
                      "os": "linux",
                      "os_version": null,
                      "os_features": "",
                      "variant": null,
                      "features": ""
                  }
              ],
              "id": 75028785,
              "repository": 21412,
              "creator": 1156886,
              "last_updater": 1156886,
              "last_updater_username": "doijanky",
              "image_id": null,
              "v2": true,
              "last_updated": "2019-11-05T05:08:24.84415Z"
}]}}

```

Figure 2.10: The existing metadata schema for Docker images.

metadata of Docker registry entries. Furthermore, with Docker Hub's new rate-limiting policy (i. e., 100 image pulls every 6 hours), pulling an image simply for the sake of inspecting is problematic for free-tier users: with better metadata, operators can make informed decisions about image pulls ahead of time. Note that part of the additional information we propose to include is contained in the experimental image manifest v2, schema 2 of the Docker registry API specification[91] (see Listing 2.2, `features` element). Evidently, this implementation has some limitations. First and foremost, the image manifests are not available via the public API. As a consequence, one must pull the image and inspect it manually, a process that is hard to automate and requires storage space and network bandwidth for then potentially non-executable images. For instance, `docker run` will currently pull an AMD64 image on ARM before complaining about incompatible binary formats. Secondly, in terms of hardware,

Listing 2.2: Hardware features extensions.

```
{
  "platform": {
    "architecture": "amd64",
    "os": "linux",
    "features": [
      "sse4"
    ]}}
```

Listing 2.3: Support for heterogeneous hardware sources.

```
{
  "platform": {
    "cpu": {
      "architecture": "amd64",
      "features": {
        "SGX": "supported",
        "sse4": "required"
      }
    },
    "hardware": {
      "GPU": "nVidia"
    },
    "os": "linux"
  }}
```

Figure 2.11: Support for heterogeneous hardware sources.

only CPU architectures and instruction sets are considered. Consequently, one cannot identify if an image uses GPUs or FPGAs from its manifest alone. Finally, CPU architecture variants and features are written to the manifest manually but cannot be verified during validation. This leaves the inclusion of this information at the discretion of the developer and the validation up to the end user.

Thus, we propose an extension of the platform information that distinguishes CPU architecture and features, as well as other hardware devices supported. Listing 2.3 shows an example in augmented JSON format that serves as downward-compatible replacement for the current Docker registry metadata. We argue that it is important to distinguish a hardware feature as `supported`, e. g., its functionality can be emulated/simulated if the hardware component is missing), or `required`, e. g., the container will fail without it.

Moreover, specific hardware support from the host needs to be passed through to the running container. Hence, rather than replicating pass-through parameters in all scripts and composition documents (Listing 2.4), we argue for tightly binding them with the container metadata. Similar to how permissions are confirmed by users of mobile phone applications, administrators can interactively approve applications based on displayed host device access permissions. In addition, some containers require volumes to be mounted, or access to the Docker daemon socket, to be indicated in the extended metadata structure.

Improved Container Image Tooling

Finally, we introduce `hdocker`, a hardware-aware wrapper around the `docker run` command, to perform several tasks that streamline the use of Docker containers across heterogeneous hardware architectures. Specifically, `hdocker` can:

- Report on the availability of tags per architecture and architectures per tag.
- Check whether CPU flags and hardware devices are present and activated, beyond the standard CPU architecture and operating system.

Listing 2.4: Pass-through support.

```
{
  "platform" as defined above
  ...
  "passthru-devices": [
    "/dev/kvm", "/dev/net/tun", "/dev/bus"
  ],
  "volumes": [
    "/run/secrets", "/var/run/docker.sock"
  ]}
```

Figure 2.12: Pass-through support.

- Check whether device files and volumes are appropriately declared to be mounted in the invocation parameters.
- Inform the user with clear actionable advice in case of unsupported features.
- Automate the analysis (using imagexplore) and decisions if asked to do so and if possible.

The `hdocker` tool is metadata-driven, as it parses the augmented metadata curated from our experiments. Unless cached locally, it downloads all container image layers, runs `imagexplore` over all images, and store the resulting JSON metadata files. If the registry uses our augmented schema to retain this information, end users would not have to perform the image pull and analytics to obtain the hardware requirements of the image and `hdocker` could configure the hardware pass-through automatically.

Beyond the designed and implemented `hdocker` CLI, we also envision further tools to assess the metadata in a similar way. Among them are autonomous container migration tools that work as target node filters in heterogenous computing continuums (ie cloud/edge/fog/IoT or osmotic settings[92]).

2.2.5 Discussion

While Docker images that support or require specialised hardware still comprise a very small share of the total available images, we argue that their significance to industries that make use of heterogeneous or specialised hardware justifies communicating support for such features in an image's public metadata.

Our proposed support is nevertheless not a simple addition to the metadata schema. There are certain requirements for making its adoption meaningful.

Docker CLI support. The hardware features would have to be specified by the user through the Docker CLI during a build, or detected by the Docker builder. The latter is harder to implement, but would enforce the hardware specification, whereas developers would neglect specifying the hardware requirements.

Dockerd support. When attempting to run an image with specific hardware requirements that are not met, the Docker daemon would need to alert the user of the issue, rather than `hdocker`.

Docker Registry support. The hardware-related metadata would have to be part of the public Docker registry API, so that end-users can make informed decisions on the images they deploy ahead of time. The need for this is accentuated by the rate limit introduced by Docker Hub recently, forcing operators to be mindful of image pulls they make.

2.2.6 Related Work

Analyzing and understanding large container datasets offers substantial insights into current trends among practitioners and DevOps engineers' best practices. In[93], authors produce a large-scale dataset of more than 450'000 Docker images by crawling Docker Hub, downloading and decompressing images and analysing their contents. The dataset is populated with file-level metrics such as compression ratios, file types and sizes, and content-level metrics, i. e., programming languages, compressed files and database systems. Despite rich metadata including deduplication metrics, the underlying OS or architecture features are ignored. Moreover, the dataset is not made available, forcing the research community and ourselves to implement, deploy and run our own crawler to get complementary insights. While our automated monitoring is on a smaller scale, we produce and release a curated dataset that provides historical metrics by monitoring at regular intervals over a longer period of time. Our dataset also serves as an example of the type of insights that could be obtained by applying our methods to other repositories, such as those maintained by specific organisations.

Approximate Concrete Execution[94] applies binary analysis to container images used in serverless platforms such as Google Cloud Run and AWS Fargate. It uses function-level fingerprinting to identify security problems in an early stage, based on a database associating fingerprints with such issues. The same technique could be used to fingerprint hardware dependencies. For simplicity, we look at container images at a coarse-grained level – if any script or application therein depends on a specific hardware feature, no matter if this is run on startup or just placed accidentally in the image, we consider the dependency to exist.

Several container orchestration tools have been recently proposed specifically for heterogeneous hardware clusters. However, they lack the necessary detection and expression of hardware dependencies within the container images or composed applications. DRAPS[95] focuses on resource allocation, and thus in quantitative differences in CPU performance, amount of RAM and network/block device I/O latency, whereas our work investigates hardware differences at a deeper and qualitative level. Our work is complementary and necessary to further improve heterogeneous cloud application orchestration with upfront predictability about the likelihood of failure.

Dockemu extends the network simulator NS-3 with portable containers to cover IoT scenarios[96]. It supports heterogeneity in nodes and links between them, but does not cover hardware differences in detail. We anticipate the use of our results for building better heterogeneous cloud simulators and emulators.

The metadata expressivity limitation of Docker Hub and privately deployed container image registries have been widely covered by researchers. ConHub[97] is an extended metadata management system for Docker images on top of a relational database. It supports CQL queries and user-defined metadata, letting users manage hardware-dependent images when importing augmented metadata, similar to the ones we propose. Similarly, Docker2RDF[98] and DockerFinder[99] offer queryable endpoints that will benefit from richer descriptions including on hardware restrictions at pull time. While we focus on hardware features, studies exist[90] to analyse image content in terms of programming languages, runtimes, security vulnerabilities and reproducible builds, leading to further metadata.

2.2.7 Conclusion

Our multi-method analysis of Docker container images identified limitations on the handling of heterogeneous hardware, an increasingly relevant matter in typical container environments such as differentiated cloud services. We released a new DockerHub dataset on the metadata

of ∼11'000 Docker images over the period of more than one year. We offer insights on available and used hardware features, e. g., related to security (SGX) and machine learning (Nvidia). We validated initial search findings with static analysis, and cross-validated those with runtime testing. Further, we contributed new proposals to augment the container image metadata with hardware, device and volume information as well as tools to exploit the knowledge at runtime for more user-friendly application handling. Overall, we observed a solid growth of images and tags across architectures, with the exception of `x386`, whilst a low number of non-toy, non-base images exploiting specific hardware beyond the basic CPU architecture. In addition to our work on hardware support, our metadata analysis was later extended to include other aspects of Docker images, namely Common Vulnerabilities and Exposures (CVEs) and the presence of malware. The augmented metadata was used for IoT-Fog-Cloud continuum deployment matchmaking[100].

2.3 Reproducible Batch Data Acquisition

It quickly became apparent that a generic execution system based on Docker containers is not sufficient to ensure reproducible data acquisition and data provenance in a complex system. This is because we need to reproduce the entire execution environment, the parameters of each software tool used, and to record the data provenance information before and after every step. We thus investigated the integration of our workflow with an existing framework based on executing data science workflows in a reproducible manner, and maintaining a knowledge graph for data provenance in each workflow step. We selected the Swiss Data Science Center's Renku platform for this task. We continue this chapter by presenting the Renku platform, and how we utilized it as a reproducible environment for executing data analysis on data obtained using the MAO framework. This integration allows all steps in the workflow, from acquisition to analysis and generation of insights, to be reproducible, thus allowing more complex workflows to be built using our framework.

2.3.1 Significance

Microservice artefacts are basic units of software which are composed to yield executable, distributable and scalable applications. Often appearing as single (atomic) files or compound archives, the characteristics of each of them influence the overall characteristics of the resulting applications at runtime. Software technology researchers are thus interested in code analysis, static property determination at the artefact level, and dynamic runtime assessment at one point in time, as well as in the evolution of these metrics over time. From the plethora of already published works, one can estimate the multidimensional space of research on microservices:

1. **Artefact types:** There are conventional artefact types per programming language (for Platform-as-a-Service, or PaaS), such as Java's web application archives (WAR), Python's egg files or Ruby's gems[101], and the respective source files. Often, they are set up as microservices using glue code. There are also directly executable language- and provider-specific cloud function artefacts (for Function-as-a-Service, or FaaS), such as Google Cloud Functions and AWS Lambda, and smart contracts (for Blockchain-as-a-Service, or BCaaS), such as Solidity[102]. Polyglot artefact types further encompass container and virtual machine images (e.g. Docker, OVA) and their build documents

(e.g. Dockerfiles[103]). Finally, there are composition documents (e.g. Docker Compose, Kubernetes manifests) and even bundled compositions (e.g. Helm charts).

2. **Artefact repositories:** The spectrum ranges from generic software hosting (e.g. private and public GitLab/GitHub instances) over specialised repositories (e.g. Docker Hub, Maven Central) to hybrid solutions (e.g. Artifactory). Often, there are quality and activity differences between them, making it necessary to use many of them in combination as representative samples.
3. **Lifecycle:** There are works concerned with the design and proper abstraction levels and factors[104], implementation techniques, code transformation, deployment[105], live migration, benchmarking[106] and self-management.
4. **Research methods:** There are empirical studies with interviews[107], quantitative studies based on data collection[108], simulation[109], experiments[110], as well as design science approaches[111].
5. **Research data handling:** Several authors publish no data at all. Others point to datasets, in some cases even reproducible or evolvable ones. They fall into different categories of the reproducibility spectrum. In contrast to other communities, there are no standard repositories for insights into microservices, and few widely accepted reference datasets. Feeding machine-readable data back into development and runtime tools around microservices is however considered useful, as evidenced by the proliferation of knowledge bases[112]. Furthermore, several studies make wide assumptions that can be tightened with representative data samples.

The Microservice Artefact Observatory (MAO) has been designed to overcome the differences and difficulties in automated acquisition, aggregation, verification and analytics of the research data related to microservices. Its main features are federation of independently operated nodes involving several research institutions to establish ground truth based on consensus, resilient operation to ensure the data analysis pipeline is working flawlessly, and extensibility for new microservice technologies.

At the single node level, there is apparent overlap with other data management and monitoring frameworks existing in the cloud-native space. However, MAO's main feature is the distributed and collaborative environment it enables. MAO integrates data acquisition tools, but those are meant to be generic tools scheduled for automated operation, therefore differentiating it from monitoring frameworks, such as SonarQube[113]. While MAO has a basic data workflow automation system, similar to frameworks such as Airflow[29], Dagster[30] or Prefect[114], it is primarily a distributed collaboration tool, rather than an automation framework. MAO's primary feature is the ability to share experiment code and configurations between collaborators, and arrange the resulting reproduced data in a way that it can easily be compared and cross-verified and to partially automate this verification.

Once all data has been aggregated and confirmed, the scope of MAO ends. In this article, after presenting MAO itself, we thus demonstrate how to complement it with Renku.

As opposed to static data repositories, Renku serves not only as one-time sink but rather as a long-term platform for other researchers to curate, augment and engage with the data continuously injected by MAO for increased reproducibility (see Figure 2.13).

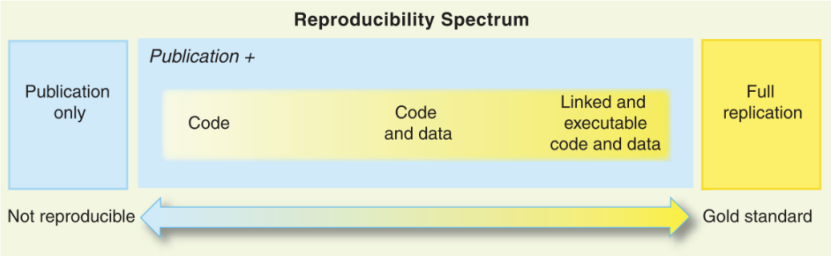

Figure 2.13: Reproducibility spectrum (source[115])

2.3.2 Software description

Microservice Artefact Observatory

The task of the observatory is to gather information and insights on software artefacts in the form of metrics which allow for assessing entire software applications in which these artefacts are used as microservices. The metrics may refer to completeness of metadata, code quality, presence of security vulnerability, unit test coverage, startup latency, robustness in the presence of fuzz testing or elasticity, among other characteristics. The specifics of these metrics are user-defined. To convert an observation tool for deployment to MAO, it only needs to adhere to a specific folder structure and be accompanied by a simple metadata file that acts as a specification for the data the tool produces. As with all automated solutions, the long-term aim is to remove the need to conduct isolated and error-prone data acquisition for each study. Instead, researchers can just query a set of metrics over a set of artefact repositories within a time window, and base their analysis on the query result which is guaranteed to be stable, verified and citable.

The observatory consists of an generic underlying federated infrastructure in which nodes are operated independently and coordinated via consensus voting. Nodes fetch new data and/or create new data through aggregation and analytics. All operations are meant to be resilient, automatable and auditable in the interest of providing a ground truth knowledge about the software artefacts with the highest possible level of verification from trusted operators. This knowledge is then suitable to be cited and used for further studies by researchers. Specifically, MAO aims at these factors:

1. **Automation:** Repetitive data acquisition and exploration should be repeated with fixed time periods to yield a precise representation. A task scheduler is therefore a key component, while still allowing ad-hoc invocation for early experiments.
2. **Resilience:** In case a network connection fails or a scheduled task produces no or evidently unusable results, it should be marked for repetition within a tolerance window after the originally scheduled time. To tolerate more extended failure scenarios, redundancy can be achieved by scheduling an experiment on multiple nodes.
3. **Voting:** If there are gaps or conflicts in the output data of nodes that are running the same experiment, a conflict resolution system based on simple voting algorithms will attempt to provide a solution that acts as the ground truth. Researchers can specify which algorithm they wish to be used for this in their metric specification file. At this stage the prototype implementation supports majority voting, weighted mean voting or the option to list all diverging values instead of resolving the conflict automatically.

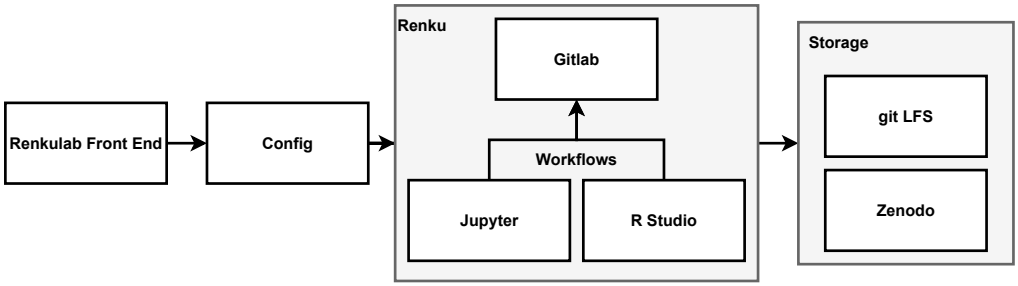

Figure 2.14: The Renku Platform (source:[7])

4. **Audits:** Information about when which data were acquired or which experiments were conducted is logged along with hardware information about the node so that a comparison of for instance different runtime behaviour can be further investigated.

A number of observatory modules have been implemented specifically to address microservice artefacts. Thus, each MAO node is capable of scheduling a daily retrieval of metadata from Docker Hub or a weekly download of serverless applications from the AWS Serverless Application Repository, among other types of artefacts. The determination of aggregate values, such as the average number of maintainers per artefact, is part of these modules but can also be implemented as dedicated aggregation modules that are scheduled in succession. Moreover, MAO ensures that all metrics are kept in a timeline so that the evolution of the software artefacts as well as developer/maintainer activities can be tracked over time.

MAO is primarily driven by Zurich University of Applied Sciences, although with increasing participation of interested researchers from other institutions, as evidenced on its website[6]. Available publications cover the framework itself[116, 117] as well as artefact type-specific quantitative assessments, covering Helm charts[118], SAM applications[119], DApps[120], Docker images and Dockerfiles[121].

Renku

The Renku platform consists of RenkuLab, a web-based application and Renku, a command-line tool for managing code, data, workflows and making practical use of the knowledge graph. It focuses on three pillars: reproducibility, reusability and collaboration. A public instance of RenkuLab is available[7] but institutions can also host their own instances and federate them.

Renku workspaces are divided into projects, which are further linked with datasets and interactive environments as well as an integrated GitLab instance in which the Renku project is persisted as a versioned GitLab project (Figure 2.14). Environments such as R Studio, Jupyter Notebooks and interactive shells are available.

The backbone of Renku is the knowledge graph, used to record and query the relationships between data and code. The analysis workflows are linked with a knowledge graph in the following way: 1. Inputs and outputs of analysis steps are recorded into a knowledge graph while the work is being done. 2. These steps can be repeated or integrated into more complex workflows. 3. The provenance of all data products is always accessible via simple tools. 4. Version control is built-in for data, code and workflows.

[6]MAO website: `https://mao-mao-research.github.io/`
[7]RenkuLab website: `https://renkulab.io/`

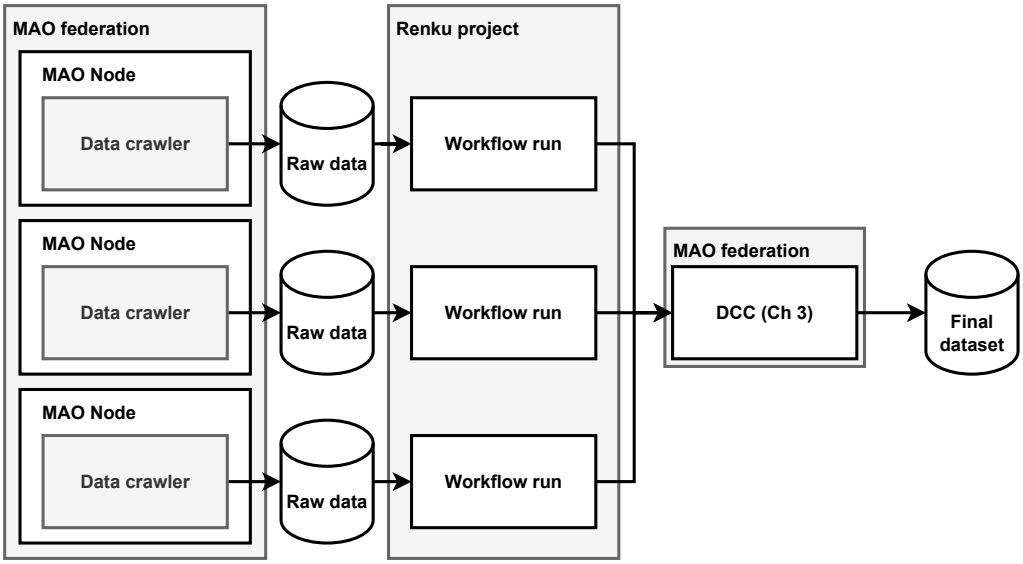

Figure 2.15: Integrated Workflow

Integrated system

There are two alternative designs to integrate MAO with Renku: Running MAO as an interactive environment within Renku, or using Renku's API to store the output data of MAO and perform data aggregation and analysis as a reproducible Renku workflow. We have implemented the latter design due to the loose coupling advantage. MAO's interfacing with a regular Git repository is substituted by interfacing with the repository provided by the configured Renku project. In addition, the MAO orchestrator will use Renku's command-line interface to run the reproducible workflow (Figure 2.15) and update the output datasets.

2.3.3 Illustrative example

We demonstrate the capabilities of MAO with an end-to-end data analysis pipeline. We start with a standard pipeline including data acquisition, aggregation and analysis. Then we outline the conversion of said pipeline to take advantage of the automation features of MAO and Renku.

Standard pipeline

The standard pipeline for this example (Figure 2.16) will include the following stages:

1. **Data acquisition:** Execution of a tool that gathers raw data, such as a downloader for software artefacts and code repositories or a web scraper for software metadata. This can be scheduled periodically with the use of cron jobs or similar tools.
2. **Aggregation:** Once enough data is collected, a user-provided aggregation script will generate a dataset from the raw data, converting them to a usable format for analysis.
3. **Analysis:** The dataset can now be used for analysis in various ways, for example with R scripts or Jupyter notebooks.

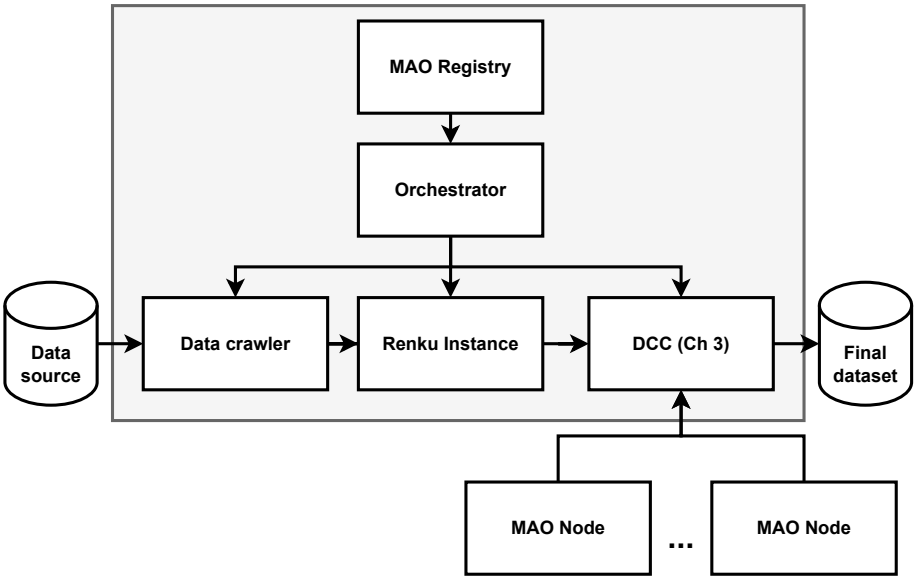

Figure 2.16: Data Pipeline

Converting to a Renku project

The first step to convert this pipeline is to setup a Renku project. The raw data can be used as input data, and the aggregation and analysis scripts can be added to the project. Then, the aggregation and analysis steps of the pipeline can be recorded as a Renku reproducible workflow. This will allow the dataset to be regenerated and the analysis to be updated, whenever new input data is pushed to the repository.

Data acquisition with the MAO orchestrator

The researchers will need to setup a MAO orchestrator instance. While the system can be started as standalone via Docker-Compose, it can also be configured to join an existing MAO federation, in which case it will have access to the shared registry of data acquisition tools (corresponding to stage 1 of the pipeline above) and data sets of the federation (corresponding to Renku projects).

The data acquisition tool will need to be containerised so that the orchestrator can interact with it via the Docker API. After that the tool can be registered into the orchestrator which will add it to the registry, so that other members of the federation can deploy it as well. The new tool can now be scheduled with standard crontab syntax to run periodically via the orchestrator interface.

Renku reproducible workflow

Once the data acquisition is completed, the orchestrator will update the Renku repository with the new input data. Then it can automatically invoke Renku's update functionality and run the previously recorded reproducible workflow, updating both the aggregated dataset and the analysis results.

This means that with every scheduled execution of the data acquisition tool, the entire pipeline will run automatically, from the raw data to the analysis results. The knowledge graph is updated on each run.

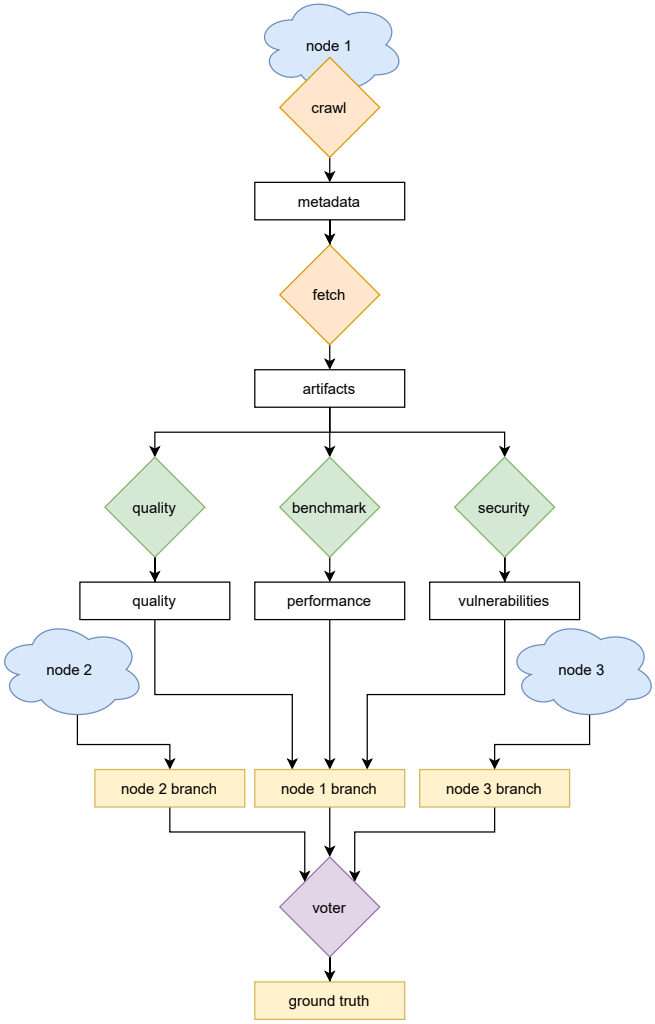

Figure 2.17: Conceptual diagram of the integration

2.3.4 Impact

While our research around MAO maintains the focus on microservices, the underlying framework could also modernise quantitative research on other discrete data points retrieved through the network in regular intervals. This encompasses finance data (e.g. foreign exchange rates or stock tickers), health data (e.g. pandemic cases per administrative region), weather and social networks. We expect that beyond the current focus of funding agencies on the existence of a data management plan, increasingly the emphasis will shift towards highly capable research infrastructures and research data management systems. With MAO and its option to interface with Renku, this concern can be addressed. MAO's containerised tool sharing feature allows data acquisition tools to be reproducible, and experiments to thus be repeated and verified with minimal setup. Renku, on the other hand can handle the data management in a reproducible manner and automate the analysis, granting easy access to replication and verification studies, as well as collaborations. For the software example we discussed above, MAO can automate the acquisition and also allow anyone to reproduce it, while Renku can automate the process of repeating the analysis when new data is acquired, while maintaining data provenance and versioning. In the collaborative example, different institutions can pool aggregate data using MAO from their own monitoring infrastructures

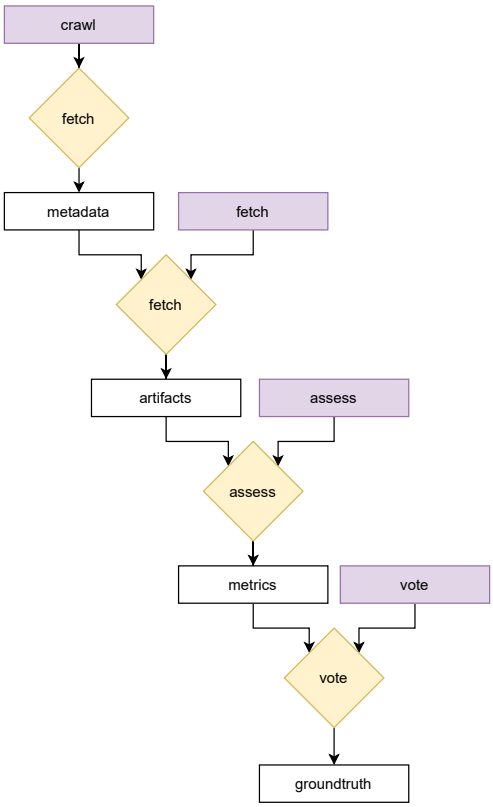

Figure 2.18: Knowledge graph generated by Renku (schematic reproduction)

(e.g. software defects, weather or pandemic-related data) and easily share a Renku workflow to manage and analyse it in the same way.

2.3.5 Conclusions

MAO delivers a federated infrastructure to perform independent quantitative software metrics observations, to foster a shared ground truth by comparing the observations, and to gain insights through reproducible experiments. The reproducibility is ensured by Renku, the usage of which is transparently integrated into MAO for this matter. Software engineering researchers looking for appropriate tools to perform long-term observations, evolution/trend studies and assessments of recently emerging artefact technologies should consider the use of MAO to benefit from these characteristics.

2.4 Summary

In this first chapter, we presented and formalised the data acquisition aspect of the multiperspective observation methodology. We motivated our introductory use-case, the Microservice Artefact Observatory, and presented its application in obtaining data from the public Docker Hub registry. This demonstrates one possible application scenario for our multiperspective observation methodology, as well as being a standalone study of the support for heterogeneous hardware in the Docker ecosystem, and how it can be improved. This work allowed us to assess the capabilities and limitations of our observation framework, and to identify the issues and challenges in producing a data-centric consensus mechanism.

We continued by presenting our work on integrating MAO with the Renku platform, and how this integration can be used to create a reproducible environment for data acquisition. This integration allows the combined system to leverage our observation tools in a platform that ensures reproducible execution and data provenance.

In the next chapters we will focus on the remaining aspect of our methodology, how to reconcile these observations to obtain a common view agreed upon by all observers. We build on our acquisition system presented here, and discuss the issues and considerations in producing a data-centric consensus mechanism.

As we will explain in the following chapters, when running our distributed acquisition systems for prolonged periods of time (e. g., the year-long collection of DockerHub data), faults and inconsistencies tend to appear in the data. These faults can be caused by a variety of reasons, such as software bugs, hardware failures, or network issues. Now that we have concretely presented the systems with which we accumulate data, the next chapters will focus on the reconciliation of these observations, and the production of a common view of the truth among observers. More specifically, Chapter 3 will present our first work on introducing data-centric consensus to our MAO platform and datasets, and Chapter 4 will dive deeper into using voting algorithms to reconcile observations, along with how to apply them to IoT and sensor fusion scenarios.

Chapter 3

Decentralised Data Quality Control for Autonomic Decisions

Until now, we have focused on the data acquisition aspect of this work. The previous chapter detailed the ways we can ensure the reproducible and portable collection of data from multiple nodes, and how this data can be valuable in practice. In this chapter, we introduce the second key component of the multiperspective observation process: How to merge the observation into one combined view, that will form the consensus agreement between nodes. This agreement will serve as the ground truth in further rounds of observations. This is an important reference point when ground truth measurements cannot be obtained in advance, such as when monitoring public datasets or sensors. Yet, a ground truth measurement is necessary to determine the accuracy of the data, so this agreement plays the role of being the best available approximation of the ground truth. We present our Data Centric Consensus (DCC) methodology (Figure 3.1) and the considerations that are involved in designing the process from the integration with data acquisition tools, to the methods of resolving the conflicts that arise when merging the data. We also present an experimental testbed used for evaluation and our experiments to validate the effect that DCC has on system performance and data quality.

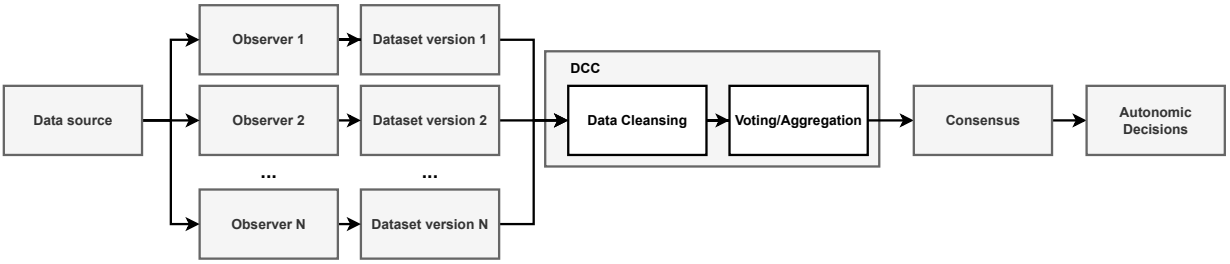

Figure 3.1: Conceptual diagram of our Data-Centric Consensus methodology. Multiple versions of a dataset are combined using a combination of data cleansing techniques and conflict resolution to obtain a consensus agreement on the state of the object under observation.

3.1 Introduction

Humans can be excellent decision-makers in critical situations. They take the context, constraints and possible options, risks and consequences into account to arrive at accountable decisions in a short amount of time. Nevertheless, software-defined processes are increasingly entering into society and often demand faster and more accurate decision-making than

humans can deliver. Both in digital and cyber-physical systems, human operators then merely approve pre-calculated decisions or even opt for completely autonomous system control. This need is exacerbated by AI-based automation, which is intrinsically data-driven and requires a high degree of data quality to be effective. Researchers have pointed out the benefits of autonomic and data-driven decision-making. Examples include future cyber-physical integrated societies[122], ambulance control with critical timing based on clinical data[123], and application placement decisions based on historical quality of service data of machine runs[124] We argue that the prevalence of paradigms like Data-Centric AI[125] and Cooperative IoT will only strengthen this demand.

The essential basis for any decisions are high-quality metrics, where quality is defined in terms of completeness, correctness, precision, accuracy, timeliness and other dimensions[126]. In distributed systems, consistency among multiple, potentially independently conducted observations is another important quality dimension. Severe faults could occur if different system components would base their decisions on diverging local views instead of relying on globally accepted metrics in the form of a consensus-based ground truth.

From a data perspective, current distributed systems reach consensus on the request level[38, 127]. In practice, they collectively agree on the next operation to execute. The data are not under scrutiny on the value level, but rather the ability of the system to arrive at the same overall result. Scaling the granularity down to single values in a dataset of arbitrary size would be inefficient, and indeed is not done in practice, at least not at the scale of entire batches or datasets. This type of value-level consensus is an aspect seen in multi-view data fusion[41], which is typically associated with sensor readings. We have thus identified the unused potential of exploiting the multiplicity of semi-independent nodes in a distributed system to improve data quality, using group assessment methodology to generate value-level granular consensus without the added infrastructure complexity of running a consensus mechanism for each individual value. The main contribution presented in this chapter is a recomputable group assessment workflow, which allows the data generation to be reproduced among nodes and the resulting data to be subjected to a *data-centric consensus (DCC) process* (detailed later) to leverage the redundancy of independent observations, hence reducing deficiencies and improving the overall data quality. This is achieved by comparing the data from each observer, using voting and aggregation algorithms to converge to/agree on a common view, and producing auditable statistics about the process at each step. DCC allows a high degree of transparency, as the degree of success of our method can be verified by a human operator at any point, even if the system does not require such intervention functionally. We argue that systems to produce and manage ground truth become more important because distributed systems with data-driven autonomic behaviour are increasingly deployed in critical domains, such as healthcare, transportation, and smart cities. Consequently, we propose a scalable distributed approach and system design and investigate its concrete benefits in terms of mitigating or correcting deficiencies in data. The system design is applicable to many current behavioural observation problems in distributed software technologies, such as quality control for data-rich domains, for instance as cloud-native software artefacts, which we use as a model use case in our experiments in Section 3.7.

The rest of this chapter is organized as follows. We first give detailed information on data-driven system automation (Section 3.2). Next, we outline common deficiencies in metrics that should support decision-making in Section 3.3. We introduce our federated voting approach and our system design requirements in Section 3.4. The two modules that comprise our system, for reproducible data collection and federated voting-based verification and detailed in Section 3.5 and Section 3.6 respectively. We validate our research experimentally,

presenting our results in Section 3.7. We survey related work in Section 2.2.6, before concluding and discussing future work in Section 2.2.5.

3.2 System Automation and Autonomous Decision Making

Decisions in most complex systems are taken based on rules that encode conditional execution, with conditions referring to metrics. For instance, if the logical conjunction of A and B holds in a situation, where A and B could be metrics-involving equations such as $a < 10$ and $b =$ "*success*", then C is activated. We refer to a and b as metrics that result from direct observation in the form of monitoring, tracing, scanning, service responses or other forms of automated information acquisition. Further metrics may result from model-based predictions[128], aggregations[129] and other form of metrics processing.

When all observations are performed by a single entity, consensus is implied. However, scalable, robust and reliable system design in general calls for diversified and redundant availability of all critical system parts[130], including distributed observers[131].

Due to the need for these parts to operate independently for increased reliability, a decentralised observation model[132] emerges.

The background for this work is focused on autonomic systems. We define these as systems that rely on a continuous stream of data that they use to make critical decisions. These decisions affect the behaviour of the entire system. Thus, systems like this are highly reliant on the quality of the data present, and thus a decentralised observation model with multi-authority verification can increase the reliability of their operation.

Examples of performing critical runtime decisions based on decentralised observation results include: *(1)* Probabilistic macro observations leading to decentralised decision-making in deployments of multiple robots[133]. This technique helps to overcome uncertainty in local observation models; *(2)* Intelligent operation in Cloud and Fog environments through Prometheus monitoring of instrumented applications, inference and prediction in machine learning contexts[134]; *(3)* Automatic blocking of container[135] or VM images in deployment pipelines upon the presence of security vulnerabilities in these artefacts. A policy-driven approach toward this goal is CIPolicE[136]. This work shows the drawbacks induced by thorough artefact scans, and highlights the operational advantages of a distributed observation infrastructure.

Such observation may have deviations, ranging from chaotic variations within a tolerable range (e. g., when measuring a dynamic property), to systemic deviations caused by system or network faults (e. g., crashes or disconnections). In case of deviations between multiple independently conducted observations, practical consensus algorithms[137] help to achieve a single global view within a bounded time. Specifically, practicality refers to fast convergence to common values to avoid blocking the triggering decision-making processes, or alternatively recording the final view as a dispute and hence requiring contingency plans for decisions. Much of the research on practical consensus algorithms focuses on *system-level failure handling* often requiring *leader election*, such as broken network links or node outages in sampled-data settings in multi-agent systems[138]. Another line of research focuses on *binary admission*, such as peer-to-peer and consortium blockchain voting[139]. At this scale, these simple voting models serve to treat the output as a singular entity, and to verify that the required number of identical responses has been received to establish consensus, without allowing for counter-proposals and compromises.

However, when the output data are complex such as a research dataset, granular verification and alignment down to the single value level (i. e., a rich dataset as the output) would give more flexibility in establishing a common ground truth. In some data management workflows, values can fluctuate within an expected range, thus an agreement among nodes needs to be approximate with a given error margin[140]. In larger datasets, we would need to select the behaviour of the system based on the value being verified. Some values would need a majority agreement, some others need an approximate agreement, and another set will require storing all replicas of the value for redundancy or manual analysis. Furthermore, the error margins may need to be computed in the presence of data from all nodes at runtime. With all those data-derived constraints, infrastructure that would utilise an existing consensus protocol becomes ineffective, as much power and responsibility is then assigned to the 'client' to perform all those verification steps. In addition, one must consider the complexity and cost of infrastructure required to implement existing protocols, and the development effort to adapt existing workflows.

This served as motivation to create a group-based quality control approach to data-centric consensus (DCC), with looser security constraints to negate the need for infrastructure, and granting stakeholders the ability to reuse their existing tools and workflow setups, as consensus is only applied to the outputs. We can thus determine the quality assurance methods required by studying the deficiencies of existing data, as detailed below, and design a DCC mechanism that fits the requirements.

3.3 Deficiencies in Distributed Data

Designing a consensus mechanism in distributed systems requires a structured understanding of the reasons for faults and value-level differences. We consider metrics data to be in need of improvement, when there are entries missing or different from other observations made by peers within the system. We identify the following major reasons for the absence of metrics.

System outage. A regularly scheduled observation does not take place because the system is down at observation time. This is unavoidable in a system that operates over a prolonged period of time but can be resolved (at a cost) with redundancy via geo-distribution and multi-zone/-region operation.

Query or transmission errors. The network is down or the data service (e. g., registry) used for obtaining metrics is not usable. This can cause an observation that depends on public data sources to fail, even though a scheduling system unaware of such condition will still register the attempt as complete.

Software errors. Occasionally the environment of a testbed may be misconfigured or undergo updates that may break compatibility with a dependency of the artefact.

On a similar tone, we identify, and list in the following, a few selected reasons for diverging observations.

Differences in timing. As not all observant clocks can be assumed to be perfectly synchronised, and moreover schedules and frequencies may deviate, even static assessments differ due to evolving subjects under observation.

System differences. In particular for dynamic runtime testing where system-dependent performance metrics are obtained, having two or more observatories with different hardware setups will lead to systematic differences. This can also be an advantage though, as collecting enough metadata on the diverging environments can grant a better view of how software runs on different setups.

Malicious intent. A malicious person may want to disturb measurements by injecting fake metric data. An example of this within our use case would be a client contributing to observations of software quality, while being a stakeholder for some of the software observed. In this sense, they are incentivised to skew the results in favour of their own software.

Measurement differences. Some workflows may have a component in their data that is either too volatile or not fully deterministic[140]. In this case, redundancy of independent reproductions of the measurement offers an improved view of the range of possible results, and whether they can be confidently fit into a particular model.

Given these causes of faults, we can broadly categorize the resulting effect of the data into two main categories, missing data and diverging data. For both cases, we argue that having multiple independently obtained copies of the data can help restore missing values and resolve divergent ones, the main goal we aim to achieve with this work.

It is thus clear that data redundancy and provenance tracking of metric observation workflows grant a plethora of advantages. As a result, we argue that automating the process of sharing a workflow within a federation and then collecting and comparing/reconciling the resulting metrics data can significantly boost the resulting data quality and, in consequence, the ground truth-supported decisions. In the context of a real system, a standard ground truth view is not available. The best possible view of the truth is the consensus view of the observers. This differs from a controlled experiment, in which the ground truth is known beforehand, or provided by an implicitly trusted source. In a real system, if such a source of truth was available, it would be used to obtain the observations itself, and the need for a consensus mechanism would be moot. Thus, our dynamic definition of ground truth is needed with the assumption that the observers make use of the best available information to them, and that the consensus view of the observers is the best possible view of the truth. As such, we use the latest consensus view of the observers as the ground truth for the next round of observations.

Regardless of how concrete observation deficiencies manifest, such a system will allow a collaborative system to multiply the amount of data it can produce and collectively manage. This is helpful in cases where one does not anticipate identical data to be produced with every workflow execution. Generating group consensus and aggregated statistics automatically will help such a collaboration effort within a federation. This functionality can be achieved even in decentralised systems by a DCC mechanism, which we detail next.

3.4 Data-Centric Consensus Approach

We propose to use data-centric majority voting algorithms to achieve a ground truth that is universally agreed on by all participants in a federation.

This differs from other, non-data-centric, forms of consensus voting within cluster registries (e. g., Kubernetes[141] and `etcd`[142]) or blockchains, in that it is performed at the data-value level assuming a previously agreed upon federation of nodes, rather than the request level, as shown in Figure 3.2. DCC allows granular verification of the data without having to rebuild the entire data-generating workflow around a consensus protocol. Thus, our system aims to automate the process of reproducing measurements among peers in the federation and validating the results. As such it aims to detect and attempt to correct data anomalies, which means it cannot replace the aforementioned protocols. It instead applies consensus principles to improve the quality of the resulting data. Figure 3.3 shows an overview

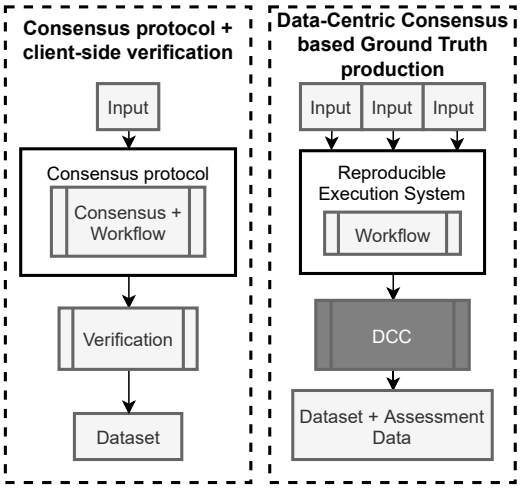

Figure 3.2: Comparison of the proposed approach (right) to state-of-the-art consensus protocols (left). Adapting a state-of-the-art protocol would require modification of the original workflow. Our approach only acts on the data, requiring only the workflow to be reproducible upon its execution.

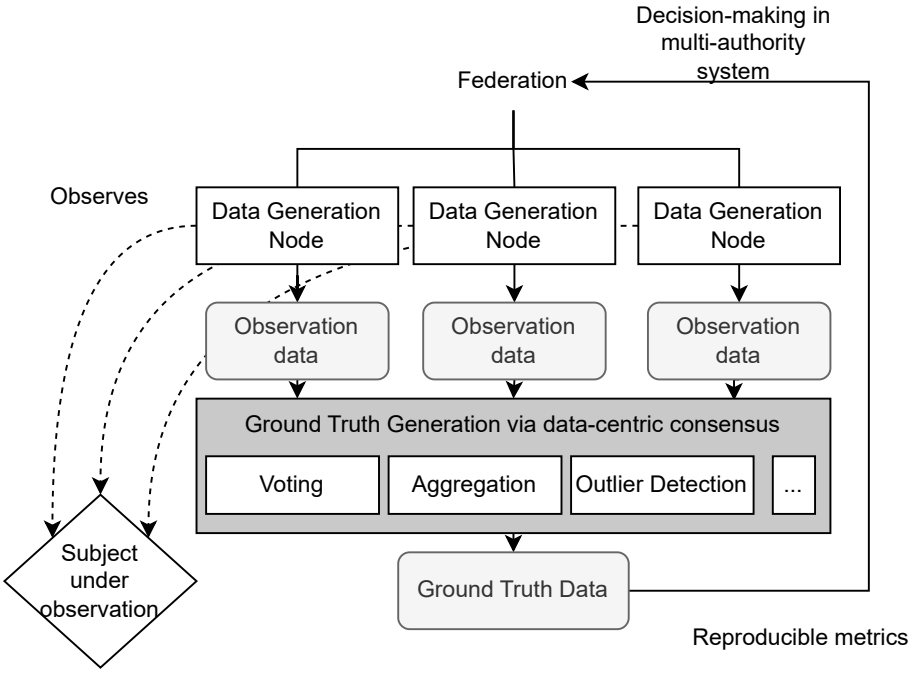

Figure 3.3: Federated observation with DCC overview. Multiple observers collect data on the same target, and the resulting common ground truth can improve the behaviour of the data-driven system that relies on said data.

of our approach. To achieve this, we first define the two essential stages in the operation of such a system.

Data generation stage. This stage includes the generation of data on the individual nodes in the federation. This can be any process that results in data, including but not limited to, web crawling, analysing software artefacts or running an experiment (in our use case, this can be running an artefact in a sandbox). When resolving conflicts, this data is handled differently by data type (numeric, non-numeric, time-series e.t.c.). This is detailed

in Section 3.6. This stage needs to be executed independently among nodes, with each node only sharing enough metadata for others to be able to trace their activity and collect data for the next phase.

Data pooling and generation of ground truth. This stage is performed collaboratively, but to maintain the loose coupling and low infrastructure overhead of the federation, we elect an arbiter and perform the voting centrally. This is described in detail in Section 3.6. The goal is to pool data together to fill any gaps (caused by the failures outlined in Section 3.3 and discover possible data corruption. At the same time, data can be improved upon by combining the findings of nodes (e. g., computing arithmetic means and standard deviation in resource usage among nodes for a runtime experiment).

This two-staged process, as illustrated in Figure 3.3 helps to define the constraints and requirements of such a system. For our prototype implementation (detailed in Section 3.7), we make the assumption that even though the data a node generates cannot be trusted, its intentions can. This is an important distinction, as a hypothetical malicious node can corrupt data, and also skew the result of any of the aforementioned statistical pooling of data, a security issue that when pooling data is non-trivial to resolve.

We illustrate this with the following example. A hypothetical federation is performing an evaluation of a set of software artefacts. If we negate the assumption of trustworthy intention, one of the nodes can be the creator of one of the artefacts and wishes to skew the result in favour of their own software. Knowing how the system operates, they know that the system will filter our maliciously modified data if it strays too far from what the other nodes observe. Therefore this node will stay within the acceptable margin, but still provide modified data, for example stating an execution time that is lower than expected, but not low enough to be caught by an outlier detection mechanism. The only way to detect such an attempt without the use of a system-level standard consensus protocol, would be to audit the workflow code for modifications, which exceeds the scope of this work. For this reason, our system assumes a consortium-like structure, where members join by application and permission. In other words, it is not within the scope of this work to provide Byzantine Fault Tolerance or Fault Avoidance[143], as it would be too complex to implement for a system that runs 'consensus' among data values, rather than requests or system nodes. Beyond this, we define the requirements that the system must meet in Table 3.1.

3.5 Data Generation

Orchestration is the process responsible for managing the activity of all nodes. For this purpose we define an orchestration system that needs to be capable of scheduling and executing portable measurement programs and accessing a shared registry so that each node can communicate its activity to the rest of the system. This is a more specialised use case compared to what existing state-of-the-art container orchestrators, such as Kubernetes[141], Docker Swarm [144] or Titus[145] do. To ensure the maximum amount of independence among nodes, the nodes cannot assign tasks to each other, but rather voluntarily join experiments in order to independently verify or add data to them.

We identify two main components of an orchestration system, a distributed registry and an executor model.

Distributed registry. A distributed registry is shared among all nodes. To maintain independence, the registry will record the activity of each node and include enough information for other nodes to join them. The registry maintains several entry types:

Table 3.1: System requirements

Requirement	**Description**
Reproducible but independent data generation or collection	It should be trivial for a node to be able to retrieve and execute a data collection or experimental workflow created by another federation member
Rich metadata scheme	The system must be able to treat data points differently depending on their type. Comparing runtime of a piece of software among nodes is different from ensuring all nodes have the same version of said software. The metadata needs to inform the consensus system on the type of assessment that needs to be performed. Additionally, runtime information that should be included in the analysis (whether a node is running in a VM or what system resources are available) must be visible in the metadata
Data-centric consensus for verification and augmentation	The main reason to implement such a system is to collect and verify data from independent replications of a measurement/experiment. Thus, it needs to implement an algorithm or set of algorithms to combine findings, discover discrepancies when possible, and determine the integrity of data
Full transparency of decisions made by the system	The purpose of the system is to ease provenance and redundancy and enhance collaboration. In this effort it will make decisions automatically that would typically be made by an independent researcher attempting to reproduce an experiment. We recognise that no system, however advanced can fully replace the human observer, thus a detailed report of all decisions to discard data as outliers or corrupt values is produced in both machine and human-readable format and also includes statistics on all data that were retained, such as confidence intervals and standard deviations
Propagation of resulting ground truth	The result of the aggregation needs to be made known to all nodes in the system. This can be achieved in a number of ways (public repository, shared database, blockchain, etc). Implementation details can vary by application domain

1. Reference to workflow code. This entry needs to contain enough information for a node to access the code to a workflow and execute it. Specially designed containers can aid this effort due to their portability. For the container case, a link to a publicly available container may suffice for the orchestrator to execute it. In case special requirements or parameters are needed by a container, the entry needs to specify them so that the system can execute it.
2. Reference to required input data or generated output data. Datasets needed as inputs for a workflow need to be listed and accessible in the registry. Data produced by the nodes must be findable in much the same way. How this is implemented can vary wildly by implementation domain. Datasets could be available as public repositories, such

that the registry only needs to link them. Alternatively, there may be an underlying private data federation among the nodes, in which case the registry could be part of a larger database. Data must be linked back to the node that produced it and the tool that was used to obtain/create it. This information is needed by the DCC mechanism.

3. Metadata related to the consensus system. Depending on the algorithm used, nodes may need additional information on each other.

Orchestrator and executor. This component schedules and runs the workflow code as needed. It needs to be able to run portable experiment artefacts and record the necessary runtime information. Moreover, it needs to correctly arrange shared data (e. g., pre-processed datasets) in such a way that the consensus module can determine their relationship to the ground truth.

The primary focus of the orchestration and execution module is to ensure that workflows are executed by nodes with the minimum possible overhead. A primary design goal to that end was to allow existing workflows to function as they would in a standalone environment. As a result, the format needed for the DCC system is relatively simple and can be achieved even in the absence of an orchestrator module. This advantage is reduced in the presence of volatile input conditions (e. g., using a public data source as input) as the orchestration system becomes responsible for ensuring the input is identical among nodes. This is a constraint not present in classical consensus mechanisms as the entire process of serving a request is subjected to the mechanism. In the considered scenarios, the process and its output are too complex to be treated as a single unit (i. e., it would require unrealistic development time to adapt a data science workflow to adhere to a computation model where each value paired to each input is subjected to a consensus algorithm to produce output), hence the decision to reconcile outputs after each step to verify the assumption that the process is reproducible. As a consequence, subsequent observations can use prior agreed-upon input data to produce further output down the line.

3.6 Ground Truth Generation in DCC

We define the family of processes a system such as ours would use to combine and compare datasets as the DCC process. The process of generating the combined dataset consists of the following steps: *(1)* Data pooling (see Section 3.6.1), where the collection of independent measurements and experiments are stored in a location accessible to all peers; *(2)* Leader election (see Section 3.6.2), where it is decided which node will run the combination step; *(3)* Data-Centric Consensus (see Section 3.6.3), in which the collaborative ground truth dataset is generated; and finally *(4)* Propagation (see Section 3.6.4), placing the generated ground truth dataset in a location available to all nodes or (optionally) to the public.

3.6.1 Data Pooling

Data Pooling is the process of collecting node data to initiate the consensus process. Nodes may submit their data either as full datasets, i.e. the full output of an experiment or as updates to the already existing ground truth from a previous consensus round. This has no effect on how the system will resolve conflicts, as it only affects the volume of new data that will be subjected to the consensus process. Therefore, the choice of how data is submitted is up to the implementation specifics.

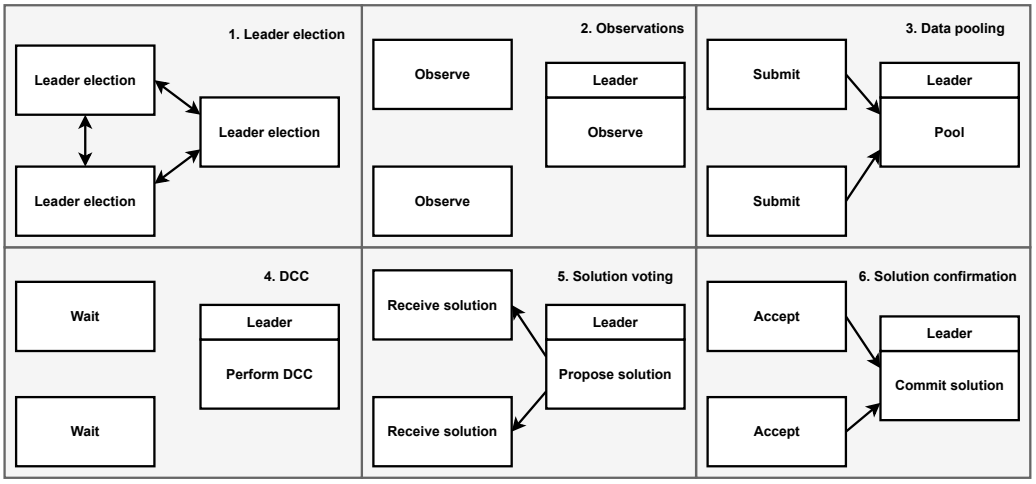

Figure 3.4: DCC workflow with leader election consensus algorithm

3.6.2 Choosing the Arbiter

To perform the voting, the federation will need a temporary arbiter node. Depending of the implementation, this arbiter might perform the entire consensus process, or act as a coordinator, distributing tasks to the nodes.

The arbiter can be selected manually, by the operator triggering the vote. This is the implementation we used for evaluation, to make the same node conduct the DCC process every time and make the experiments more controllable. However, it is also viable to choose a leader-election algorithm, such as Raft[146] to rotate the arbiter role among the nodes, an implementation more suited to production systems. A reputation system, based on past performance in consensus rounds is also an approach for selecting an arbiter, using an approach similar to a recent work in consensus protocols[32]. Since the consensus process is centralised, the choice does not affect the procedure, but in a production system, being able to rotate the arbiter can be beneficial for robustness and security. A sample of how the DCC workflow would work alongside a standard leader election-based consensus protocol can be seen in Figure 3.4. The arbiter is chosen using a consensus protocol, then DCC is performed by the arbiter, and finally the result of DCC can be further validated by the follower nodes. It is thus important to note that though a 'consensus' among nodes is a core DCC concept, it is separate from the concept of consensus in a standard consensus algorithm, as illustrated by this example.

3.6.3 Consensus Generation

The assessment procedure is the core of the process. It is performed in 4 distinct phases. To ease the explanation, we define the following:

Duplicated row. A piece of a joined dataset comprised of rows with identical indices. Each row is submitted by a different node. This is the sub-structure that needs to be merged.

Merged row. The result of merging a duplicated row.

Cluster turnout. A property of the duplicated row, indicating the number of nodes that submitted this row for voting. It affects the options we have when it comes to statistical methods used to resolve conflicts.

We also differentiate between two data types:

Measurement.A numeric value, usually subject to errors.

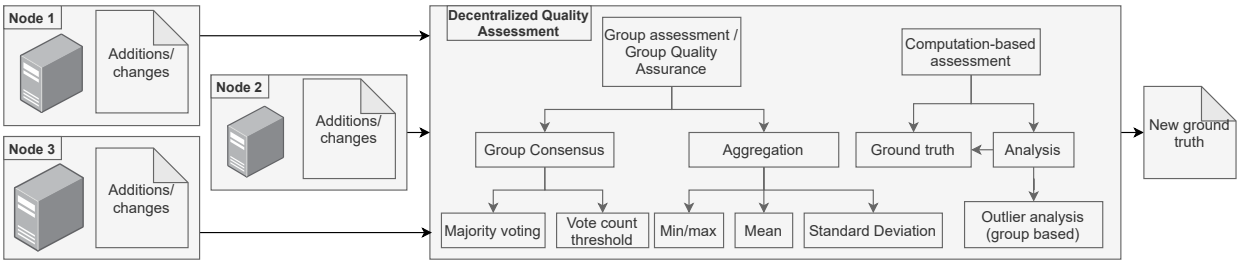

Figure 3.5: Map of assessment methods and techniques

Property.A non-numeric value, where agreement between the nodes requires a selection of the correct value.

The DCC phases are as follows (as summarized in Algorithm 1):

1. Collection of submissions. All data additions/alterations are collected by the leader from all contributing nodes.
2. Selection of changes. Rows that have not been submitted as additions or alterations by the majority of contributed nodes are discarded and remain as per the ground truth. Rows to be added or updated that were submitted by the majority of the nodes advance to the next phase.
3. Conflict resolution. This is detailed below.
4. Ground truth generation. The merged rows are combined to produce the new ground truth dataframe.

Algorithm 1: Ground truth production

Data : N datasets
Result : Ground truth dataset
concatenate *datasets* into $dataframe$;
for *unique index in dataframe* **do**
  if *index appears (N/2+1) times in dataframe* **then**
    add all $dataframe[index]$ to $duplicatedentries[index]$;
  else
    discard all $dataframe[index]$
  end
end
for *index, in duplicatedentries* **do**
  $mergedrows[index] \leftarrow resolve(duplicatedentries[index])$;
end
combine $mergedrows$ into $Groundtruthdataset$;

Each column of a duplicated row is assessed separately depending on whether it is a property or measurement. These are categorised by accompanying metadata. The cluster turnout also influences the options available for statistics and naturally their accuracy. We argue that even though a large sample is difficult to obtain (since in this context each point is contributed by an independent node), advantages in data quality can be observed even with the minimum amount of nodes, as incomplete data can be filled in and extreme outlier

measurements can be identified. This will be further discussed in the evaluation section, as we present our experimental findings.

The possible methods to resolve a conflict are presented below and illustrated in Figure 3.5:

Aggregation. This method is applied to a low-population sample (less than 10 nodes) of a measurement. Mean, median, min, max and sigma values can be calculated. Outlier detection is not reliable at this scale, though potential outliers can still be flagged for manual inspection.

Outlier detection and aggregation. Outlier detection (e. g., the z-score[147]) can be computed reliably (as shown later) for samples larger than 10 nodes. Barring malicious faults this method can produce a result identical to approximate agreement[140].

Clustering majority[148]. We can use ML-based clustering algorithms (i. e., Meanshift[149]) to cluster the data and then perform majority voting on the resulting clusters.

Majority or plurality voting. In cases of values that are expected to be identical, we have observed missing data or partial corruption as the only culprits for deviations, given we assume a lack of malicious nodes. Thus for low populations (3–9 nodes) we can employ majority voting to detect the correct value. For a larger federation, we can require a high percentage of votes to be for the same value.

3.6.4 Propagation

Propagation of the ground truth is entirely application-specific, and thus mostly out of scope for this work. Nevertheless, all nodes need to have knowledge of the new ground truth, so they can submit their future changes. This step can only be omitted when using a stateless strategy, that is, a node submits all of its data instead of the delta. The generated data can be then turned into a distributed knowledge base, either public (e. g., github or published dataset) or private (e. g., distributed database or consortium blockchain).

3.6.5 Module Design

Our consensus module prototype was developed by selecting the best candidate strategy for each of those steps, according to our use case. It was also designed to interoperate with the reproducible orchestrator setup we have developed previously, however it is modular enough to be used with any software that enables reproducible data workflows, as long as:

Repository structure is compatible: Our module supports peer data being arranged as either branches in a git repository, or folders in the local filesystem.

Metadata schema: The repository must contain a metadata file that outlines which files need to be subjected to the consensus algorithm and how each field in the data needs to be treated, according to the consensus generation strategies mentioned above. A sample of this schema is given in Listing 3.1.

3.7 Evaluation

Replaying captured long-term observations under the influence of artificial metric changes is used to validate the voting. First, we describe our experimental setup and well-known drawbacks from two real-world datasets from the area of cloud software engineering. From this analysis, we derive test data with which to evaluate the DCC techniques.

Listing 3.1: Metadata schema for the consensus module.

```
{[
  {
    "file": "package_info.csv",
    "index": "package_id",
    "majority": ["version", "language"],
    "approximate": ["mem_usage"],
    "keep_all": ["deploy_time", "cpu_usage"]
  }
]}
```

Figure 3.6: Metadata schema for the consensus module

3.7.1 Experimental Setting

We implemented a working prototype of the orchestrator/executor module and deployed it to two test nodes which are VMs within an OpenStack cluster and part of the same network (latency of approx. 2ms among nodes), which had been collecting data for various other experiments[150]. The prototype (Source code: https://github.com/serviceprototypinglab/mao-orchestrator) is implemented in Python and orchestrates the execution of data science tools implemented as Docker containers. With data from this test deployment, we investigated discrepancies between the two nodes and analysed some of the properties of the data to better understand and demonstrate the needs of our use case (Figure 3.7). Each node maintained its own data branch (implemented as a git repository) and all nodes registered their branches in the registry, so that the consensus module could locate them. Based on this data, we constructed multiple test datasets that we used to evaluate the capabilities of the consensus module. Automatic interoperation between the orchestrator and consensus modules has not been implemented at this time, so we triggered the consensus step remotely. The consensus module retrieves the git branches from each node, runs the ground truth algorithm, and commits the ground truth to the master branch for the dataset repository.

3.7.2 Deficiencies in Data

To create a point of reference in creating test data we used long-term software observations retrieved from the Microservice Artefact Observatory (MAO)[151]. In particular, there are two datasets (summarized in Table 3.2) obtained from monitoring different public data sources over time across two nodes. First, a dataset of metadata from a sample of 1220 Dockerhub repositories[12]. The dataset contains information on Docker container images compiled from 434 daily snapshots. Second, a dataset of metadata from Artifacthub[16] referring to higher-level cloud application orchestrations including Helm charts and Kubernetes operators, compiled over 142 days of observation.

To obtain a view of the data deficiencies encountered, we calculated the rate of change each day for the same node in these datasets, and then the differences between the two nodes. The two time-series-like datasets show a high rate of change from day to day, with measured values changing typically 10-15% per day as shown in Figure 3.8. In order to obtain meaningful DCC among nodes in the case of monitoring data from a public source, some form of scheme to make the input common among nodes or a way to timestamp the

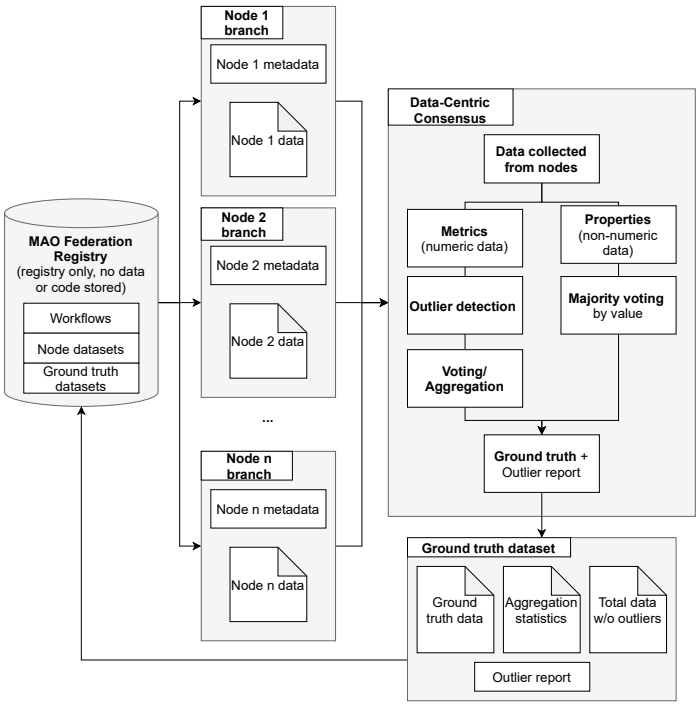

Figure 3.7: Consensus module testbed design. The observer nodes share access to the workflow and data via a shared registry. The consensus module collects the data submissions and creates the combined view, as well as statistics to help a human auditor evaluate the effectiveness of the consensus process.

Table 3.2: Reference datasets

Dataset	# **files**	# **rows**	# **cols**	**file size**	**index**
dockerhub general	1	434	12	32kB	date
dockerhub detail	434	1220	6	38.7kB	repository
artifacthub general	1	142	5	4.5kB	date
artifacthub detail	142	2227	6	76.1kB	package

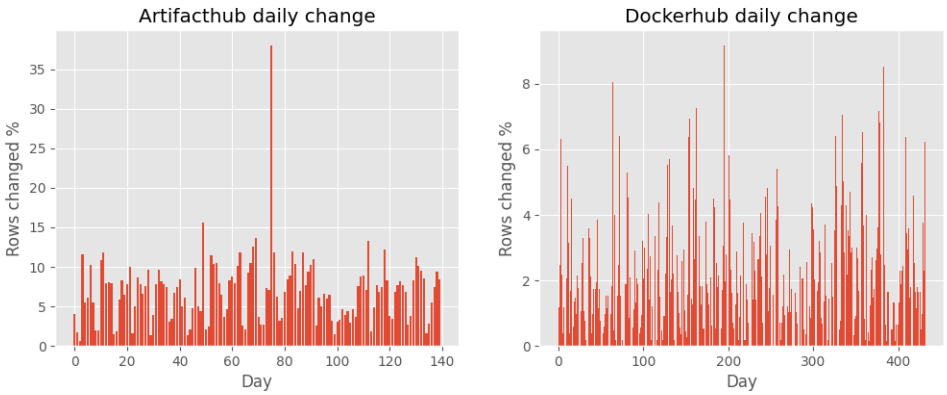

Figure 3.8: Daily change in reference datasets. These results highlight the importance of factoring in the volatility of public data sources when merging multi-observer data.

data and establish causal relationships is required. It is the responsibility of the orchestrator infrastructure and depends on the type of experiment conducted. The artefact property datasets present interesting elements of discussion. By comparing each snapshot with the one immediately preceding it (by 24 hours each time) we observe the following:

For the Docker dataset an average of 1.86% of repositories were updated each day (max: 8.64%, standard deviation σ: 1.58%). For Artifacthub, an average of 8.08% of packages were updated each day (including the addition of new ones) (max: 39.88%, σ: 6.60%). This can be of relevance to stateful consensus strategies, as a node can compare the current snapshot with the existing ground truth and only submit the differences, reducing the volume of data that needs to be assessed by up to an order of magnitude.

Next, we look for missing data and differences in values at common indices between the nodes. For the Docker dataset we encountered 8 cases where one of the two nodes failed to execute on a particular day, while the other succeeded. On the remaining dates where both nodes retrieved data, there were 18 days where at least 1 row (representing a repository) differed, and the biggest difference observed was 2 rows. For the Artifacthub dataset, there were a combined 27 failures between the two nodes. However, only 2 of the days when both nodes ran (out of 113) had any differences in data, with a maximum difference of 6 rows.

3.7.3 Synthetic Test Data

Using our real datasets as a set of guidelines we devised the following test datasets:

1. Number of entries: Varying the number of entries in the data from 200 to 2000 in increments of 200
2. Number of columns: Varying the (non-index) columns from 1 to 10, 2000 entries
3. Ratio of numeric properties to non-numeric properties from 0/10 to 10/0, 2000 entries, 10 columns
4. Cluster turnout (number of simulated nodes) from 3 to 21 in increments of 2, 2000 entries, 10 columns, all measurements

Finally, we created a dataset of 10000 identical values into which we can inject randomized errors, in order to assess the fault tolerance of our approach.

3.7.4 Evaluation of Data-Centric Consensus Techniques

Properties

Reaching consensus on data values requires different techniques for different types of data. In the simplest of cases, we have observations that we expect to be identical among nodes, as they represent fixed information, called *properties* in the following. These are data values that cannot be subjected to numeric operations or comparisons, such as characters and strings. Obtaining consensus on properties is a matter of majority voting among the federation nodes. This can be augmented by expecting a given majority threshold, or applying weights on votes based on past performance.

Measurements

For measurements, DCC requires more attention. Here, we expect no two nodes to offer the exact same value. While clustering is a common choice in the state of the art[152], our scenario has some peculiarities that make the selection non-trivial. The first constraint is the population of each sample. In our case, a single data cell can be flagged as a conflict among nodes and evaluated using the consensus technique. As a result, the corresponding data size will be equal to the population of the federation. Since the nodes are independently operated, we expect the populations to be low (generally lower than the 21 members in the virtual test data defined in 3.7.2). While this means each conflict resolution will be fast ($\sim$2 seconds) due to the small samples, there is the potential of executing a large number of conflict resolutions (as many as 20000 values than can potentially be in conflict in our test data). Each one would have to be clustered separately, creating a large performance bottleneck.

For a subset of datasets with few measurements with a large degree of variability, this can be vastly improved by performing a conflict detection pass on the data, similar to what we did for 3.7.2. As per our observations based on our real data, less than 1% of the data are in conflict between nodes, meaning this pass can reduce the number of clustering steps by two orders of magnitude. However, all of our reference data are observations, and thus we expect little conflicts. In datasets with a high amount of measurements, potentially all of the data will be in conflict among nodes.

Because of the restrictions of the use case, we can make additional assumptions to speed up the consensus generation. Due to the way data are processed, all observations are treated as 1-dimensional. Higher dimensional data can be processed at the node level, but when making simple comparisons, the system can afford to be blind to the relationships between columns. Additionally, as discussed, the sample size is small, as it is tied to cluster population.

Based on these constraints, we can simplify the clustering consensus by applying the assumption that only one valid cluster of measurements exists, (i. e., 1-dimensional), and all data outside of it are outliers[153]. We can thus simplify the clustering consensus, by filtering out outlier values and aggregating the remaining values. Outlier detection can be performed using a method such as calculating a z-Score for each value. This is already a more involved end step than what is done with approximate agreement[140] systems though with the naive assumption that input correctness is guaranteed by another system component, as discussed above.

We recognise that this method would become invalid for much larger clusters with hundreds of nodes, which however we esteem not to be realistic for real-world federations of independent nodes. The inverse limitation can also be investigated, i. e., a cluster with

too few members. It can be proven mathematically that z-Score outlier detection cannot be performed on a sample size of less than 11: Making the example assumption of 1 outlier $k + e$ and n-1 identical values k and calculating the mean μ and standard deviation σ we can then substitute in the formula for the 3 Sigma rule:

$$\mu + 3\sigma \leq X \leq \mu - 3\sigma$$

which solved for n yields $n > 10$ which means we need 11 values to identify an outlier. With such a small sample size, however, clustering may also not detect outlier values correctly (though in a best-case scenario with identical values and a single outlier it will return a correct result even with a sample size of 3). We resolve this problem by including as many statistics as possible and preserving all 'candidate' values, rather than relying on the 'consensus' value exclusively, leveraging our policy of a transparent user-space consensus generation procedure.

Meta-Conflicts

A third type of conflict applies to situations that cannot be resolved by the data alone. For instance, performance data for a software artefact running in a sandbox environment. Nodes can record this data and include metadata on their runtime environment and hardware configuration. However, data are not directly comparable and thus should not be subjected to any automated consensus procedure. Our system allows for such data to be flagged on the dataset's metadata so the consensus module ignores it and instead only writes it on the pre-consensus combination file (a concatenation of all node observations that includes a node's identifier for each row) so that custom analysis can be performed as needed.

3.7.5 Consensus Generation Testing

For testing purposes, we implemented the following for the ground truth generation. The consensus module first retrieved the data from all node branches. Then it merged the datasets and discarded any entries that did not appear at least N/2 + 1 times (where N is the number of nodes). Then, for each group of rows that share the same index, it resolved disagreements, with the following strategy.

If a value is non-numeric, it is resolved by a vote, where the value submitted by the most node is selected. The rest are marked as discarded. For numeric values, the standard deviation is calculated, and values outside of the 3 Sigma rule are marked as discarded. The rest are averaged, and the mean value is selected as the ground truth for the value. Additionally, the standard deviation, minimum and maximum value among those not discarded are also saved. All results are then compiled into the ground truth dataset, with the same format as the original, including the newly added columns to store such statistical values. Finally, a list of discarded values with the index, row, column, value and the node ID that submitted this value is written. All results are then committed to the master branch of the dataset as described in Section 3.7.1.

3.7.6 Consensus Generation Performance

We evaluated the performance of the selected DCC strategy using our test data. We tested the performance using both local data, to isolate the DCC procedure, and by placing the synthetic data in our real data generation nodes, to test in real network conditions. The nodes used were running the same data collection software used to obtain the real-world

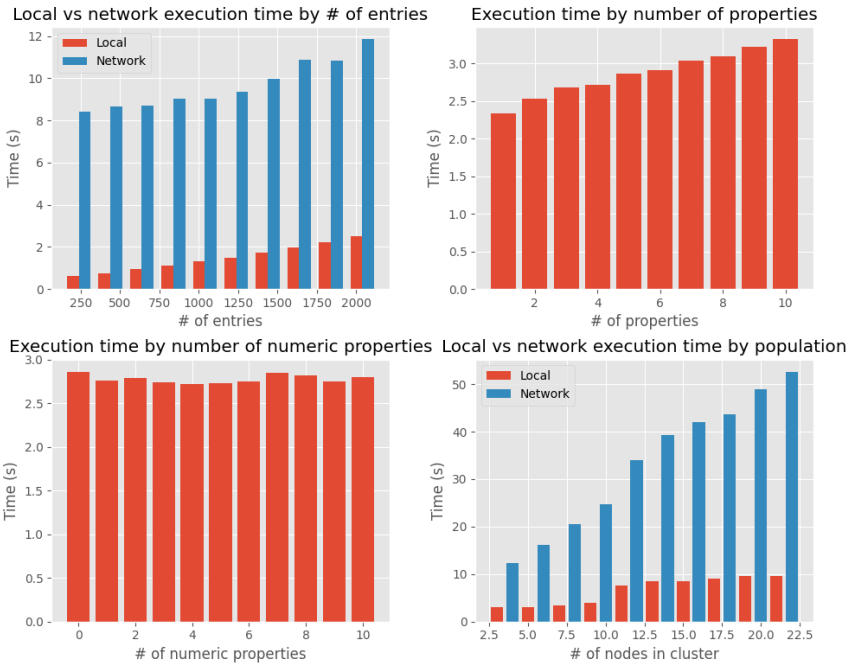

Figure 3.9: Performance of the consensus module for various test data. We measure the execution time while varying dataset size, data dimensions, number of numeric properties and number of voters. In the first and last scenario we also test with remote observer nodes to compare the performance of the system to the network overhead in real conditions.

reference data referred to in 3.7.2. We conducted four different experiments to evaluate the impact of different parameters on the overall performance of the solution (Figure 3.9). For this work, we did not include the performance of the distributed data generation step, as the data generation setup used is implementation-specific, and the DCC process was our object of observation.

Number of entries

The first experiment varied the number of simulated artefacts in the test dataset, as described above. All datasets were restricted to one index column and one data column. We averaged the execution time over 10 executions each for the local and remote scenarios. We observe near-linear scaling here, with network overhead taking up the majority of time in the networked scenario.

Number of properties

Next we measured performance for 2000 simulated artefacts, this time varying the properties from 1 to 10. Once again we observe linear scaling.

Ratio of numeric vs non-numeric properties

For the third experiment we examined the performance for different ratios of properties to measurements. We started with 10 non-numeric and 0 numeric properties and kept increasing the numeric properties by 1 and reducing the non-numeric properties by 1 for each

iteration until there were 10 numeric and 0 non-numeric. We did not observe a difference in performance between the different data ratios.

Number of nodes

For this experiment we used the 2000 entry, 10 property dataset and this time varied the number of nodes between 3 and 21 in increments of 2. This experiment was also carried out over the network to measure the difference in overhead as the number of node branches varies. Both experiments exhibited linear scaling, though the network was significantly steeper in this case, as the data transfer was overshadowed by the overhead of retrieving and arranging multiple datasets. A noticeable jump in execution time ($\sim$3 seconds) is noticeable between 9 and 11 nodes. That is because this is when the z-Score outlier detection becomes enabled. This would not be observed for properties, as they are only subjected to majority voting.

3.7.7 Data Quality Testing

Next, we compared the correctness and performance of different techniques for data verification. For generic data such as this, voting algorithms, and more particularly result amalgamation algorithms[154] are often employed (when the data is too small to permit an ML approach). We compared our own method of detecting outliers and then averaging the remaining values with the still commonly used method of obtaining a weighted average, variations of which are widely used in practice. Additionally, we added a method based on the concept of Byzantine Approximate Agreement[155] to the comparison. Though this method is not commonly used, it is interesting to test its performance since, as we mention above, our system can implement any number of algorithms allowing the user to select.

To run the comparison we ran an error injection experiment that sets a correct value and injects randomized errors up to a given error amplitude. In addition, we varied the occurrence rate of errors to simulate environments where errors might be rare or ubiquitous. The results presented correspond to varying the error rate between 0 and 100% for an error amplitude of 0.3 (as a ratio of the correct value) with a step of 2% (10000 rounds of voting each step), varying the error amplitude between 0.1 and 1 for an error rate of 4, 40 and 100% (selected to better illustrate the behaviour of the algorithms in different conditions). In all cases each round of voting had 11 variant values as candidates (we explain above that this is the threshold for gaining an advantage when doing outlier detection). A voting result was accepted as correct if it was within $\pm$1% of the selected correct value, which means the error was corrected acceptably in the output data. As can be seen in Figure 3.10, our augmented approach outperforms the state-of-the-art weighted average algorithm when the error rate is $<$50%, and matches it beyond that point. The Byzantine voting algorithm has the more correct results when errors are $<$50%, but one should consider that the other two will always return a best effort value, even if it does not meet the $\pm$1% standard of correctness, while the Byzantine will drop values that do not meet the standard, resulting in loss of data when the data is noisy.

3.8 Related Work

There has been a significant body of previous work in automating some aspects of distributed data management. Use of other federated systems[156] or peer-to-peer networks[157] for

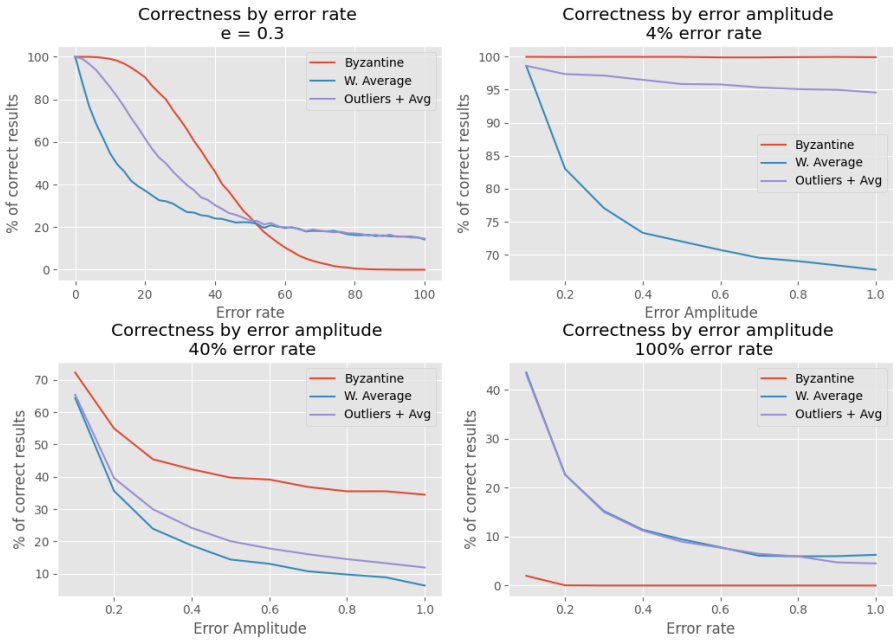

Figure 3.10: Data correctness comparison between 3 algorithms for resolving conflicts via voting. We measure the percentage of correct results for a given error amplitude by varying the frequency of errors, then vary the error amplitude to measure correctness at 3 key error rate values. Adding an outlier removal step improves the weighted average method for low error rates without compromising its performance on noisy data. Grouping the values and testing for a Byzantine agreement is effective at low error rates but becomes unusable for noisy data.

data management exists in prior practice, though with different practical architectures, implemented to achieve different goals.

Closer to our methodology, consensus algorithms are utilised extensively in blockchain systems[158] and indeed some methods employed in private blockchains[159], such as Proof-of-Majority consensus [160] may appear to overlap in some semantics. However, the intention of our DCC mechanism is not to guarantee the integrity of the data, but rather to select or generate an agreed-upon truth among several peer contributors that offer data. Thus, we argue our approach offers a different benefit, that could be complementary to a private blockchain, rather than being a directly comparable alternative solution.

In fact, there are possible ways our consensus system and a consortium blockchain [161] could interoperate and complement each other. As an example, we mention that our system does not take into account malicious data modification, a problem that has long been solved for blockchain systems, for example granting Byzantine Fault Tolerance[162]. Thus our granular verification of the data could act as an end step for a consensus protocol-secured process.

In terms of our conflict resolution methodology, it is partially derived from majority-based group consensus approaches typically used in crowd-sourced tasks[32], related to quality control.[6] Aggregation of results is also a method that is applied in such scenarios selectively. Our use-case is similar, though much smaller in scale, though with the distinction of using equally trusted participants and with the goal of data quality control and lineage tracing through redundancy. The scenario closely mirrors the concept of N-Module Redundancy, for which voting algorithms are commonly used to ensure reliability. Several algorithms exist to serve different scenarios [154], with variations of the weighted average being common for generic datasets [163] [164]. In addition, the concept of Approximate Agreement [155] can be leveraged for modifying selection voting algorithms to work with noisy data.

Finally, the emerging field of Data-Centric AI is now driving the adoption of data cleaning and augmentation techniques [125], which aligns with our goals of improving data quality.

3.9 Discussion

In this chapter we presented a methodology for generating data-centric consensus in a federation of peers. The method consists of sharing reproducible workflows in the form of containerised artefacts, pooling data and performing various conflict resolution techniques such as group consensus with majority voting, outlier analysis and statistical aggregation. We described a prototype implementation of the consensus system and the data pipeline used and described the requirements for a compliant orchestration/execution system. We validated the design by applying our consensus module to our existing real-world data as well as test data designed to emulate different datasets.

We believe that this work will ease collaboration among distributed teams from different research institutions and promote reproducible research. In particular, tools that comply with our system automatically comply with two of the standards defined by the ACM for Artifact Review and Badging[3]. The tools will be available both as images and public code repositories and registered in the federation registry (potentially published outside of it in a marketplace or similar portal), thus qualifying for the Artifacts Available badge. Additionally, the orchestrator combined with the consensus system allow observations using

the tools to be reproduced and then verified, thus arguably complying with the Results Reproduced badge.

We recognise some limitations and ongoing challenges in this approach, some of which have candidate solutions but are nonetheless non-trivial to solve. Perhaps the most important assumption the consensus system makes, is that the infrastructure before it (meaning the combination of orchestration and execution systems and the tooling they manage) ensures the data are comparable. However in some instances that is non-trivial to guarantee. For example when retrieving data from a public source, the time of retrieval is important information as a timing difference between the data retrievals for two nodes can render the data of one incomparable to the data of the other. Another example is when data depends not only on the software stack of the framework but also on the underlying configuration of the node itself (hardware, operating system, whether or not the node is a VM etc). Runtime metrics are especially sensitive to such parameters. Another limitation is the lack of protection against malicious data skewing. As mentioned before, extreme cases of data manipulation would trigger the outlier detection and be removed from the data, but subtle skewing would not. This can be achieved by running modified tools in a workflow to produce data that will skew consensus in one direction. Since consensus is only performed on the resulting data, an additional security layer would need to be in place for this type of attack, such as verification of code signatures by the orchestration/execution modules.

3.10 Summary

In this chapter, we investigated more concretely the methods and systems needed to merge and combine datasets obtained from multiple observers. We started by investigating the challenges related to data quality faced by a multiperspective system.

We then introduced our DCC strategy, an end-to-end approach for data management in a multiperspective environment. We presented the different stages of the data lifecycle, from the generation of data to the aggregation and consensus production. We then presented a model implementation using MAO.

Our methodology was evaluated using a combination of data collected by a federation of MAO nodes, as well as synthetic data created to stress the system and measure the impact of our procedure on performance. Finally, we evaluated the effect of data quality by running an error injection experiment, and measuring the system's ability to reach consensus on the correct value in the presence of errors of varying magnitude and frequency.

Overall, we observed that our DCC strategy does not have a significant impact on performance, and that its error correction capabilities are sufficient to reach a consensus in the presence of errors.

Chapter 4

Software-Defined Voters

In the previous chapter we set the stage for the process of assembling a multiperspective system based on our Data-Centric Consensus (DCC) methodology. We introduced the concept of DCC, along with architectural considerations for collecting observations in batches and performing voting. In this chapter we focus more on the voting process itself, with the introduction of our Voting Definition eXtended (VDX), a system used to create software-defined voters for any kind of application. We designed VDX to bring together all previously mentioned techniques related to voting, so it would work on both batches of data, as well as real-time sensor readings. As we focused on batch processing before, in this chapter we present a new application domain, that of IoT. Here, we have the constraint that consensus needs to be performed in real-time, and voting rounds are performed continuously. We examine how these constraints affect our design choices and how well our overarching strategy adapts to this domain.

4.1 Introduction

Automated decision-making based on interpretations of data from the real world[165, 166] is a growing trend across societies, especially in digitally transformed economies and legislations. This trend entails a number of ethical concerns around mispredictions and wrong decisions, economic trade-offs related to investment into data acquisition equipment and maintenance, as well as technical challenges associated with ensuring the proper quality of the acquired data. Data cleaning techniques[167] are commonly used on a single measurement source, on a single time series to mitigate device downtimes and transmission errors, as well as to eliminate unsuitable data records, effectively increasing the data quality. These *ex-post* cleansing techniques can however not prevent wrong observations that result from combining multiple sources with potentially diverging or conflicting views. In addition to the above techniques that are applied to each individual data stream, we need to perform multi-view data fusion[41] to arrive at a common ground truth across multiple data sources beforehand, especially in the case of redundant sensors. In this context, we employ voting-based data fusion to merge observations from multiple sensors that, in ideal conditions, would produce identical outputs. In these cases, where ground truth measurements are not present, we dynamically assign the role of ground truth to the latest agreement among candidate values. These additional steps are shown in Figure 4.1, which shows where our methodology fits in a system that uses IoT devices to obtain observations.

The necessity for higher-quality sensor data applies especially to emerging digitalised systems, such as smart cities, industrial production and other cyber-physical domains. They involve large amounts of continuous data streams aggregated from a multitude of sensors,

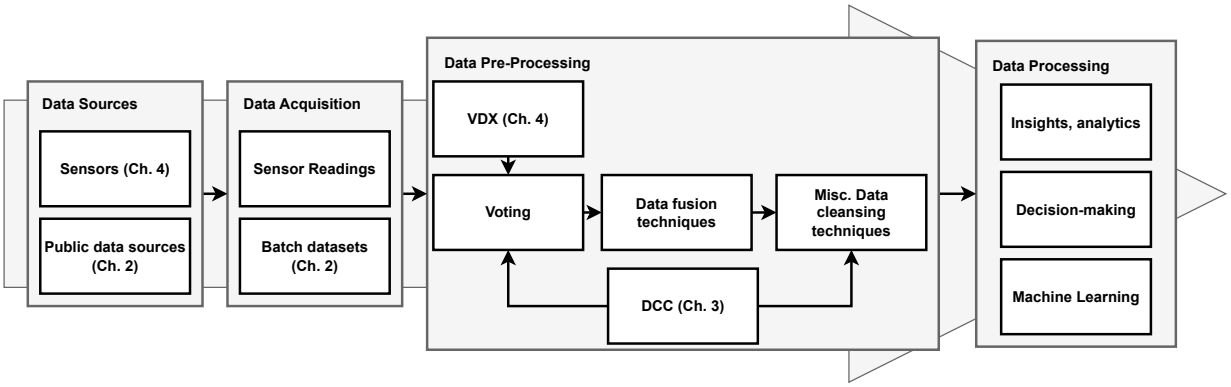

Figure 4.1: Positioning of data fusion, voting and deriving the ground truth in a typical IoT data pipeline

used as data sources. Sensor-based measurements are common especially in the Internet of Things (IoT) for triggering data-driven decisions. Quality issues in such distributed measurements[168] have profound adverse effects on systems leverage, and by extension, on humans and society, as shown in *irresponsible AI* community discussions[169]. Consequently, steps to improve input data quality within the data pre-processing step and in conjunction with data fusion are necessary to achieve better decisions and overall more reliable applications. Data fusion in general is a technique of merging different inputs[170] in an application-independent middleware to obtain a holistic view of physical objects. Data sources used as inputs to a data fusion system can have partial information that becomes meaningful once combined, or complete information but with potential conflicts between sources. Different techniques can be applied to each of these cases. Voting is an approach to fuse sensor data for the purposes of reliability and error mitigation[43, 154] in safety-critical environments. For instance, in avionics, three redundant physical sensors are mandated for each logical sensor[43], and vehicles with autopilot abilities have configurations such as eight cameras (including three forward ones) to achieve redundancy on the critical data acquisition path. Thus, in the absence of external ground truth (i. e., a fully trusted and accurate data source, often too expensive in practice), voting is a pragmatic substitute as it leads to internal ground truth upon which critical decision-making can be based.

In this chapter, we study and observe state-of-the-art voting approaches for sensor data fusion, applying them to three IoT scenarios relying on redundant sensor measurements: *(i)* light sensors in a smart building setting, *(ii)* Bluetooth Low Energy (BLE) beacons to track vehicle position in a (simulated) tunnel and *(iii)* an indoor positioning experiment, also using BLE beacons, emulating a smart shopping scenario. We focus on voting algorithms used to reach *data-centric consensus*[13] on numerical values, as these are relevant when merging sensor readings and leveraging historical records to factor in the reliability of individual sensors. In Section 4.7 we conduct three experiments on such IoT setups for real-time validation and pre-recorded data for the purpose of reproducibility. We exploit our findings to contribute a generic specification format that can be used to define voting schemes for several applications, particularly optimized for IoT and cyber-physical applications. We argue that such a format aids the development of distributed analytics applications by making them more needs-focused and reliable, while shielding software engineers from the voting implementation. Moreover, the increased robustness of the data, induced by multi-view data fusion implemented through the use of voting, facilitates the input data quality in data-centric artificial intelligence, a recent research direction aimed at overcoming *misprediction* due to

lack of input data assurance[171]. We replicate and evaluate the state-of-the-art history-based voting algorithms, i. e., voting algorithms in which the weight of votes is determined by past performance of the candidate in terms of agreement with the consensus. We observe the need for a method to bootstrap the algorithms to be more accurate before the history has been established, which would also improve the number of voting rounds it would take for the weights to converge to stable values representative of the reliability of the sensors.

Our contributions are twofold: *(1)* AVOC (Accurate Voting with Clustering), a novel bootstrapping method for initializing history-based voting systems, which we fully implement and evaluate with three practical IoT scenarios; *(2)* VDX, a new voting definition specification that precisely defines application requirements and allows users to select appropriate parameters for software voters.

The rest of the chapter is structured as follows. Section 4.2 surveys state-of-the-art voting algorithms, data fusion and data quality issues relevant to IoT. Section 4.3 investigates voting algorithms used to reconcile redundant data values. Section 4.4 presents AVOC, our approach. We describe the proposed format VDX for defining the voting process in Section 4.5. In Section 4.6 we detail our use-case scenarios and how we built our hardware prototypes. Section 4.7 presents our experimental evaluation using both a reference scenario dataset and our experimental setup. Our findings are discussed in Section 4.8. We conclude in Section 4.9 by discussing our findings and prospect future research directions in redundancy-based data quality in IoT.

4.2 Related Work

Data-driven decisions[172] are key elements of cyber-physical systems and digitalised applications of all scales and domains (e. g., smart cities, mobility, industrial production, home automation, etc). In many such systems, incoming data are subject to real-time analysis and subsequent decision-making. However, while systems research has allowed dealing with the volume, velocity and variety of multi-source data involved in the processing chain[173], issues still emerge regarding data quality, value and veracity. In this work, we focus on the accuracy (i. e., quality) of sensor measurements[4]. Data fusion across multiple homogeneous or heterogeneous sensors has been utilised to tackle the challenges induced by problematic data, improving analytics reliability through a variety of techniques (e. g., data association, state estimation, decision fusion, classification, prediction, machine learning and analytics[174]). Initial efforts exist to create standards and frameworks for data management and interoperability, for instance in the smart city space[175, 176]. However, they currently lack a common framework and standardized format. Our work proposes a new interoperable format to define voting-related data fusion.

Voting algorithms increase the reliability of measurements[154] in safety-critical domains, e. g., aviation[43] or self-driving cars[44]. We focus on reconciling numeric data using software voters, with either result selection or amalgamation techniques[154]. Specifically, we consider history-based voting algorithms[177] that weigh values based on the historical performance record of the candidate sensor. History-Based Weighted Average[177] weights the historical data to compute an output value. To improve the granularity of historical records,[178] uses a soft dynamic threshold. In[179], authors apply a hybrid approach using module elimination and dynamic threshold. We further detail these approaches in Section 4.3. Our approach, AVOC, extends the hybrid algorithm from[179] with a clustering component

Table 4.1: Comparison of state-of-the-art voting schemes.

Algorithm	Module Elimination	Dynamic Threshold	Output Selection
History-Based Weighted Average (Standard)	✗	✗	Mean
Module Elimination Weighted Average (ME)	✓	✗	Mean
Soft-Dynamic Threshold Weighted Average (SDT)	✗	✓	Mean
Hybrid History-Based Weighted Average (Hybrid)	✓	✓	Mean Nearest Neighbour

to improve performance before there is a long enough historical record, as well as to speed up convergence of the weights to stable values.

Some voting-based data fusion frameworks rely on specific description languages to define algorithmic details[180]. However, these approaches ignore history-based measurements. We also observe that modern voting algorithms are too complex to be represented in such terms. Where[180] defines voting as a three-step process (reaching quorum, excluding outliers and calculating results), modern algorithms often include further steps like weighing and updating historical records, or optimising reliability metrics[36]. We argue that a customisable voting framework serves as an encapsulation for sensor-fusion applications if built atop state-of-the-art approaches, as shown next, when we present our proposal for a generic definition specification for voting algorithms, VDX. A further benefit of our approach, is that it can then be integrated into data-centric tooling, such as ETL (Extract-Transform-Load) tools. Such tools, e. g., Singer.io[181] and Node-RED[182] are gaining popularity, but still have no support for voting-related data cleansing methods, which we argue would be beneficial for IoT scenarios with redundant sensors.

4.3 History-Aware Voting Algorithms

To combine values from uncalibrated redundant sensors, the history-based averaging algorithm (i. e., henceforth referred to as Standard algorithm[177]) either chooses a sensor output value or creates an amalgamation of these values. This approach can be optimized by temporarily ignoring values produced by modules with below-average historical records. This variant, i. e., Module Elimination Weighted Average (ME), assigns zero weights to the discarded values in the voting until their historical records improve by submitting better values, even if discarded in the voting itself.

The Soft Dynamic Threshold History-Based Weighted Average (SDT) introduces a finer-grain definition of agreement, beyond the binary-only definition[178]. Values between 1 and 0 can be assigned if values are not in agreement based on the accepted error threshold, but are in agreement based on a multiple of it. The magnitude of the multiple is defined by a parameter of the algorithm that can be tuned according to the needs of the specific use case.

We further consider the Hybrid History-Based Weighted Average (henceforth HYBRID [179]). It combines ME and SDT, while utilising agreement-based and not history-based weights. The HYBRID algorithm allows to choose a winning value rather than assigning the resulting average, using the mean nearest neighbour approach. Table 4.1 recaps these alternatives and their supported optimizations, which we describe in further detail in the remainder of this section. We discuss our findings on the output quality of all algorithms in §4.7.

4.3.1 History-Based Weighted Average

In the standard version of the algorithm[177], agreement of two values x_i, x_j of a set of N values is defined when they satisfy the inequality :

$$d_{ij} = |x_i - x_j| < \alpha \tag{4.1}$$

where α is a pre-selected margin. When a value x_i is in agreement with at least $(N-1)/2$ other values, then we set $S_i = 1$ otherwise $S_i = 0$. Thus, an input $S_i = 1$ represents an input in agreement with the majority. When n runs are completed the historical record of a module is defined as:

$$H_i(n) = \sum_{l=1}^{n} S_i(l) \tag{4.2}$$

This value is then normalised by n to obtain the state indicator P:

$$P_i(n) = \frac{H_i(n)}{n} \tag{4.3}$$

During a voting round, modules are assigned a weight based on their state indicator as:

$$w_i = P_i^2 \tag{4.4}$$

Finally, the weighted average is calculated as:

$$x_o = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i} \tag{4.5}$$

4.3.2 Module Elimination Weighted Average

This optimization[177] eliminates modules (sensors in our case) that perform below the average from the vote, assigning each a weight of zero. Specifically, a P_{avg} is calculated:

$$P_{avg} = \frac{\sum_{i=1}^{N} P_i}{N} \tag{4.6}$$

The weight calculation is modified by setting the weight w_i of a module to zero when it performs below average, i. e., its P_i is below P_{avg} as follows:

$$w_i = \begin{cases} 0 & \text{if } P_i < P_{avg} \\ P_i^2 & \text{otherwise} \end{cases} \tag{4.7}$$

And then the average calculation is performed as above in Equation (4.5).

4.3.3 Soft Dynamic Threshold Weighted Average

The Soft Dynamic Threshold algorithm[178] replaces the margin α with a dynamic variant v_t based on the input x, computed as:

$$v_t = px \tag{4.8}$$

Where p is a proportional constant, i. e., the error margin is a proportion of the input value and not an absolute value for all inputs. The algorithm then replaces the Boolean definition

of Si defined above. Rather than two modules i, j with a distance of d_{ij} ,the parameter S_{ij} becomes:

$$S_{ij} = \begin{cases} 1 & \text{if } d_{ij} \leq v_t \\ (\frac{k}{k-1})(1 - \frac{d_{ij}}{k*v_t}) & \text{if } v_t < d_{ij} < k * v_t \\ 0 & \text{if } d_{ij} \geq k * v_t \end{cases} \tag{4.9}$$

Where k is a tunable parameter. This parameter is chosen on a use-case basis and will determine how permissive the algorithm is to considering values to be in partial agreement. Partial agreement in this context means that the value is outside of the agreement threshold v_t but within the tunable, more lenient $k * v_t$. Then S_i is calculated by normalising:

$$S_i = \frac{\sum_{j=1, j \neq i}^{N} S_{ij}}{N - 1} \tag{4.10}$$

$H_i(n)$ and $P_i(n)$ are calculated as before in Equation (4.2) and Equation (4.3). Hence, SDT computes the weights, where w_i is obtained as:

$$w_i = \begin{cases} 2P_i(n) & \text{if } 0.5 \leq S_i \leq 1 \\ P_i^2(n) & \text{if } 0 < S_i < 0.5 \\ 0 & \text{if } S_i = 0 \end{cases} \tag{4.11}$$

Finally, the output is calculated as before in Equation (4.5).

4.3.4 Hybrid History-Based Weighted Average

The hybrid algorithm can be viewed as a combination of the Module Elimination algorithm and the Soft Dynamic Threshold algorithm. It works as follows. S_{ij} and S_i are computed by Equation (4.9) and Equation (4.10) with the change that v_t is once again replaced with α as in the standard algorithm. Weights are calculated similarly to the Module Elimination algorithm, specifically:

$$w_i = \begin{cases} 0 & \text{if } S_i = 0 \text{ or } P_i(n) < P_{avg}(n) \\ \frac{\sum_{i=1}^{N} S_i}{N-1} & \text{otherwise} \end{cases} \tag{4.12}$$

However, the final output y of the vote is not the average $\overline{y}$ but rather the input x_i closest to it. The final difference is the calculation of the history, based on the final output y and computed in a similar manner to S_{ij}:

$$h_i(n) = \begin{cases} 1 & \text{if } d_{iy} \leq \alpha \\ (\frac{k}{k-1})(1 - \frac{d_{iy}}{k*\alpha}) & \text{if } \alpha < d_{iy} < k * \alpha \\ 0 & \text{if } d_{iy} \geq k * \alpha \end{cases} \tag{4.13}$$

The history of the module for n rounds is:

$$H_i(n) = \sum_{i=1}^{N} h_1(n) \tag{4.14}$$

Finally $P_i(n)$ is calculated as in Equation (4.3).

Algorithm 2: Soft-Threshold 1-D Clustering

```
Input  : Values: V, threshold-factor: f
Output : Clusters: C
Create-cluster(C, [])
for value in V do
    for Cluster in C do
        if (1 − f) × avg(Cluster) ≤ value ≤ (1 + f) × avg(Cluster) then
            cluster.add(value)
        end
    end
    else
        Create-cluster(C, [value])
    end
end
```

4.4 AVOC: Accurate Voting with Clustering

History-based algorithms typically fall back to standard average (or a similar unweighted approach) on the first round until a historical record is established or when the weights become 0 due to severe issues with the data. Weights can drop to 0 after a series of disagreements, which results in notorious disagreers being rated as untrustworthy by the system applying the algorithm. To counteract these issues, we introduced AVOC[183]. AVOC builds atop the Hybrid algorithm by applying a simplified clustering algorithm during the first round when the weights are all 0. The clustering step eliminates obvious outliers, improving the accuracy of that round compared to mean average, with little performance overhead and faster convergence speed. We evaluate how AVOC performs in comparison to the state-of-the-art voting algorithm, and how it can be combined with it to achieve optimal performance in an indoor positioning scenario.

Algorithmic approach.

For the clustering step, we leverage a similar logic to the agreement calculation in voting algorithms: we check for values within a given scaling threshold of each other (which is selected to mirror the parameters of the given algorithm), and group the values in agreement. Then, we select as output value the average (or its closest real value) of the largest group (if we are using the mean nearest-neighbour approach to output selection). Algorithm 2 formalizes this approach. This grouping logic is similar to DBSCAN[184]; AVOC opts for self-calibration, rather than requiring costly parameters tuning. This is achieved through a majority vote with a soft-dynamic error margin (as the margin depends on a reference value). The clustering step is used for bootstrapping a new set of modules, or as a fallback in cases of issues. To reach this goal, we use the historical record value for each module, and declare that the clustering approach should be used when all records are 1 (indicating a new set) or 0 (indicating a failure of the system or an extreme data spike). Figure 4.2 depicts this logic.

Generalisation. Generalising this approach for multi-dimensional data, an unsupervised clustering algorithm can be used such as Meanshift[185] or X-Means[186]. The logic of choosing an output value would be similar. However, in such scenarios, choosing a single output vector for multiple dimensions is non-trivial as the complexity of data and correlation of errors considerably increases. To mitigate, the voting approach can be applied for each dimension separately, leaving other data fusion techniques to process the multi-dimensional

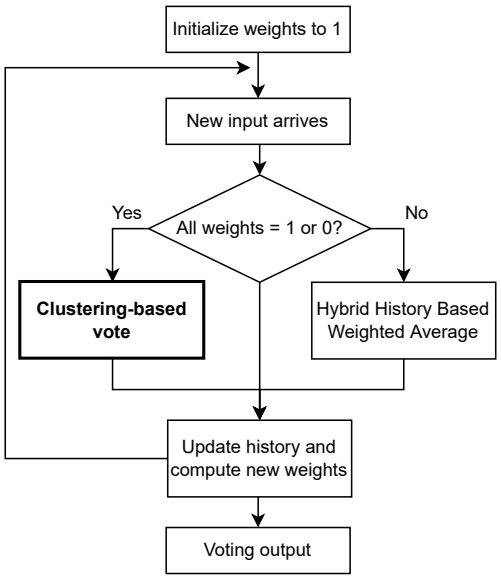

Figure 4.2: Flowchart of our extension to the state-of-the-art voting algorithm, resulting in AVOC. The clustering version of the vote is triggered when weights are all 1 or 0, indicating a freshly initialized system or a large amount of flaws that made the weights unusable. After the clustering vote concludes the weights are calculated as in the unmodified algorithm, but using the output of our custom vote, therefore reducing the transient errors.

results. In AVOC, we follow the approach of voting on each dimension separately, without incorporating the clustering itself.

4.5 VDX: Voting Definition Specification

To enable reliable implementations and improve the usability of software voters, we contribute a new voting specification scheme and parsing logic. The scheme can define any of the algorithms described above, as well as simpler ones without history.

Software-defined voting schemes exist, e. g., Voting Definition Language (VDL)[180]. Those predate more complex history-based voting approaches, and extending VDL for finer-grain algorithmic definitions is challenging. However, our specification VDX supports the relevant parameters of VDL, enabling our definition to describe a superset of VDL-scoped algorithms. The full schema, as well as a sample implementation and usage examples can be found at: `https://doi.org/10.5281/zenodo.8069916`. The repository also includes an interactive application that allows users to compare the algorithms presented with the state of the art (Figure 4.6) as well as experimental extensions still under development (eg. for handling multi-dimensional data as mentioned in Section 4.4).

Capabilities. Listing 4.1 shows our AVOC algorithm using this scheme as an example. VDX allows the specification of several parameters, including quorum (i. e., how many candidates need to submit values for a vote to be triggered), exclusions (to automatically prune outliers) collation techniques similar to VDL, e. g., "mean nearest neighbour" (Listing 4.1, line 12). It extends VDL by allowing the selection of a history algorithm (Listing 4.1, line 7), additional parameters (Listing 4.1, lines 8-11), and whether to enable clustering algorithm as a bootstrap/fallback mechanism (Listing 4.1, line 13). Another extension VDX adds over VDL is the ability to vote on categorical i. e., non-numeric values, such as character

Listing 4.1: Vote definition sample in VDX JSON format.

```
{
  "algorithm_name": "AVOC",
  "quorum": "UNTIL",
  "quorum_percentage": 100,
  "exclusion": "NONE",
  "exclusion_threshold": 0,
  "history": "HYBRID",
  "params":{
      "error": 0.05,
      "soft_threshold": 2
  },
  "collation": "MEAN_NEAREST_NEIGHBOR",
  "bootstrapping": true,
}
```

Figure 4.3: Vote definition sample in VDX JSON format.

strings and JSON blobs. In such cases however, several features are disabled. Value-based exclusion cannot be applied, as there can be no mean or standard deviation calculation. The 'standard' and 'module-elimination' algorithms for deriving module history are available, however the 'hybrid' algorithm is not, as the fine-grained agreement definition cannot be applied to non-numeric values. Finally, clustering-based bootstrapping cannot be applied to categorical values and the only collation method is the weighted majority vote. Software voting implementers may re-introduce some of these features by supplying a custom distance metric for categorical values. Figure 4.4 depicts the decisions an implementer needs to make to design a voter using VDX. Though the tree itself is simple, it can be used as a basis for a dialog-based configuration system, to assist developers even further in designing their algorithms.

Figure 4.5 depicts the proposed workflow leveraging the voting specification. The voting format description is submitted to the voting system by the user to initialize the system. Sensor reading can then be submitted to the same system, directly to the exposed sensor reading fusion service. These will trigger voting rounds. The system needs to maintain a record of the success rate of each sensor (defined as P_i in Section 4.3) and the parameters each voter group uses. In principle, it is not required to retain the sensor readings from previous submission rounds (though some algorithms require it to retain the last output value to be used in the next round).

Limitations and assumptions. VDX currently cannot define algorithms that use parameters for the candidate values, e. g., MLV[36], or genetic voting algorithms[34], but assists already by introducing voting into further IoT software for reliable input data and analytics. Another limitation is that observations are always assumed to be single values. As such, for a more complex data structure, the implementation needs to present the VDX system with the values one at a time. This limits the accuracy of the voting for multidimensional or inter-dependent data, but is a necessity for the algorithms to function as implemented. Adapting the algorithms for use with configurable data structures is an avenue for future work. It should also be noted that VDX itself has no security features that protect against malicious actors, so this is left up to the client code to implement as needed.

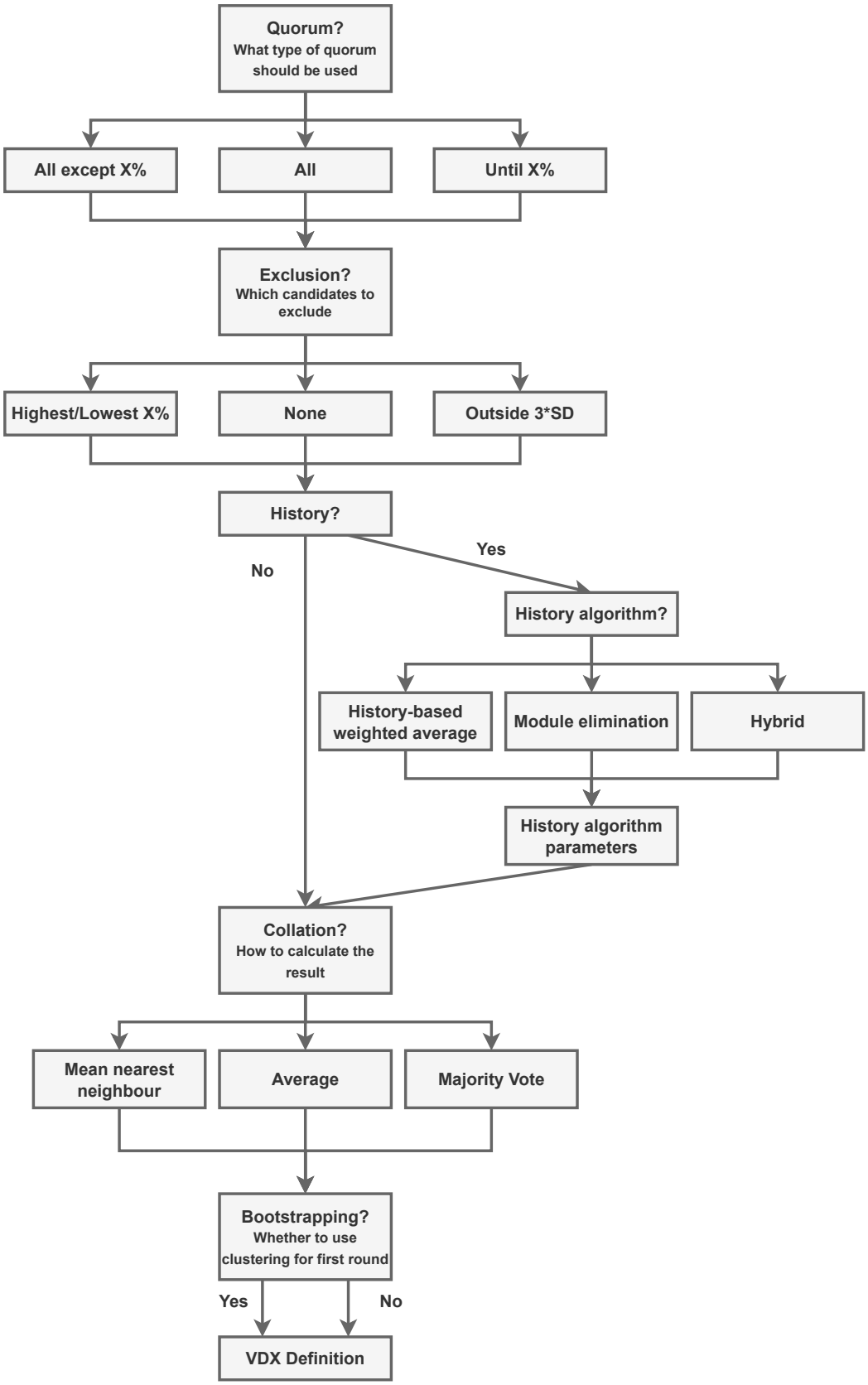

Figure 4.4: VDX decision tree for defining algorithms.

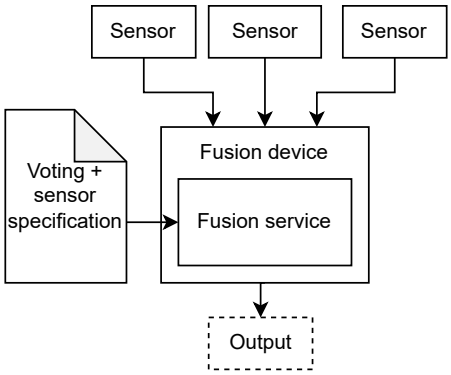

Figure 4.5: Overview of the voting specification usage. The same fusion service can be applied to a variety of setups (eg. edge/fog nodes, robots, IoT devices etc.) and a custom definition can be supplied during setup to select the appropriate behavior.

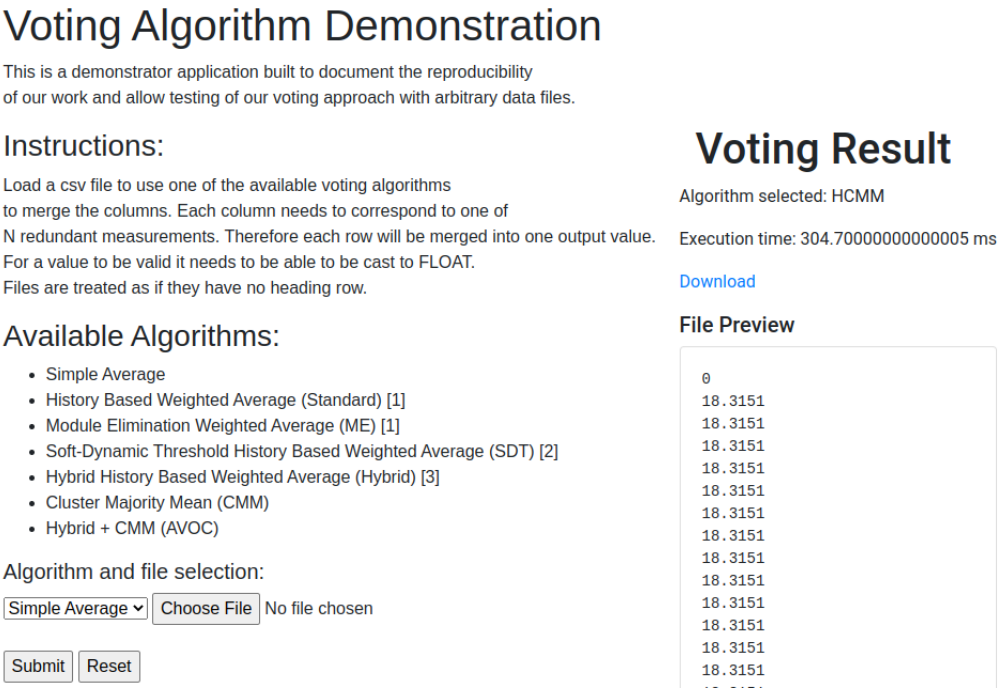

Figure 4.6: Algorithm comparison application.

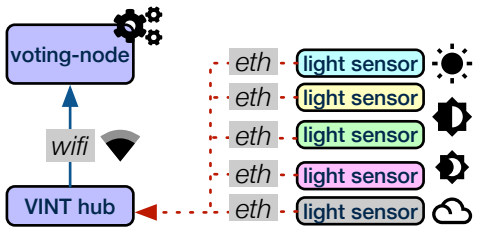

Figure 4.7: Light sensor use case: the sensors are wired via ethernet to a hub, streaming data via WiFi to the voting sink-node.

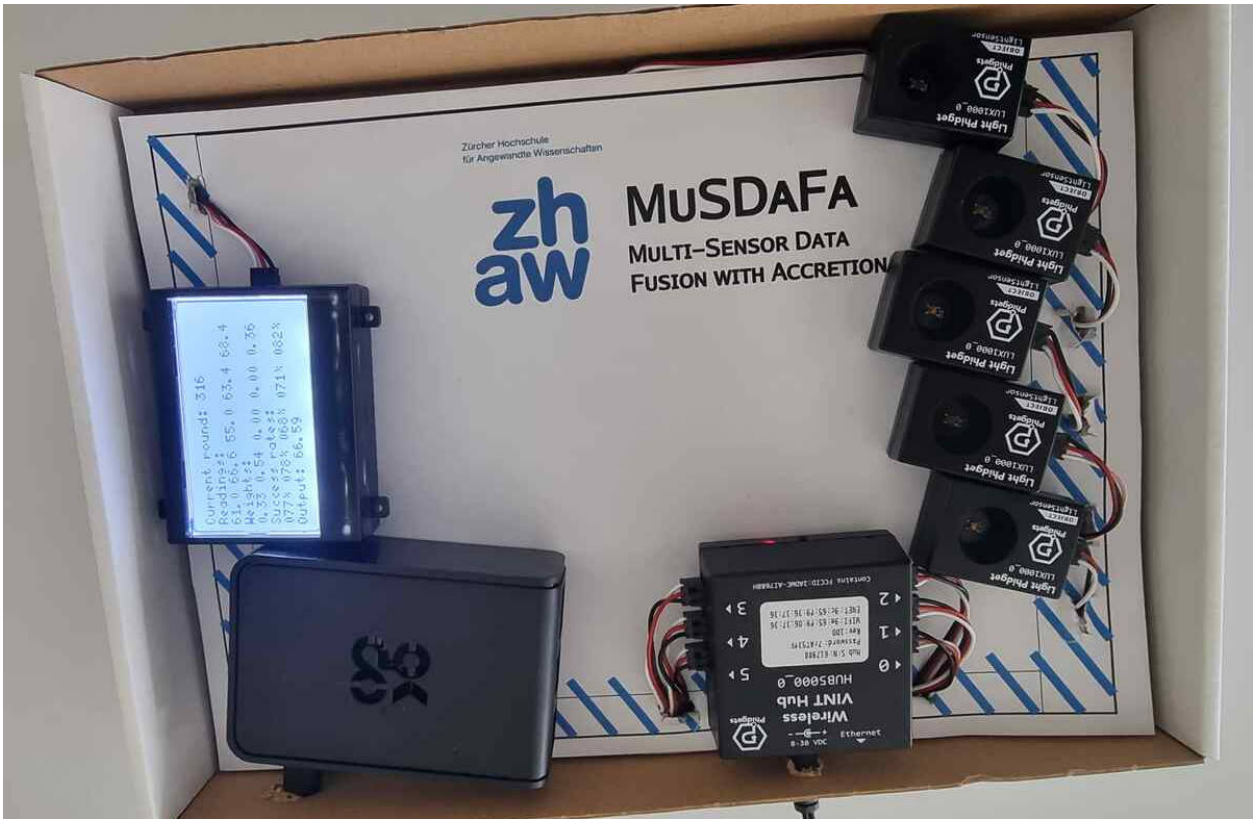

Figure 4.8: Portable 'shoe-box' testbed for our light sensor setup (Figure 4.7). A Raspberry Pi 4 runs the fusion script. An LCD display shows the voting results and weight values.

4.6 Multi-Sensor Application Scenarios

We devise three use-case scenarios to validate our approach. The scenarios represent sensor-reliant deployments where redundancy proves valuable. Specifically, we draw on the smart building and self-driving vehicle domains, both showcasing cyber-physical systems relying on sensor measurements to control other critical systems.

Figure 4.7 depicts our first use case, a sunlight detection system in a hypothetical smart building. Such a setup could be used to control automated window blinds, for example. This example was selected to showcase a scenario with redundant sensors that are expected to produce identical results. The hub is connected to a sink node to record 10'000 rounds of concurrent measurements from 5 sensors, polling at 8 samples/s, to create a reference dataset representing 1250 seconds of data collection. Each round produces 5 float values per sensor. The reference dataset consists of the raw readings from all sensors and is used to compare all voting algorithms on the same set of values. We built a portable demonstrator

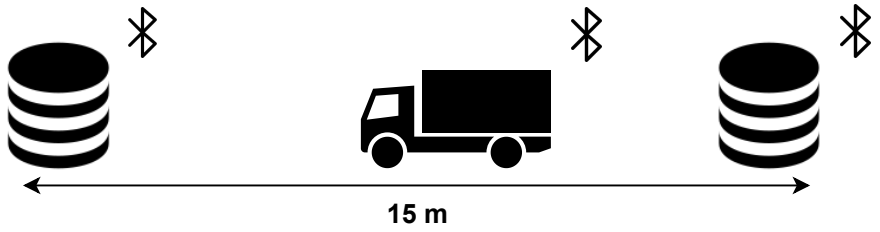

Figure 4.9: BLE beacon use case. 2 stacks of beacons 15 meters apart with a robot driving between them, taking signal strength measurements along the way.

that executes them and our proposed voting algorithm, AVOC (detailed in Section 4.4), leveraging a Raspberry Pi 4B unit (see Figure 4.8). It uses a Phidget Wifi hub[187] (VINT) connected to a set of 5 LUX1000 Light Phidget sensors[188]. Input, weights and results are shown on an LCD screen[189] connected to the hub. The portable version let us confirm the feasibility to execute on constrained hardware, combining both redundant measurements and voting.

In the second use case, we mimic a typical smart-city/self-driving vehicle application that relies on indoor positioning to track the position of a moving unit (e. g., a robot). We simulate the operations to track a cargo vehicle traversing a tunnel, as shown in Figure 4.9. Such vehicles use Bluetooth (BLE[190]) beacons as *milestones* to locate the position of the truck, as done in emerging country-wide systems (i. e., CST – Cargo Sous Terrain[191, 192]). We deploy two stacks of nine redundant beacons and a cargo vehicle using a Lego Mindstorms EV3[193] robot. As the Bluetooth receiver on the robot was incompatible with the beacons due to lack of BLE support, we installed a laptop on top of it acting as a pragmatic substitute receiver with edge processing capabilities, and without affecting the generality of the findings other than affecting the speed of the robot (which has no effect on what we are measuring). Figure 4.10 shows the prototype.

The robot drives slowly in a straight line with no line-of-sight obstacles from one beacon stack to the other, across a distance of 15 meters. The speed of the robot was set to 7% of its specified top speed (0.09 m/s). We collected as many data points as possible along the route, resulting in 297 measurements per beacon, noting that autonomous cargo systems like CST proceed at around 8.3 m/s, thus having 99% fewer measurement samples available for voting. We elected not to apply filtering and to vote on the raw values in this experiment, to evaluate the performance of our voting system in the presence of the typical unpredictability of BLE singal-strength measurements.

The third use case is a positioning experiment in a real-world environment, as shown in Figure 4.11. The experiment context is meant to emulate a smart shopping scenario, recreated in our office building. Here, we have placed 4 stacks of 5 beacons in a room of dimensions 7.18m x 1.98m. Each stack represents a store shelf in a hypothetical store. A user with a bluetooth-enabled device walks around the room to collect measurements. This scenario allows us to compare our approach to other sensor fusion methods for indoor localisation, namely Kalman filtering.

4.7 Evaluation

This section presents the experimental evaluation of AVOC and the VDX system in general. With VDX, we fully implemented the three use-case scenarios from Section 4.6, and compared

Figure 4.10: Robot driving to the circled beacon stack destination. The laptop acts as bluetooth receiver and edge voter.

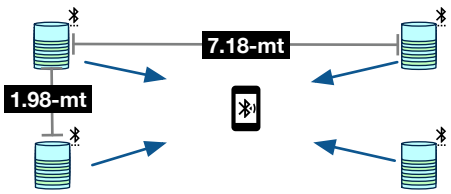

Figure 4.11: Indoor positioning experiment setup. 4 stacks of beacons are deployed in the 4 "store shelfs". The user device is placed in the middle of the room and the user walks around the room to collect measurements.

the error-correction performance of the various voting schemes. More concretely, we use our light sensor experiment as a model for sensor-based time-series data, which can be voted upon using history-based algorithms. We use the BLE beacon experiment to test various algorithms on their ability to function on low-quality data. Finally, we use the localisation experiment to compare our voting approach to Kalman filtering, a state-of-the-art approach for sensor fusion, as well as test how we can effectively combine the two approaches to improve accuracy and performance. Our evaluation answers the following questions: *(Q1)* Can AVOC improve the output quality of our use-case scenarios? *(Q2)* Can it mitigate injected errors and reach the same output? *(Q3)* Which algorithm fits which scenario better, and *(Q4)* How can we leverage VDX to customise the voting behaviour for each scenario?

Implementation details. We implemented AVOC and the approaches from Section 4.3 in Python 3.9, for a total of 490 LOC. We note that though the evaluation was done with pre-recorded data for reproducibility purposes, the system can execute a history-aware voting round in 1 millisecond and a stateless vote in 50 microseconds (datastore reads and writes being the bottleneck). Thus, the system can operate under soft real-time constraints, as in the case of our 'shoe-box' demonstrator in Figure 4.8 as well as many critical cyber-physical applications in practice.

UC-1: Light sensors. We used the 10'000 value dataset recorded using our light sensor setup (Section 4.6) to gather the raw data and reference values for the baseline algorithms (Figure 4.12-a). Then, we injected an artificial outlier sensor, by adding +6 lumen to one of the sensors. We compare the performance of the different algorithms according to

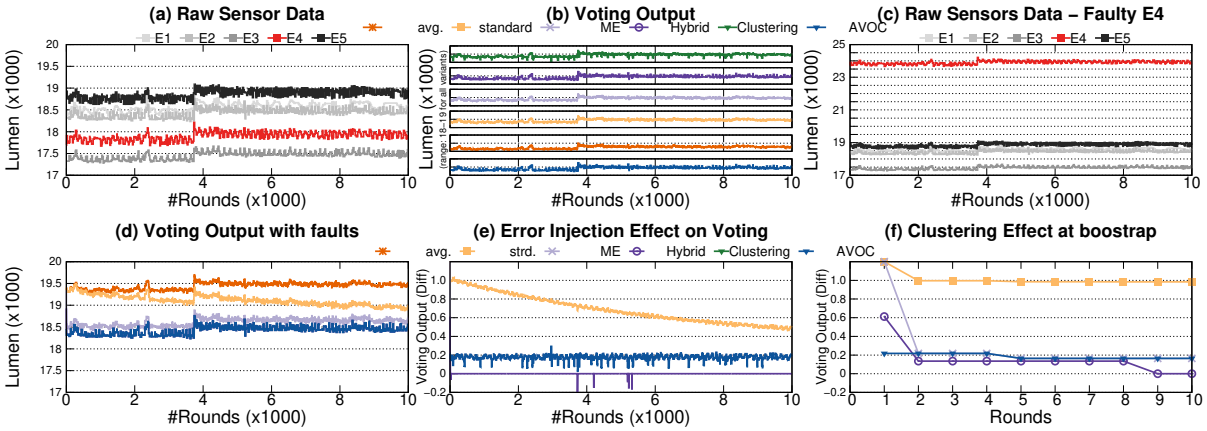

Figure 4.12: Comparison between our voting approach AVOC and the state-of-the-art approaches. (a) reference data, captured by our light sensor setup for 20min. (b) voting output using AVOC on the reference data. (c) Reference data with injected errors (1 faulty sensor). (d) Output of HYBRID, Clustering and AVOC under these errors. (e) Output difference between voting on the raw values and voting on the error-injected values. (f) Zoom on the first 10 rounds.

the following metrics: *(a)* voting rounds required to converge back to the baseline, and by extension how quickly outliers are eliminated; and *(b)* how far the new stable value is from the original.

We make a first comparison using the raw reference data. In this scenario (Figure 4.12-b), all 6 variants performed equally well, with outputs matching almost completely. The error injection case (shown in Figure 4.12-c) exhibits some interesting facts. First, the Standard algorithm exhibits high initial skew, which is then slowly mitigated as the *faulty* sensor (E4) is de-emphasised. However, even after 10000 voting rounds of voting (20 minutes in our experiments), the skew is not eliminated completely. This is where the module elimination feature of ME is beneficial, as the faulty sensor is quickly eliminated in round 2, as performing below average compared to the rest. However, as seen in Figure 4.12-c, the gap created by skewing E4 indicates how E3 is also now tagged as an outlier, and its result is skewed upwards by 0.2lm.

The HYBRID algorithm also uses a granular definition of agreement score, but combines it with the aggressive elimination of modules from ME. For our experiment, this is the *best of both worlds* result (also shown by the differentials in Figure 4.12-e) where, minus a few spikes, the value is identical to the pre-error output.

For completeness, we show clustering-based voting on its own without combining it with HYBRID (which remains ideal for this scenario). This can be seen in Figure4.12-e. We observe similar behaviour to ME, with E4 being excluded from the output immediately. Differently from ME, E4 was also excluded from the first round. In contrast, E3 was not always excluded, due to the lack of history-based elimination, indicating higher variations in the output. Regarding clustering-only voting (COV), it significantly outperforms other stateless approaches, i. e., weighted average without history. This result indicates that the COV approach fits well scenarios where maintaining historical result records is impractical: short-lived sensor measurements, one-time comparisons of datasets, etc.

One consistent observation in the error injection experiment is that history-based algorithms experience a spike on startup, as the artificially modified value is skewing the output but is not yet mitigated by the history. This is the phase where in principle the clustering step detailed above has a higher chance of affecting the results. Indeed, although

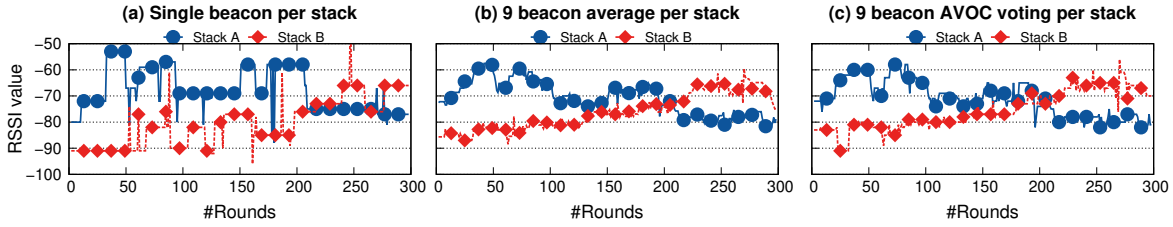

Figure 4.13: Results of the BLE beacon experiment. (a) shows the RSSI output when only one beacon from each stack is used. (b) shows the average RSSI value of all 9 beacons in the stack for each round. (c) shows the AVOC voting output for all 9 beacons in the stack. Averaging provides visibly less ambiguity in determining which stack is closest to the robot, when compared with the mean nearest neighbour selection used for HYBRID.

the clustering algorithm alone is not as accurate as ME or HYBRID, it overcomes the initial data spikes. Thus, a system capable of fallback to it when history is not available or suspected unreliable, can benefit from its inclusion.

Next, we run AVOC, which concretely combines the clustering step with HYBRID. We observe how the initial spike is quickly pruned (Figure 4.12-f) within the initial rounds. The bootstrap boost can also be noticed: due to the better history adjustment in round 1, the voter already *learns* to exclude E3 from round 2, returning to its pre-error output almost instantly, despite the clustering being only used once. As AVOC converges within 200ms, the experiment confirms its utility for fast accurate voting and provides a positive response to our questions (Q1) and (Q2), as we improve the reliability of the output even in the presence of the injected errors.

UC-2: BLE beacons. We leverage this second use case with BLE beacons and a Lego Mindstorms EV3 robot to study a scenario with more anomalies and faults. We set up two stacks of 9 beacons each 15 meters apart in an indoor corridor with no obstacles. The robot drives at 7% of its full speed in a straight line, from one stack to the other. The robot takes continuous RSSI (Received Signal Strength Indicator) measurements for each beacon along the way. This experiment examines if the high redundancy and voting actually are beneficial. The scenario simulates the operations to locate a vehicle traversing a tunnel using beacon stacks as *milestones* along the way, determining the closest stack to the vehicle. The state of the art in leveraging RSSI for positioning relies on filtering[194] or collaborative positioning[1] to improve stability and output quality. However, since the scope of our experiment is to evaluate the effects of voting on sensor measurements, we kept the raw RSSI values from the sensors, to keep the values as close to the original measurement as possible for the voting, before applying other techniques to improve positioning performance. The resulting data, which we plan to publicly release, lacks several values as well as mismatched readings in each stack, providing a more challenging fusion scenario. Such missing values allowed us to identify several fault scenarios, which we describe next.

Fault scenario: missing values. Due to some beacons not being reachable from the BLE receiver (i. e., the laptop in our experiment). Missing values can reduce the reliability of the output measurement, since fewer candidate values are being considered, and potentially prevent reaching a consensus value if most or all values are missing. A small amount of missing values, i. e., less than the majority, does not prevent the system from converging to a common result, though it reduces the redundancy as well as the number of candidates considered, and as a consequence the trustworthiness of the outcome. If the majority or

all values are missing, the result would no longer be trustworthy, and the system should either revert to the last accepted result, or raise an error. Obviously, these failure scenarios should be accounted for when the voting behaviour is defined. Due to the complexity possible and the variability by scenario, these behaviours are currently not modelled by VDX itself, and are instead left up to the client code to define. In our test-bed implementation, the error-handling procedure was programmed into the implementation of each algorithm. It is possible to make this behaviour customisable, including the custom error-handling in the voting parameters of the schema definition.

Fault scenario: conflicting results. In case of conflicts, a majority agreement on outputs using automated voting is less likely to be reached. It is possible that a relative majority agrees on an output, but they are an overall minority, and no absolute majority exists. Especially in systems with a small number of votes, ties might occur more easily and tie-breaking mechanisms kick in, such as proximity to the previous output. These fault scenarios clearly show that **setting the constraints of a voting system is non-trivial, and that voting algorithm implementations in a generic data fusion platform should be parametric**. In addition, there should be a voting specification declared for the target application, to take the desired error-handling behavior into account. Accounting for such fault scenarios allows to implement more robust versions of the algorithms. It is also possible to extend VDX in a future revision to support high-level descriptions of the desired fault-handling policy. Examples of such policies include rejecting a round of measurements if there is no majority quorum or majority agreement.

We then tested our voting algorithms on the BLE experiment data. We used the same recorded values for each algorithm. We run voting between the 9 sensors of each stack to create 2 output values per round (i. e., 1 per stack). We observed that the method for computing the history of each sensor has no effect. The output of all history-based algorithms overlaps completely. (They are not all plotted due to space constraints.) This is because the chaotic nature of the measurements meant the history values were all very low, as there were few agreements between the sensors. We observe however that the value collation method has impact, e. g., averaging the weighted values, a mean-nearest-neighbour selection, etc. This created 2 algorithm groups, those averaging and those choosing the mean-nearest neighbour value, with every algorithm in each group performing identically to each other. In order to determine the best results, we study the number of rounds while it is ambiguous which stack of sensors is closest to the robot at any given time. Figure 4.13 presents these results.

Figure 4.13-a is the reference: if each stack only had one sensor, it is not possible to identify the closest stack to the robot for most of the duration of the experiment. Simply averaging the values of the 9 sensors (Figure 4.13-b) produces a less ambiguous result. We present the results using AVOC in Figure 4.13-c. In spite of the method used to create a historical record for each sensor, what had the most impact on the output was whether or not the last step was to average the values or to select a value (with averaging being the better option in our experiment). In scenarios with a high degree of noisy data, such as the BLE one, relying on historical record has no practical value. This confirms our initial assumption that **there is no optimal voting method for all applications**, despite common elements shared by our scenarios (e. g., having to reconcile numerical values from a group of sensors). Thus we conclude that the answer to (Q3) depends on the specifics of the use case, and that the customisation offered by our specification allows us to address (Q4).

UC-3: Indoor positioning.

Our last experiment utilized the same BLE beacons as the previous one, but this time in a standard indoor positioning scenario. The scenario was based on a smart shopping

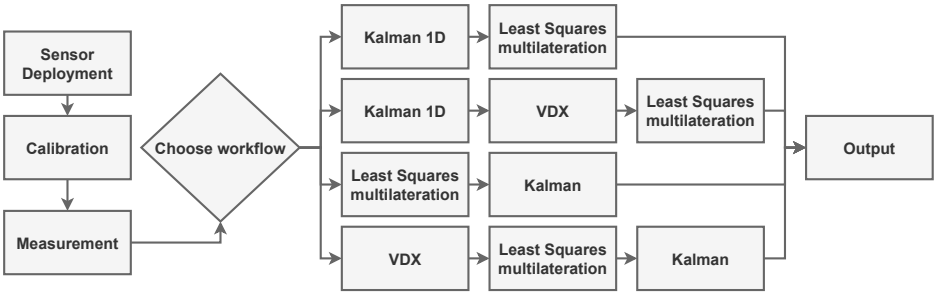

Figure 4.14: Overview of the 4 workflows we used for data cleaning. 2 workflows are from the state of the art, and employ different versions of the Kalman filter to clean the data. The 2 other workflows use voting between the 5 sensors in each stack rather than treating each sensor independently.

context, in which we placed 4 "store shelves" in the corners of a rectangular area in our lab, and stacked 5 beacons on each shelf (to simulate the 5 levels of a supermarket shelf). The dimensions of the experiment area are 7.18m x 1.98m.

To calculate the position using BLE beacons, we used the following formula for range estimation, based on the path loss model[195]

$$d = 10^{\frac{rssi_0 - rssi}{10 \times n}} \tag{4.15}$$

where $rssi_0$ is the RSSI value at 1m, $rssi$ is the RSSI value measured by the receiver, and n is the path loss exponent. As the beacons used do not advertise their TX power (a pre-calculated value for the RSSI at 1m), we needed to measure the RSSI at 1m for each beacon, as well as the path loss exponent. To obtain the path loss exponent, we measured the RSSI at 2m, and solved for n using Equation 4.15.

To calculate the position, we used the ranges obtained from the beacons, and the position of the beacons, to define circles which we used to solve the multilateraion problem using the least squares method. The system of circle equations is:

$$(x - x_i)^2 + (y - y_i)^2 = d_i^2 \tag{4.16}$$

where x_i and y_i are the coordinates of the beacon, and d_i is the range to the beacon. The solution to this system of equations is the position of the user. We used the `scipy.optimize.least_squares` function in Python to solve the system of equations. This uses the Levenberg-Marquardt algorithm[196] to solve the non-linear least squares problem.

These two parts of the position estimation remained the same for all the algorithms we tested. We then recorded RSSI values from the beacons at different known locations in the experiment area, and used these values to test the algorithms. As before, we used the pre-recorded data to ensure all 4 algorithms were tested on the same inputs.

The 4 workflows we used are presented in Figure 4.14. All methods used variations of the Kalman filter[195] for smoothing, according to the state of the art. First, we used the 1D Kalman filter to clean the data, and then used the Least Squares Multilateration to derive the output. This means we applied the filter directly to each individual range estimation, and then used the filtered values to calculate the position. This method of data cleaning is similar to the one used in[197].

Table 4.2: Execution breakdown of the 4 data cleaning methodologies in the indoor positioning experiment

Method	**Filter calculations**	**Filter time**	**LSML calculations**	**LSML time**	**Voting rounds**	**Voting time**	**Mean Positioning Error**	**Time**
Kalman 1D	20 x 3 measurements	80 µs	1 x 20 circles	20 ms	-	-	156.75 cm	502 ms
Kalman	1 x 3 measurements	236 µs	3 x 20 circles	44 ms	-	-	208.25 cm	567 ms
Kalman 1D + Voting	20 x 3 measurements	80 µs	1 x 4 circles	12 ms	4 x 5 values	6 µs	145.75 cm	496 ms
Kalman + Voting	1 x 3 measurements	319 µs	3 x 4 circles	31 ms	12 x 5 values	18 µs	159.25 cm	557 ms

The second workflow adds a voting step between the filter and the multilateraion. As with the tunnel experiment, we used a simple averaging voter here, as the data is limited in history and also quite noisy.

The third workflow uses the 2D Kalman filter after the multilateration. This means we applied the filter to the calculated position to smooth it. This method of data cleaning is similar to the one used in[198].

Finally, the fourth workflow adds a voting step before the multilateration to the third one. The raw range data was voted on, before the multilateration was calculated.

A visual representation of the two Kalman filter workflows is shown in Figure 4.15 and Figure 4.16. The aggregated results of the 4 workflows are presented in Table 4.2. The table presents the results per measured user location, averaged across all positions used in the experiment. In each position we took 3 measurements, 5 seconds apart, from each of the 20 beacons (5 beacons per stack, 4 stacks). Here we can see that voting improved the result of both methods. Perhaps surprisingly, the 2D Kalman filter did not improve the result, and in fact made it worse. This can be explained by the fact that the measurements were only taken from static positions, and not from a moving user. This means that the 2D Kalman filter was not able to take advantage of the temporal information and velocity.

The performance results from the same table are also of interest. We observe that even though the 2D Kalman filter takes more time to compute, the difference is negligible compared to the total runtime. Voting reduces the amount of time that the Least Squares method takes to compute, which is always the most time-consuming step.

To further test the effect of voting on performance, we artificially increased the number of beacon groups by duplicating the real measurements. As expected the filtering time for the Kalman 1D filter scaled linearly, as the number of measurements in each filtering step remained the same, and only the number of steps scaled with the number of beacons. Each filtering step takes approximately 1.33 µs (assuming again 3 measurements per beacon).

Similar behavior was observed for the voting with each voting round between 5 values taking approximately 1.5 µs. It should be noted here that enabling the history-aware features of VDX would increase the time of a voting round depending on the implemented storage method for the history. In that case, reading and writing the histories into a backend component would take the majority of the voting time, depending on the efficiency of the implementation.

Increasing the number of sensors does not increase the time the 2D Kalman filter takes to compute, as it is applied to the final positioning result, so only 2D coordinates are applied as a measurement. Therefore only the amount of measurements per sensor increasing

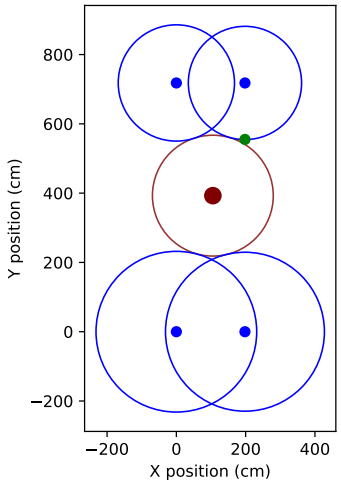

Figure 4.15: Sample visual output of the localisation program using the 1D Kalman filter. Here the filter is applied to each range measurement individually and multilateration is performed at the end. The 4 blue circles represent the raw range measurements. The red circle represents the filtered position estimate. The green dot represents the true position of the user.

would affect it, but that would require a prolonged measurement period, which would not fit the indoor navigation scenario.

The Least Squares Multilateration seemingly scales differently in the real experiments, though when inflating the number of sensors we observed that for very high sensor counts (we tested between 100 and 20000 circles) the time is quite consistently linear. The average time divided by the number of circles was 400 µs per circle, above 5000 circles, as below that the overhead still makes the duration less predictable. It is consistently the most time-consuming part of the workflow, so voting, which reduces the number of circles it needs to solve, is a good way to improve performance (provided the scenario allows the sensors to be grouped).

UC-3+: Comparison with Neural Network fingerprinting method.

As a further point of comparison, we repeated the experiment of UC-3 using a different setup. This extension was made as part of our work in extending the localisation experiment setup for use in smart shopping[199]. Rather than using 4 stacks of 5 beacons each, we arranged 12 beacons in a 3 x 4 grid, so that they would be evenly spaced. This arrangement can be seen in Figure 4.17. We then decided to utilize the fingerprinting method[200] for positioning. The standard way to implement this, is to draw a grid on the location, visit each location in the grid and take a measurement from each of the beacons at that location. The result is a 'fingerprint database' of signal strengths at each location, which can be compared to the live measurement to obtain the location of the user.

Our implementation included multiple measurements at each location, for a total of 524 data points. For the actual location prediction, we used a Sequential Neural Network. The neural network's input is the 12 signal strengths from the beacons. The network consists of 3 dense layers of 50 neurons, the first using a sigmoid activation function and the rest (including the output) using ReLu (Rectified Linear Unit). The output is the 2 coordinates of the predicted location. The data was split as 80/20 for training/test and training of this simple model took 6 seconds on a laptop with an 8th Gen Intel i7 processor and no GPU.

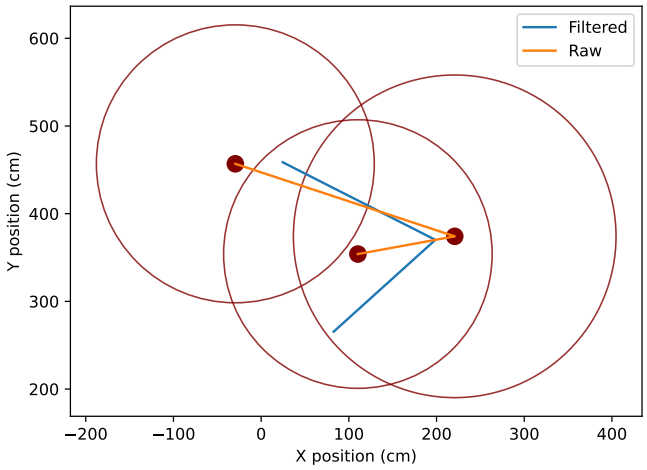

Figure 4.16: Sample visual output of the localisation program using the 2D Kalman filter. Here the filter is applied to the position estimate after multilateration. The 3 red circles represent the raw position estimates. The blue line represents the filtered position estimate.

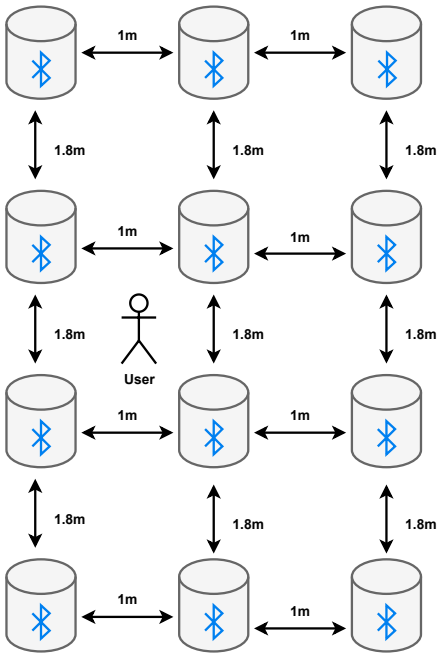

Figure 4.17: The 3 x 4 grid of beacons used for the Bluetooth fingerprint experiment

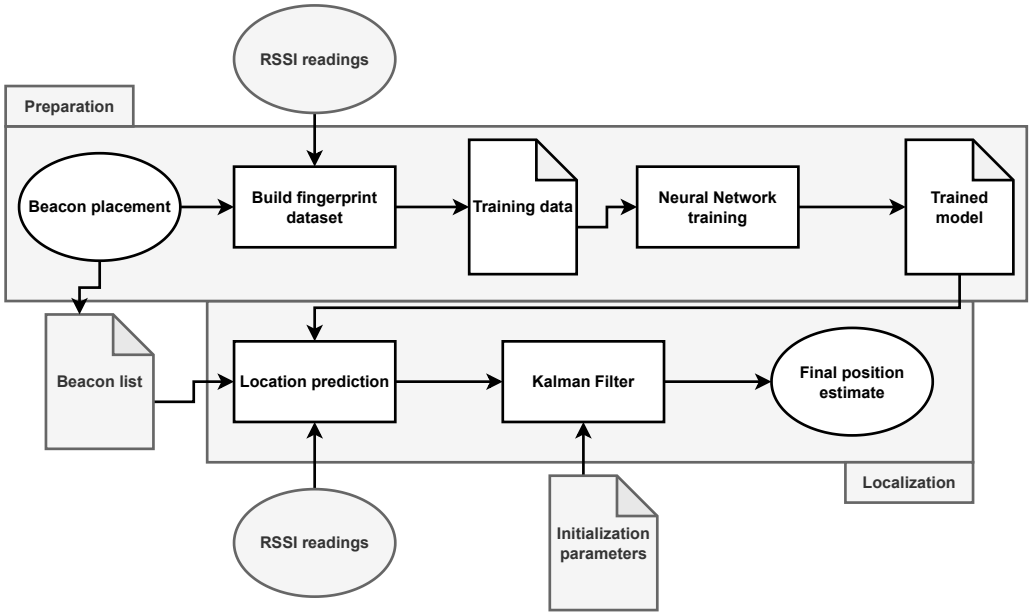

Figure 4.18: The full fingerprint localisation pipeline. A fingerprint dataset is built by taking signal strength (RSSI) measurements at each location in the grid. This dataset is used to train a neural network, that predicts the user location. The prediction is passed through a Kalman Filter to obtain the final position estimate.

For the prediction, the predicted location by the model is also passed through the same 2D Kalman filter used above. The full pipeline from training to localisation can be seen in Figure 4.18.

We observed a mean positioning error of 157.07cm, very close to our other methods. The time taken for each prediction is 1ms, though it should be noted that unlike the other method, only one measurement per sensor is needed instead of 3, thus there is no need to wait for bluetooth between measurements. In any case, both methods are fast enough to react to the bluetooth devices advertising in real time, so the performance difference is less meaningful.

As for the practicality of the fingerprint method, initial deployment is more time-consuming, as it requires hundreds of measurements and retraining of the model for each new location of deployment. By comparison, the multilateration method requires 2 measurements per beacon and some calculation for the path loss formula, though that again needs to be repeated per location for optimal accuracy. Another restriction of the fingerprint method, is that it cannot provide any usable prediction if the location of the target is outside of the grid. The multilateration method does not have such restriction, though the accuracy will of course be lower as all beacons will be in the same direction with respect to the target.

UC-0: Reimplementation of MAO using VDX.

As a last step in the evaluation of our VDX implementation, we reimplemented MAO using VDX as the core of the consensus system. The resulting system was named DiMOS (Distributed Metrics Observation System). It consists of the same MAO executor used in Chapters 2 and 3, and a new voting system implementing the DCC logic using VDX. The architecture of this implementation can be seen in Figure 4.19. This reimplementation shows that VDX is compatible with all prior MAO and DCC work, so all prior results can be reproduced using the VDX-based DiMOS in place of MAO.

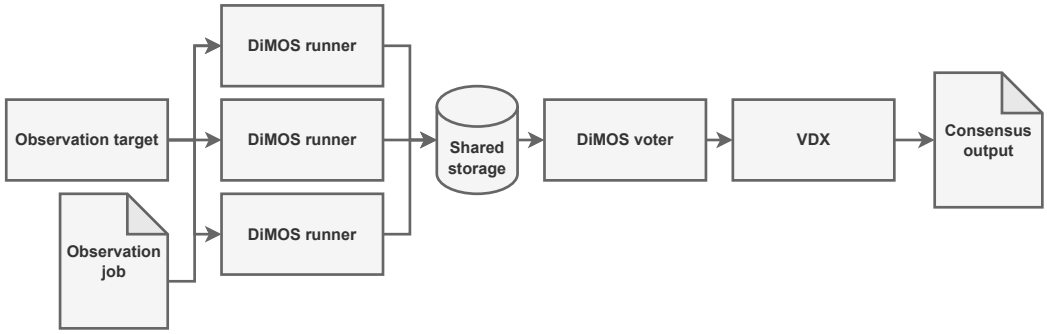

Figure 4.19: DiMOS: The reimplementation of MAO using VDX.

4.8 Findings and Discussion

Evaluating AVOC and VDX yielded some interesting results. First, the light sensor experiment to evaluate the time-series performance of history-based voting. We observe that both the clustering component of AVOC and the history component from the HYBRID algorithm (that is part of AVOC) are important in removing faulty modules and improving the accuracy of results. It is also an important finding that clustering alone is a better voter than distance-based weighted averaging, the typical method for reconciling numeric data which was used as one of the points of comparison in the experiment. This is because clustering is able to remove faulty modules, while distance-based weighted averaging is not. This is useful in scenarios where history is not utilized, due to the system being short-lived or memory constraints. The extra computation in any case is not significant, as shown above, due to the small amount of data involved in each round of voting, even for a large number of modules. Worth noting is the convergence speedup AVOC yields over the state-of-the-art Hybrid algorithm. The speedup from 8 rounds to 2 would not be significant in a system that votes 10 times per second, as the one in our experiment, as the difference would last less than a second. However, it would be significant in a system with a longer period between voting rounds, such as a hypothetical system that evaluates satellite imagery once every 2 weeks. In such an example, without AVOC, the system would carry the errors forward for almost 4 months, while AVOC would resolve them already by the second round. Thus, AVOC has a clearer advantage when history-based voting is needed, but there is a scarcity of historical data.

The first BLE experiment showed that noisy and unstable data is unsuitable for clustering-based methods or history-based methods. In such examples averaging works best, though as we examined in the third experiment, the correct approach would be to filter the data or use other smoothing algorithms first. Our latest positioning experiment presents both a comparison against the state of the art, as well as an approach for combining our methods with the state of the art in data fusion. We observe that our approach can speed up the more inefficient part of the process, which is optimising the multilateration, as well as slightly improving the accuracy of the results. We argue that with enough ease of deployment, this method can be applied in real-world scenarios without much effort, and can be used to improve the accuracy of existing systems, with only software changes. This ease of deployment is exactly why we developed VDX, which was used in all our experiments to implement the different voting methods used.

4.9 Summary

In this chapter, we presented a detailed view of the voting aspect of our multiperspective observation methodology. We began with a study of the state-of-the-art history-aware voting algorithms. We then build upon the state of the art in two ways.

We first introduce an extension of the existing algorithms, AVOC. Our new algorithm improves the performance of the voting system during its bootstrapping phase, or when the data input is equivalently unreliable, such as in the presence of transient errors.

We also presented our effort in formalizing the design of software-defined voters using our VDX specification. We demonstrate VDX and how it can be used to employ the existing algorithms as well as our own AVOC strategy. VDX is also used as the implementation of DCC in our latest DiMOS prototype, completing the loop and integrating all the engineering aspects of this work.

We finally proceed by investigating 3 IoT-based use cases, to evaluate the DCC process and by extension VDX and AVOC in a real-time scenario, expanding from the batch processing we investigated in the previous chapters. We demonstrate how different voting approaches can be beneficial in different scenarios and how there is no one-size-fits-all solution.

Chapter 5

Conclusion

In this work we presented a novel method of managing data in collaborative environments. We coined this method multiperspective observation. Here we will summarize the contributions of this work, present the answers to our research questions, and discuss the perspectives for future work.

5.1 Summary by Chapter

This work is thematically divided into two parts, that combined form the multiperspective observation methodology. The first part is the data acquisition aspect, and the second part is the data verification and merging aspect.

In Chapter 2, we present the data acquisition aspect of the work. We began by introducing our MAO framework, the data collection system used for a large portion of this work. We then used MAO to collect publicly available metadata for Docker images over the period of a year, and used this data as a starting point for conducting an analysis of heterogeneous hardware support in Docker images. We concluded that specialized hardware support was not yet commonplace in the Docker ecosystem, but it is our belief that as heterogeneous hardware becomes more common in practice, this trend will shift. With this in mind, we presented an augmented metadata schema for Docker images, that is aware of hardware pass-through, and built a wrapper tool for the Docker runtime to apply it in practice. We then presented our work in integrating MAO with the SDSC's Renku platform, in an effort to support more complex pipelines in data processing with reproducible execution and tracking of data provenance. This work served as the first use-case for both our MAO framework, and to point out the significance of systematically monitoring public data sources. Over the course of monitoring, this early version of MAO had no built-in support of our DCC methodology, and the testbed federation of nodes was built exclusively on one private cloud, operated by the ZHAW. This meant that outages affected both nodes simultaneously, leading to a small number of missing days in the dataset. This served as further evidence for the need of collaborative monitoring solutions, the main motivation behind our DCC methodology.

In Chapter 3, we begin our investigation of methods of resolving these faults using our collaborative methodology. We discuss the main issues in data quality, and how our methodology is designed to address them. We also present the system design decision and their constraints for implementing our methodology in practice. We presented our DCC methodology, the name we assigned to the end-to-end approach in multiperspective data management. This comprises reproducible or portable data collection workflows, the accumulation and aggregation of data, the resolution of conflicts using methods such as outlier detection and voting, and finally the publishing or propagation of the data as the new ground

truth. We evaluated this method using our MAO testbed federation and a combination of real and synthetic data. We observed that the performance impact of such a system is not significant, and generally the networking overhead involved in moving data between nodes was greater than the time needed to reconcile the data. We also observe the effectiveness of our voting-based strategy in resolving injected errors by measuring whether the system can recover to the original value. We observed that voting-based approaches perform well in most scenarios, further motivating us to study them in more detail.

In Chapter 4, we do a deep-dive into the state-of-the-art in numeric voting algorithms, focusing on the historical component, which fits both real-time systems and long-running monitoring projects that produce time-series-like output, like our metadata datasets. We continue by proposing an extension to the existing approaches by combining them with a clustering step to form AVOC, an algorithm meant to converge faster, making it suitable in scenarios where we need high accuracy even when the history is still short. We also collect the exiting algorithms and AVOC, along with other methods such as averaging and simple majority voting into a generic specification for software-defined voters, VDX. VDX was developed with all the previous components investigated in prior chapters in mind, meaning we could also integrate it into a successor software to MAO, called DiMOS. We continue the chapter by validating VDX and applying the different voting strategies to 3 IoT-based scenarios. We observed that AVOC outperforms the other algorithms in scenarios with continuous monitoring of sensors. By studying the two localisation-based scenarios we found that there is no perfect voting strategy however, thus confirming that a customizable solution (such as VDX) is needed to bridge the use cases under a unified format.

5.2 Summary of Contributions

The main contributions of this work can be concretely summarized as follows:

Our first contribution is our software prototype, DiMOS, which is a lightweight end-to-end implementation of our multiperspective observation strategy for datasets stored within git repositories and workflows implemented as Docker containers. DiMOS is a successor to our previous MAO framework, but without the peer-to-peer networking and built-in scheduler. It integrates much of the output of our work, such as a DCC system implemented using VDX, and an example tool which is a port of our Docker metadata crawler, used to collect data for our Docker hardware analysis in Chapter 3.

Second, we contributed an integration with Renku and MAO. Even though MAO as a software is discontinued, DiMOS can be used as a drop-in alternative in this integration. Combining the data acquisition aspect of DiMOS with the reproducible workflows of Renku, we demonstrated the possibility of building an end-to-end data management solution. The VDX capabilities of DiMOS can be used to further this integration, though this will be discussed in more detail in the future work section.

Third, we conducted a study of hardware support in Docker images. We contributed a proposed metadata schema that is hardware-aware and can be used to ease the configuration of Docker containers. In addition, we contributed HDocker, a software wrapper for Docker that uses our augmented metadata to enable hardware pass-through.

Further, we contributed DCC, a methodology for merging and conflict resolution in multiperspective data. This strategy was verified using our MAO testbed. As an extension to this, we created VDX, our generic specification for software-defined voters. We provide the specification itself along with a reference implementation and examples that use it for different

scenarios (including our IoT scenarios and a DiMOS prototype) as open-source software. As part of our work on VDX we also contributed AVOC, an algorithm that combines the best of the existing algorithms into a single voting strategy that is suitable for both long-running monitoring systems and real-time systems.

5.3 Research Questions

To further summarize the conclusions of this work, we will now discuss our answers to the research questions posed in the introduction.

- RQ 1: How can reproducible or portable workflows be utilized to obtain multiple perspectives on data?

  We have shown three approaches for obtaining multiperspective data. First, we demonstrated the use of portable software, in the form of Docker containers, distributed among members of a federation, with the data managed in a distributed version control system. Second, a reproducible data science workflow in Renku, augmented by a data acquisition framework, can achieve the same result. Additionally, we investigated the scenario of sensor fusion for observation from multiple perspectives.

- RQ 2: How can crowdsourced or decentrally provided insights be utilized in practice?

  Crowdsourced data, as produced by a multiperspective observation system such as a MAO/DiMOS federation or the collaborators in a Renku project, can be more complete and reliable than data observed by a single authority. We demonstrated its use in conducting our study of the Docker ecosystem, where the long-term observation effort gave us a better understanding of trends within that community.

- RQ 3: Can multiperspective data be aggregated/merged to resolve conflicts, restore data loss and/or improve the reliability of a dataset?

  We developed our end-to-end DCC strategy, formalized within the VDX specification and implemented both in DiMOS and our IoT experiments. We showed that such a system can provide more complete datasets and improve accuracy in the presence of unreliable measurements, without introducing significant performance degradation.

- RQ 4: Can multiperspective data reconciliation be adapted to the needs of real-time systems?

  As shown in our IoT experiments, the execution of VDX-based voting systems can be used in a real-time system without slowdown. We demonstrated the advantages this approach has for the quality of the output data.

5.4 Perspectives for Future Work

There are many interesting avenues for further work that are opened by this work. We will now discuss some of them.

A large-scale collaborative experiment utilizing the end-to-end functionalities of DiMOS and VDX is a natural next step. The integration with Renku is also a possible addition to this. In our integration with this platform, we only used our MAO framework for data acquisition, and then performed the data science workflow entirely on Renku. Now that VDX has been developed and integrated into DiMOS, a further step can be used, to merge the

branches of a Renku project using the voting capabilities discussed. The performance and data quality effects of this integrated methodology can then be studied at a much larger scale.

As an extension of our study of the Docker ecosystem, further microservice ecosystems can be studied in a similar fashion. This was one of the motivating use cases of the MAO federation, and as such an informal attempt at co-ordinating this research effort exists within the MAO-MAO project[151]. We have shown the Dockerhub collector, as well as the Artifacthub collector outputs in Chapter 3, and further tools can be integrated to further the study of public software artefacts in the cloud-native community.

Another possibility is the extension of the DCC methodology beyond the limits of VDX. Though it is our belief that purely numeric voting has been exhausted, other methods of data reconciliation are available. We have touched on clustering and outlier detection in this work, though more advanced data cleansing methods based on machine learning or neural networks may be possible. In addition, our binary data categorization and numeric and non-numeric in VDX can be extended for more complex and diverse data types to allow for more complex voting strategies.

In conclusion, we believe that multiperspective observation is a promising approach to data management, and that the work presented in this thesis is a step towards its practical implementation. We hope that the work presented here will be useful to the research community, and that it will inspire further work to extend our methods and apply them to new domains.

Appendix A

Publications

2019

Panagiotis Gkikopoulos:
Data Distribution and Exploitation in a Global Microservice Artefact Observatory.
SERVICES 2019: 319-322, IEEE World Congress on Services, 2019
doi: https://doi.org/10.1109/SERVICES.2019.00089

2020

Panagiotis Gkikopoulos, Josef Spillner, Valerio Schiavoni:
Monitoring Data Distribution and Exploitation in a Global-Scale Microservice Artefact Observatory.
CoRR abs/2006.01514 (2020)
https://arxiv.org/pdf/2006.01514.pdf

Josef Spillner, Panagiotis Gkikopoulos, Alina Buzachis, Massimo Villari:
Rule-Based Resource Matchmaking for Composite Application Deployments across IoT-Fog-Cloud Continuums.
UCC 2020: 336-341, 13th IEEE/ACM International Conference on Utility and Cloud Computing, 2020
doi: https://doi.org/10.1109/UCC48980.2020.00053

Panagiotis Gkikopoulos, Cristian Mateos, Josef Spillner, and Alfredo Teyseyre:
Given 2n Eyeballs, All Quality Flaws Are Shallow.
In Proceedings of the 21st International Middleware Conference Demos and Posters (Middleware '20 Demos and Posters). Association for Computing Machinery, New York, NY, USA, 3–4. 2020
doi: https://doi.org/10.1145/3429358.3429371

2021

Panagiotis Gkikopoulos, Valerio Schiavoni, Josef Spillner:
Analysis and Improvement of Heterogeneous Hardware Support in Docker

Images.
In: Matos M., Greve F. (eds) Distributed Applications and Interoperable Systems. DAIS 2021. Lecture Notes in Computer Science, vol 12718. Springer, Cham., 21st International Conference on Distributed Applications and Interoperable Systems, 2021
doi: https://doi.org/10.1007/978-3-030-78198-9_9

2022

Panagiotis Gkikopoulos, Valerio Schiavoni, Josef Spillner:
Decentralised Data Quality Control in Ground Truth Production for Autonomic Decisions
in IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 10, pp. 2416-2427, 1 Oct. 2022,
doi: https://doi.org/10.1109/TPDS.2022.3142967.

Josef Spillner, Panagiotis Gkikopoulos, Pamela Delgado, Christine Choirat:
Towards reproducible software studies with MAO and Renku
SoftwareX, Volume 17, 2022, 100947, ISSN 2352-7110,
doi: https://doi.org/10.1016/j.softx.2021.100947

Panagiotis Gkikopoulos, Peter Kropf, Valerio Schiavoni, and Josef Spillner:
AVOC: History-Aware Data Fusion for Reliable IoT Analytics.
In 23rd International Middleware Conference (Middleware '22 Industrial Track), November 7–11, 2022, Quebec, QC, Canada. ACM, New York, NY, USA, 7 pages.
doi: https://doi.org/10.1145/3564695.3564772

2023

Oliver Cvetkovski, Panagiotis Gkikopoulos, Josef Spillner:
Digitalisation in Shopping: An IoT and Smart Applications Perspective.
In: Klymash, M., Luntovskyy, A., Beshley, M., Melnyk, I., Schill, A. (eds) Emerging Networking in the Digital Transformation Age. TCSET 2022. Lecture Notes in Electrical Engineering, vol 965. Springer, Cham, 2023
doi: https://doi.org/10.1007/978-3-031-24963-1_11

Mehmet Cihan Sakman, Panagiotis Gkikopoulos, Francesco Martella, Massimo Villari, Josef Spillner:
Indoor Navigation for Personalised Shopping: A Real-Tech Feasibility Study.
In Proceedings of the 20th International Conference on Smart Business Technologies - ICSBT; 2023; ISBN 978-989-758-667-5; ISSN 2184-772X, SciTePress, pages 43-53.
doi: https://doi.org/10.5220/0012085100003552
Recipient of "Best Student Paper" award, ICSBT '23

Appendix B

Datasets and Software Artefacts

2019

Panagiotis Gkikopoulos, Piyush Harsh:
APPUiO PaaS container cloud tracefile.
Zenodo (2019)
doi: https://doi.org/10.5281/zenodo.3540533

2021

Panagiotis Gkikopoulos, Valerio Schiavoni, Josef Spillner:
Heterogeneous Hardware Support in Docker Images.
Zenodo (2021)
doi: https://doi.org/10.5281/zenodo.4550471

2022

Panagiotis Gkikopoulos, Josef Spillner, Valerio Schiavoni:
Data-Centric Consensus code artifacts.
Zenodo (2022)
doi: https://doi.org/10.5281/zenodo.4550471

2023

Panagiotis Gkikopoulos:
VDX: Voting Definition eXtended.
Zenodo (2023)
doi: https://doi.org/10.5281/zenodo.8069916

Mehmet Cihan Sakman, Panagiotis Gkikopoulos, Francesco Martella, Massimo Villari, Josef Spillner:
Smart personalised shopping web prototype.
Zenodo (2023)
doi: https://doi.org/10.5281/zenodo.7859411

Appendix C

Abbreviations

API - Application Programming Interface
ARM - Advanced RISC Machine
AVOC - Accurate Voting with Clustering
AWS - Amazon Web Services
BLE - Bluetooth Low Energy
CLI - Command Line Interface
CPU - Central Processing Unit
CST - Cargo Sous Terrain
CUDA - Compute Unified Device Architecture
CVE - Common Vulnerabilities and Exposures
DBSCAN - Density-Based Spatial Clustering of Applications with Noise
DCC - Data Centric Consensus
DiMOS - Distributed Metrics Observation System
EFI - Extensible Firmware Interface
FPGA - Field Programmable Gate Array
GPU - Graphics Processing Unit
HDocker - Hardware Docker
HISC - Highly Information Sensitive Computing
HPC - High Performance Computing
IoT - Internet of Things
JSON - JavaScript Object Notation
LCD - Liquid Crystal Display
MAO - Microservice Artefact Observatory
ME - Module Elimination Weighted Average
REST - Representational State Transfer
RQ - Research Question
RSSI - Received Signal Strength Indication
SDK - Software Development Kit
SDT - Soft Dynamic Threshold Weighted Average
SEV - Secure Encrypted Virtualization
SGX - Software Guard Extensions
TEE - Trusted Execution Environment
USB - Universal Serial Bus
VDL - Voting Definition Language
VDX - Voting Definition eXtended

Bibliography

[1] J. Wisanmongkol, L. Klinkusoom, T. Sanpechuda, L.-o. Kovavisaruch, and K. Kaemarungsi, "Multipath Mitigation for RSSI-Based Bluetooth Low Energy Localization", *2019 19th International Symposium on Communications and Information Technologies (ISCIT)*, pp. 47–51, 2019.

[2] "FAIR Principles", `https://www.go-fair.org/fair-principles/`, 2021.

[3] "ACM Artifact Review and Badging", `https://www.acm.org/publications/policies/artifact-review-and-badging-current`, 2021.

[4] O. E. Rhazal and M. Tomader, "Study of Smart City Data: Categories and Quality Challenges", *Proceedings of the 4th International Conference on Smart City Applications*, SCA '19, New York, NY, USA, 2019, Association for Computing Machinery.

[5] A. A. Alwan, M. A. Ciupala, A. J. Brimicombe, S. A. Ghorashi, A. Baravalle, and P. Falcarin, "Data quality challenges in large-scale cyber-physical systems: A systematic review", *Information Systems*, 105:101951, 2022.

[6] F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, and M. Allahbakhsh, "Quality Control in Crowdsourcing: A Survey of Quality Attributes, Assessment Techniques, and Assurance Actions", *ACM Comput. Surv.*, 51(1), Association for Computing Machinery, New York, NY, USA, January 2018.

[7] E. R. Jablonski, "The Renku Platform", online: https://renkulab.io/gitlab/team-renku/easyfair/-/blob/master/imgs/renku_component_ecosystem.png, Jun 2020.

[8] J. Spillner, P. Gkikopoulos, P. Delgado, and C. Choirat, "Towards reproducible software studies with MAO and Renku", *SoftwareX*, 17:100947, 2022.

[9] P. Gkikopoulos, "MAO framework source code", `https://github.com/serviceprototypinglab/mao-orchestrator`, 2021.

[10] P. Gkikopoulos, V. Schiavoni, and J. Spillner, "Analysis and Improvement of Heterogeneous Hardware Support in Docker Images", M. Matos and F. Greve, Eds., *Distributed Applications and Interoperable Systems*, pp. 125–142, Cham, 2021, Springer International Publishing.

[11] P. Gkikopoulos, "Dockerhub collector source code", `https://github.com/EcePanos/Dockerhub-Collector`, 2021.

[12] P. Gkikopoulos, V. Schiavoni, and J. Spillner, "Heterogeneous Hardware Support in Docker Images", Feb 2021.

[13] P. Gkikopoulos, V. Schiavoni, and J. Spillner, "Decentralised Data Quality Control in Ground Truth Production for Autonomic Decisions", *IEEE Transactions on Parallel and Distributed Systems*, 33(10):2416–2427, 2022.

[14] P. Gkikopoulos, J. Spillner, and V. Schiavoni, "Data-Centric Consensus code artifacts", https://doi.org/10.5281/zenodo.7288372, January 2022.

[15] P. Gkikopoulos, "ArtifactHub collector source code", `https://github.com/EcePanos/artifacthub_collector`, 2021.

[16] P. Gkikopoulos, "ArtifactHub Dataset", `https://drive.switch.ch/index.php/s/ukUnQQj8eugNz91`, 2021.

[17] P. Gkikopoulos, P. Kropf, V. Schiavoni, and J. Spillner, "AVOC: History-Aware Data Fusion for Reliable IoT Analytics", *Proceedings of the 23rd International Middleware Conference Industrial Track*, Middleware Industrial Track '22, p. 1–7, New York, NY, USA, 2022, Association for Computing Machinery.

[18] P. Gkikopoulos, "VDX source code", `https://github.com/EcePanos/vdx`, 2021.

[19] F. Piccialli and J. J. Jung, "Towards the Internet of Data: Applications, Opportunities and Future Challenges", *Journal of Parallel and Distributed Computing*, 116:1–2, 2018.

[20] M. H. Jarrahi, A. Memariani, and S. Guha, "The Principles of Data-Centric AI", *Commun. ACM*, 66(8):84–92, Association for Computing Machinery, New York, NY, USA, jul 2023.

[21] I. O. for Standardization, "Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Data quality model", Standard, International Organization for Standardization, Geneva, CH, December 2008.

[22] L. Ehrlinger and W. Wöß, "A Survey of Data Quality Measurement and Monitoring Tools", *Frontiers in Big Data*, 5, 2022.

[23] N. Chauhan and S. P. Tripathi, "QoS Aware Replica Control Strategies for Distributed Real Time Database Management System", *Wirel. Pers. Commun.*, 104(2):739–752, Kluwer Academic Publishers, USA, jan 2019.

[24] S. Melzer, H. Peukert, H. Wang, and S. Thiemann, "Model-based Development of a Federated Database Infrastructure to support the Usability of Cross-Domain Information Systems", *2022 IEEE International Systems Conference (SysCon)*, pp. 1–8, 2022.

[25] E. Màrquez-Solís, E. Pérez-Cortès, M. Lopez-Guerrero, and A. G. Medrano-Chàvez, "ClubBT: Leveraging BitTorrent for Content Distribution in Ad-Hoc Social Networks", *2018 IEEE 10th Latin-American Conference on Communications (LATINCOM)*, pp. 1–6, 2018.

[26] Q. Wei, B. Li, W. Chang, Z. Jia, Z. Shen, and Z. Shao, "A Survey of Blockchain Data Management Systems", *ACM Trans. Embed. Comput. Syst.*, 21(3), Association for Computing Machinery, New York, NY, USA, may 2022.

[27] M. Firdaus, S. Rahmadika, and K.-H. Rhee, "Decentralized Trusted Data Sharing Management on Internet of Vehicle Edge Computing (IoVEC) Networks Using Consortium Blockchain", *Sensors*, 21(7):2410, MDPI AG, Mar 2021.

[28] C. Costa and L. Murta, "Version Control in Distributed Software Development: A Systematic Mapping Study", *2013 IEEE 8th International Conference on Global Software Engineering*, pp. 90–99, 2013.

[29] P. Singh, *Airflow*, pp. 67–84, Apress, Berkeley, CA, 2019.

[30] "Dagster", `https://dagster.io/`, 2021.

[31] "Kubeflow", `https://www.kubeflow.org/`, August 2023.

[32] Y. Baba and H. Kashima, "Statistical Quality Estimation for General Crowdsourcing Tasks", *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, p. 554–562, New York, NY, USA, 2013, Association for Computing Machinery.

[33] T. D. Nguyen, J. Park, S. Kim, H. Park, and G. Lee, "Automatically Improving Image Quality Using Tensor Voting", *Neural Comput. Appl.*, 20(7):1017–1026, Springer-Verlag, Berlin, Heidelberg, oct 2011.

[34] A. Torres-Echeverría, S. Martorell, and H. Thompson, "Multi-objective optimization of design and testing of safety instrumented systems with MooN voting architectures using a genetic algorithm", *Reliability Engineering & System Safety*, 106:45–60, 2012.

[35] W.-T. Tsai, Y. Chen, D. Zhang, and H. Huang, "Voting multi-dimensional data with deviations for Web services under group testing", *25th IEEE International Conference on Distributed Computing Systems Workshops*, pp. 65–71, 2005.

[36] Y.-W. Leung, "Maximum likelihood voting for fault-tolerant software with finite output-space", *IEEE Transactions on Reliability*, 44(3):419–427, 1995.

[37] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems*, pp. 382–401, July 1982.

[38] L. Lamport et al., "Paxos made simple", *ACM Sigact News*, 32(4):18–25, 2001.

[39] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm", *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, p. 305–320, USA, 2014, USENIX Association.

[40] M. Hosseinzadeh, E. Azhir, O. H. Ahmed, M. Y. Ghafour, S. H. Ahmed, A. M. Rahmani, and B. Vo, "Data cleansing mechanisms and approaches for big data analytics: a systematic study", *Journal of Ambient Intelligence and Humanized Computing*, 14(1):99–111, Jan 2023.

[41] Y. Zheng, "Methodologies for Cross-Domain Data Fusion: An Overview", *IEEE Transactions on Big Data*, 1(1):16–34, 2015.

[42] P. Tavana, M. Akraminia, A. Koochari, and A. Bagherifard, "An efficient ensemble method for detecting spinal curvature type using deep transfer learning and soft voting classifier", *Expert Systems with Applications*, 213:119290, 2023.

[43] M. A. Kassab, H. S. Taha, S. A. Shedied, and A. Maher, "A novel voting algorithm for redundant aircraft sensors", *Proceeding of the 11th World Congress on Intelligent Control and Automation*, pp. 3741–3746, 2014.

[44] M. R. Boukhari, A. Chaibet, M. Boukhnifer, and S. Glaser, "Voting algorithm approach for autonomous vehicle safe driving", *2018 IEEE International Conference on Industrial Technology (ICIT)*, pp. 327–332, 2018.

[45] S. S. Feger and P. W. Woźniak, "Reproducibility: A Researcher-Centered Definition", *Multimodal Technologies and Interaction*, 6(2), 2022.

[46] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, Today, and Tomorrow", M. Mazzara and B. Meyer, Eds., *Present and Ulterior Software Engineering*, pp. 195–216, Springer International Publishing, Cham, 2017.

[47] J. Thones, "Microservices", *IEEE Software*, 32(1):116, 2015.

[48] "Docker Hub", `https://hub.docker.com/`, 2021.

[49] K. S.-P. Chang and S. J. Fink, "Visualizing serverless cloud application logs for program understanding", *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 261–265, 2017.

[50] S.-P. Ma, C.-Y. Fan, Y. Chuang, W.-T. Lee, S.-J. Lee, and N.-L. Hsueh, "Using Service Dependency Graph to Analyze and Test Microservices", *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 02, pp. 81–86, 2018.

[51] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems", *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pp. 3–18, New York, NY, USA, 2019, ACM.

[52] M. Viggiato, R. Terra, H. Rocha, M. T. Valente, and E. Figueiredo, "Microservices in Practice: A Survey Study", *CoRR*, abs/1808.04836, 2018.

[53] J. Spillner, "Quality Assessment and Improvement of Helm Charts for Kubernetes-Based Cloud Applications", *CoRR*, abs/1901.00644, 2019.

[54] J. Spillner, "Quantitative Analysis of Cloud Function Evolution in the AWS Serverless Application Repository", *CoRR*, abs/1905.04800, 2019.

[55] D. Petcu, "Portability and interoperability between clouds: challenges and case study", *ECSBI*, pp. 62–74, Springer, 2011.

[56] B. Di Martino, "Applications portability and services interoperability among multiple clouds", *IEEE Cloud Computing*, 1(1):74–77, 2014.

[57] J. Bozman and G. Chen, "Cloud computing: The need for portability and interoperability", *IDC Executive Insights*, 2010.

[58] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "TOSCA: Portable Automated Deployment and Management of Cloud Applications", *Advanced Web Services*, pp. 527–549, Springer, 2014.

[59] J. Carrasco, F. Durán, and E. Pimentel, "Live migration of trans-cloud applications", *Comput. Stand. Interfaces*, 69:103392, 2020.

[60] S. Murphy and A. Edmonds, "Realizing Edge Computing Connectivity with Open Virtual Networking", *IEEE/ACM Intl. Conf. Utility and Cloud Computing Companion, UCC Companion 2018*, pp. 128–133, 2018.

[61] D. Schinianakis, R. Trapero, D. S. Michalopoulos, and B. G. Crespo, "Security Considerations in 5G Networks: A Slice-Aware Trust Zone Approach", *IEEE WCNC*, pp. 1–8, 2019.

[62] T. Yeh, H. Chen, and J. Chou, "KubeShare: A Framework to Manage GPUs as First-Class and Shared Resources in Container Cloud", *29th Intl. Symp. High-Performance Parallel and Distributed Computing*, pp. 173–184, ACM, 2020.

[63] M. Shepovalov and V. Akella, "FPGA and GPU-based acceleration of ML workloads on Amazon cloud - A case study using gradient boosted decision tree library", *Integration*, 70:1–9, 2020.

[64] N. Tarafdar, N. Eskandari, T. Lin, and P. Chow, "Designing for FPGAs in the Cloud", *IEEE Design & Test*, 35(1):23–29, 2018.

[65] J. Ren, Y. Qi, Y. Dai, X. Yu, and Y. Shi, "Nosv: A lightweight nested-virtualization VMM for hosting high performance computing on cloud", *J. Syst. Softw.*, 124:137–152, 2017.

[66] R. Herardian, "The Soft Underbelly of Cloud Security", *IEEE Security & Privacy*, 17(3):90–93, 2019.

[67] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud", *Journal of network and computer applications*, 36(1):42–57, Elsevier, 2013.

[68] S. Johnson, D. Rizzo, P. Ranganathan, J. McCune, and R. Ho, "Titan: enabling a transparent silicon root of trust for Cloud", *Hot Chips: A Symposium on High Performance Chips*, 2018.

[69] L. Coppolino, S. D'Antonio, G. Mazzeo, and L. Romano, "A comprehensive survey of hardware-assisted security: From the edge to the cloud", *Internet of Things*, 6, 2019.

[70] J. Kebbedies, J. Spillner, I. Braun, and A. Schill, "Conceptualized Policy Design for User-Regulated Trusted Clouds", *8th IEEE/ACM UCC*, pp. 601–606, 2015.

[71] A. Brito, C. Fetzer, S. Köpsell, P. Pietzuch, M. Pasin, P. Felber, K. Fonseca, M. Rosa, L. Gomes, R. Riella, C. Prado, L. F. Rust, D. E. Lucani, M. Sipos, L. Nagy, and M. Fehér, "Secure end-to-end processing of smart metering data", *Journal of Cloud Computing*, 8(1):19, 2019.

[72] "Introducing Google Cloud Confidential Computing with Confidential VMs", https://cloud.google.com/blog/products/identity-security/introducing-google-cloud-confidential-computing-with-confidential-vms, 2020.

[73] "Confidential computing on Azure", https://docs.microsoft.com/en-us/azure/confidential-computing/overview, 2020.

[74] V. Costan and S. Devadas, "Intel SGX Explained.", *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[75] D. Kaplan, J. Powell, and T. Woller, "AMD memory encryption", *White paper*, 2016.

[76] S. Pinto and N. Santos, "Demystifying ARM TrustZone: A comprehensive survey", *ACM Computing Surveys (CSUR)*, 51(6):1–36, 2019.

[77] J. Amacher and V. Schiavoni, "On the performance of ARM TrustZone", *IFIP Intl. Conf. DAIS*, pp. 133–151, Springer, 2019.

[78] "IBM Cloud (formerly Bluemix)", https://console.ng.bluemix.net/, 2020.

[79] "Amazon EC2 Container Service", https://aws.amazon.com/ecs/, 2020.

[80] "AWS Container Service", https://azure.microsoft.com/en-us/services/container-service/, 2020.

[81] "Google Container Engine", https://cloud.google.com/container-engine/, 2020.

[82] "AWS Nitro Enclaves", https://aws.amazon.com/ec2/nitro/nitro-enclaves/, 2020.

[83] "NVIDIA Docker: GPU Server Application Deployment Made Easy", https://developer.nvidia.com/blog/nvidia-docker-gpu-server-application-deployment-made-easy/, 2020.

[84] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure Linux Containers with Intel SGX", *12th USENIX Conf. OSDI*, p. 689–703, 2016.

[85] "SCONTAIN Homepage", https://scontain.com/, 2020.

[86] "Graphene Secure Container Environment", https://github.com/oscarlab/graphene/tree/master/Tools, 2020.

[87] R. Florin and R. Ionut, "FPGA based architecture for securing IoT with blockchain", *Intl. Conf. Speech Technology and Human-Computer Dialogue, SpeD'19*, pp. 1–8, IEEE, 2019.

[88] K. Cho, H. Lee, K. Bang, and S. Kim, "Possibility of HPC Application on Cloud Infrastructure by Container Cluster", *IEEE Intl. Conf. CSE and Computational Science and IEEE Intl. Conf. EUC*, pp. 266–271, 2019.

[89] R. Shu, X. Gu, and W. Enck, "A Study of Security Vulnerabilities on Docker Hub", *Proc. 7th ACM CODASPY*, p. 269–280, 2017.

[90] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall, "An Empirical Analysis of the Docker Container Ecosystem on GitHub", *IEEE/ACM 14th Intl. Conf. Mining Software Repositories (MSR)*, pp. 323–333, 2017.

[91] "Image Manifest Version 2, Schema 2", `https://docs.docker.com/registry/spec/manifest-v2-2/`, 2021.

[92] M. Villari, M. Fazio, S. Dustdar, O. Rana, D. N. Jha, and R. Ranjan, "Osmosis: The Osmotic Computing Platform for Microelements in the Cloud, Edge, and Internet of Things", *IEEE Computer*, 52(8):14–26, 2019.

[93] N. Zhao, V. Tarasov, H. Albahar, A. Anwar, L. Rupprecht, D. Skourtis, A. S. Warke, M. Mohamed, and A. R. Butt, "Large-Scale Analysis of the Docker Hub Dataset", *2019 IEEE Intl. Conf. Cluster Computing, CLUSTER*, pp. 1–10, 2019.

[94] A. Byrne, S. Nadgowda, and A. Coskun, "ACE: Just-in-time Serverless Software Component Discovery Through Approximate Concrete Execution", *Proc. Middleware Workshops / Sixth Intl. Workshop on Serverless Computing (WoSC6)*, 2020.

[95] Y. Mao, J. Oak, A. Pompili, D. Beer, T. Han, and P. Hu, "DRAPS: Dynamic and Resource-Aware Placement Scheme for Docker Containers in a Heterogeneous Cluster", *CoRR*, abs/1805.08598, 2018.

[96] A. R. Portabales and M. L. Nores, "Dockemu: Extension of a Scalable Network Simulation Framework based on Docker and NS3 to Cover IoT Scenarios", *Proc. 8th Intl. Conf. Simulation and Modeling Methodologies, Technologies and Applications, SIMULTECH'18*, pp. 175–182, SciTePress, 2018.

[97] C. X. Tian, A. Pan, and Y. C. Tay, "ConHub: A Metadata Management System for Docker Containers", *Proc. 25th ACM Intl. Conf. Information and Knowledge Management, CIKM'16*, pp. 2453–2455, 2016.

[98] A. B. Ayed, J. Subercaze, F. Laforest, T. Chaari, W. Louati, and A. H. Kacem, "Docker2rdf: Lifting the docker registry hub into rdf", *2017 IEEE World Congress on Services (SERVICES)*, pp. 36–39, IEEE, 2017.

[99] A. Brogi, D. Neri, and J. Soldani, "DockerFinder: multi-attribute search of Docker images", *IEEE Int. Conf. Cloud Engineering (IC2E)*, 2017.

[100] J. Spillner, P. Gkikopoulos, A. Buzachis, and M. Villari, "Rule-Based Resource Matchmaking for Composite Application Deployments across IoT-Fog-Cloud Continuums", *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pp. 336–341, 2020.

[101] J. Cleare and C. Iacob, "GemChecker: Reporting on the Status of Gems in Ruby on Rails Projects", *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*, pp. 700–704, IEEE Computer Society, 2018.

[102] P. Chapman, D. Xu, L. Deng, and Y. Xiong, "Deviant: A Mutation Testing Tool for Solidity Smart Contracts", *IEEE International Conference on Blockchain, Blockchain 2019, Atlanta, GA, USA, July 14-17, 2019*, pp. 319–324, IEEE, 2019.

[103] M. A. Oumaziz, J. Falleri, X. Blanc, T. F. Bissyandé, and J. Klein, "Handling Duplicates in Dockerfiles Families: Learning from Experts", *2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019*, pp. 524–535, IEEE, 2019.

[104] O. Al-Debagy and P. Martinek, "A Metrics Framework for Evaluating Microservices Architecture Designs", *J. Web Eng.*, 19(3-4):341–370, 2020.

[105] M. Bravetti, S. Giallorenzo, J. Mauro, I. Talevi, and G. Zavattaro, "A Formal Approach to Microservice Architecture Deployment", A. Bucchiarone, N. Dragoni, S. Dustdar, P. Lago, M. Mazzara, V. Rivera, and A. Sadovykh, Eds., *Microservices, Science and Engineering*, pp. 183–208, Springer, 2020.

[106] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "Unveiling the Hardware and Software Implications of Microservices in Cloud and Edge Systems", *IEEE Micro*, 40(3):10–19, 2020.

[107] S. Haselbock, R. Weinreich, and G. Buchgeher, "An Expert Interview Study on Areas of Microservice Design", *11th IEEE Conference on Service-Oriented Computing and Applications, SOCA 2018, Paris, France, November 20-22, 2018*, pp. 137–144, IEEE Computer Society, 2018.

[108] K. Wist, M. Helsem, and D. Gligoroski, "Vulnerability Analysis of 2500 Docker Hub Images", *CoRR*, abs/2006.02932, 2020.

[109] D. Shadija, M. Rezai, and R. Hill, "Microservices: Granularity vs. Performance", A. Anjum, A. Sill, G. C. Fox, and Y. Chen, Eds., *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing, UCC 2017, Austin, TX, USA, December 5-8, 2017*, pp. 215–220, ACM, 2017.

[110] M. Malawski, A. Gajek, A. Zima, B. Balis, and K. Figiela, "Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions", *Future Gener. Comput. Syst.*, 110:502–514, 2020.

[111] D. Yu, Y. Jin, Y. Zhang, and X. Zheng, "A survey on security issues in services communication of Microservices-enabled fog applications", *Concurr. Comput. Pract. Exp.*, 31(22), 2019.

[112] S. Apel, F. Hertrampf, and S. Späthe, "Toward a knowledge model focusing on microservices and cloud computing", *Concurr. Comput. Pract. Exp.*, 32(13), 2020.

[113] S. L. Joshi, B. Deshpande, and S. Punnekkat, "Experimental Analysis of Dependency Factors of Software Product Reliability using SonarQube", A. K. Tarhan and A. Coskunçay, Eds., *Joint Proceedings of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM Mensura 2019), Haarlem, The Netherlands, October 7-9, 2019*, volume 2476 of *CEUR Workshop Proceedings*, pp. 130–137, CEUR-WS.org, 2019.

[114] "Prefect", `https://www.prefect.io/`, 2021.

[115] R. D. Peng, "Reproducible research in computational science", *Science*, 334(6060):1226–1227, American Association for the Advancement of Science, 2011.

[116] P. Gkikopoulos, "Data Distribution and Exploitation in a Global Microservice Artefact Observatory", *15th IEEE World Congress on Services (SERVICES)*, pp. 319–322, Milan, Italy, July 2019.

[117] P. Gkikopoulos, J. Spillner, and V. Schiavoni, "Monitoring Data Distribution and Exploitation in a Global-Scale Microservice Artefact Observatory", arχiv:2006.01514, June 2020.

[118] J. Spillner, "Quality Assessment and Improvement of Helm Charts for Kubernetes-Based Cloud Applications", arχiv:1901.00644, January 2019.

[119] J. Spillner, "Quantitative Analysis of Cloud Function Evolution in the AWS Serverless Application Repository", arχiv:1905.04800, May 2019.

[120] I. A. Qasse, J. Spillner, M. A. Talib, and Q. Nasir, "A Study on DApps Characteristics", *2nd IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, August 2020.

[121] M. Müller and M. Rüdisüli, "DQA: Docker Quality Analysis", Docker image: https://hub.docker.com/r/svl7/dqa-mao, June 2020.

[122] N. Sonehara, T. Suzuki, A. Kodate, T. Wakahara, Y. Sakai, Y. Ichifuji, H. Fujii, and H. Yoshii, "Data-Driven Decision-Making in Cyber-Physical Integrated Society", *IEICE Trans. Inf. Syst.*, 102-D(9):1607–1616, 2019.

[123] S. V. Kovalchuk, E. Krotov, P. A. Smirnov, D. A. Nasonov, and A. N. Yakovlev, "Distributed data-driven platform for urgent decision making in cardiological ambulance control", *Future Generation Computer Systems*, 79:144–154, 2018.

[124] E. M. De Oliveira, J. C. Estrella, and S. Reiff-Marganiec, "A Distributed Environment Decision Maker Based on Machine Learning Techniques", *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pp. 108–113, 2017.

[125] Z. Y.-C. Liu, S. Roychowdhury, S. Tarlow, A. Nair, S. Badhe, and T. Shah, "AutoDC: Automated data-centric processing", 2021.

[126] K.-U. Sattler, "Data Quality Dimensions", L. Liu and M. T. Özsu, Eds., *Encyclopedia of Database Systems*, pp. 817–821, Springer New York, New York, NY, 2018.

[127] H. Cui, R. Gu, C. Liu, T. Chen, and J. Yang, "Paxos made transparent", *Proceedings of the 25th Symposium on Operating Systems Principles*, pp. 105–120, 2015.

[128] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: a survey", *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.

[129] K. Mordal, N. Anquetil, J. Laval, A. Serebrenik, B. Vasilescu, and S. Ducasse, "Software quality metrics aggregation in industry", *Journal of Software: Evolution and Process*, 25(10):1117–1135, Wiley Online Library, 2013.

[130] Ø. Netland and A. Skavhaug, "Dependable Cyber-Physical Systems with Redundant Consumer Single-Board Linux Computers", F. Koornneef and C. van Gulijk, Eds., *Computer Safety, Reliability, and Security*, pp. 224–234, Cham, 2015, Springer.

[131] A. Höller, T. Rauter, J. Iber, and C. Kreiner, "Towards Dynamic Software Diversity for Resilient Redundant Embedded Systems", A. Fantechi and P. Pelliccione, Eds., *Software Engineering for Resilient Systems*, pp. 16–30, Cham, 2015, Springer.

[132] P. Fraigniaud, S. Rajsbaum, and C. Travers, "A lower bound on the number of opinions needed for fault-tolerant decentralized run-time monitoring", *J. Appl. Comput. Topol.*, 4(1):141–179, 2020.

[133] S. Omidshafiei, S. Liu, M. Everett, B. T. Lopez, C. Amato, M. Liu, J. P. How, and J. Vian, "Semantic-level decentralized multi-robot decision-making using probabilistic macro-observations", *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 871–878, 2017.

[134] S. Łaskawiec, M. Choraś, R. Kozik, and V. Varadarajan, "Intelligent operator: Machine learning based decision support and explainer for human operators and service providers in the fog, cloud and edge networks", *Journal of Information Security and Applications*, 56:102685, 2021.

[135] "Docker", `https://www.docker.com/`, 2021.

[136] P. Gkikopoulos and J. Spillner, "CIPolicE - Container Image Policy Engine", open source: `https://github.com/serviceprototypinglab/cipolice`, August 2020.

[137] J. Pâris and D. D. E. Long, "Pirogue, a lighter dynamic version of the Raft distributed consensus algorithm", *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–8, 2015.

[138] X. Ge and Q. Han, "A brief survey of recent advances in consensus of sampled-data multi-agent systems", *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pp. 6758–6763, 2016.

[139] W. Yao, J. Ye, R. Murimi, and G. Wang, "A Survey on Consortium Blockchain Consensus Mechanisms", 2021.

[140] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl, "Reaching Approximate Agreement in the Presence of Faults", *J. ACM*, 33(3):499–516, Association for Computing Machinery, New York, NY, USA, May 1986.

[141] "Kubernetes", `https://kubernetes.io/`, 2021.

[142] "etcd", `https://etcd.io/`, 2021.

[143] N. O. Ahmed and B. Bhargava, "From Byzantine Fault-Tolerance to Fault-Avoidance: An Architectural Transformation to Attack and Failure Resiliency", *IEEE Transactions on Cloud Computing*, 8(3):847–860, 2020.

[144] "Docker Swarm", `https://docs.docker.com/engine/swarm/`, 2021.

[145] "Titus", `https://netflix.github.io/titus/`, 2021.

[146] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm", *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pp. 305–319, Philadelphia, PA, June 2014, USENIX Association.

[147] F. Pukelsheim, "The Three Sigma Rule", *The American Statistician*, 48(2):88–91, [American Statistical Association, Taylor & Francis, Ltd.], 1994.

[148] C. Deus and Zhifang Liao, "Statistical consensus method for cluster ensembles", *2010 IEEE International Conference on Progress in Informatics and Computing*, volume 1, pp. 185–189, 2010.

[149] Yizong Cheng, "Mean shift, mode seeking, and clustering", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.

[150] P. Gkikopoulos, V. Schiavoni, and J. Spillner, "Analysis and Improvement of Heterogeneous Hardware Support in Docker Images", M. Matos and F. Greve, Eds., *Distributed Applications and Interoperable Systems*, pp. 125–142, Cham, 2021, Springer.

[151] "MAO-MAO: Microservice Artefact Observatory", `https://mao-mao-research.github.io/`, 2021.

[152] H. C. Mandhare and S. R. Idate, "A comparative study of cluster based outlier detection, distance based outlier detection and density based outlier detection techniques", *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 931–935, 2017.

[153] A. Boukerche, L. Zheng, and O. Alfandi, "Outlier Detection: Methods, Models, and Classification", *ACM Comput. Surv.*, 53(3), Association for Computing Machinery, New York, NY, USA, June 2020.

[154] G. Latif-Shabgahi, J. Bass, and S. Bennett, "A taxonomy for software voting algorithms used in safety-critical systems", *IEEE Transactions on Reliability*, 53(3):319–328, 2004.

[155] A. Fekete, "Asynchronous Approximate Agreement", *Information and Computation*, 115(1):95–124, 1994.

[156] O. Görlitz and S. Staab, *Federated Data Management and Query Optimization for Linked Open Data*, volume 331, pp. 109–137, 02 2011.

[157] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Data Management in Peer-to-Peer Data Integration Systems", *Journal of Intelligent Information Systems - JIIS*, 8, 01 2006.

[158] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A Survey of Distributed Consensus Protocols for Blockchain Networks", *IEEE Communications Surveys Tutorials*, 22(2):1432–1465, 2020.

[159] S. Pahlajani, A. Kshirsagar, and V. Pachghare, "Survey on Private Blockchain Consensus Algorithms", *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, pp. 1–6, 2019.

[160] W. Lin, X. Xu, L. Qi, X. Zhang, W. Dou, and M. R. Khosravi, "A Proof-of-Majority Consensus Protocol for Blockchain-Enabled Collaboration Infrastructure of 5G Network Slice Brokers", *Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, BSCI '20, p. 41–52, New York, NY, USA, 2020, Association for Computing Machinery.

[161] X. Wang, Q. Feng, and J. Chai, "The Research of Consortium Blockchain Dynamic Consensus Based on Data Transaction Evaluation", *2018 11th International Symposium on Computational Intelligence and Design (ISCID)*, volume 02, pp. 214–217, 2018.

[162] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance", *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pp. 173–186, 1999.

[163] B. Umamaheswrarao, P. Seetharamaiah, and S. Phanikumar, "An Optimal Weighted-Average Voting Algorithm for Software Safety Estimation on Safety-Critical Systems", *SIGSOFT Softw. Eng. Notes*, 40(4):1–7, Association for Computing Machinery, New York, NY, USA, July 2015.

[164] G. Latif-Shabgahi, "A novel algorithm for weighted average voting used in fault tolerant computing systems", *Microprocessors and Microsystems*, 28(7):357–361, 2004.

[165] C. Candrian and A. Scherer, "Rise of the machines: Delegating decisions to autonomous AI", *Computers in Human Behavior*, 134:107308, 2022.

[166] Z. Chen, X. Feng, and S. Zhang, "Emotion detection and face recognition of drivers in autonomous vehicles in IoT platform", *Image and Vision Computing*, 128:104569, 2022.

[167] X. Wang and C. Wang, "Time Series Data Cleaning: A Survey", *IEEE Access*, 8:1866–1881, 2020.

[168] N. Zubair, N. A, K. Hebbar, and Y. Simmhan, "Characterizing IoT Data and its Quality for Use", *CoRR*, abs/1906.10497, 2019.

[169] D. Lupton, "'Flawed', 'Cruel' and 'Irresponsible': The Framing of Automated Decision-Making Technologies in the Australian Press", April 2021.

[170] N. U. Okafor, Y. Alghorani, and D. T. Delaney, "Improving Data Quality of Low-cost IoT Sensors in Environmental Monitoring Networks Using Data Fusion and Machine Learning Approach", *ICT Express*, 6(3):220–228, 2020.

[171] S. E. Whang, Y. Roh, H. Song, and J. Lee, "Data Collection and Quality Challenges in Deep Learning: A Data-Centric AI Perspective", *CoRR*, abs/2112.06409, 2021.

[172] Y. Fang, Z. Shan, and W. Wang, "Modeling and Key Technologies of a Data-Driven Smart City System", *IEEE Access*, 9:91244–91258, 2021.

[173] J. Wang, C. Xu, J. Zhang, and R. Zhong, "Big data analytics for intelligent manufacturing systems: A review", *Journal of Manufacturing Systems*, 62:738–752, 2022.

[174] B. P. L. Lau, S. H. Marakkalage, Y. Zhou, N. U. Hassan, C. Yuen, M. Zhang, and U.-X. Tan, "A survey of data fusion in smart city applications", *Information Fusion*, 52:357–374, 2019.

[175] S. Jeong, S. Kim, and J. Kim, "City Data Hub: Implementation of Standard-Based Smart City Data Platform for Interoperability", *Sensors*, 20(23), 2020.

[176] S. Kolozali, M. Bermudez-Edo, N. Farajidavar, P. Barnaghi, F. Gao, M. Intizar Ali, A. Mileo, M. Fischer, T. Iggena, D. Kuemper, and R. Tonjes, "Observing the Pulse of a City: A Smart City Framework for Real-Time Discovery, Federation, and Aggregation of Data Streams", *IEEE Internet of Things Journal*, 6(2):2651–2668, 2019.

[177] G. Latif-Shabgahi, J. Bass, and S. Bennett, "History-based weighted average voter: a novel software voting algorithm for fault-tolerant computer systems", *Proceedings Ninth Euromicro Workshop on Parallel and Distributed Processing*, pp. 402–409, 2001.

[178] M. Das and S. Bhattacharya, "A Modified History Based Weighted Average Voting with Soft-Dynamic Threshold", *2010 International Conference on Advances in Computer Engineering*, pp. 217–222, 2010.

[179] A. Alahmadi and B. Soh, "A hybrid history based weighted voting algorithm for ultra-critical systems", *2012 International Symposium on Communications and Information Technologies (ISCIT)*, pp. 1122–1127, 2012.

[180] D. Bakken, Z. Zhan, C. Jones, and D. Karr, "Middleware support for voting and data fusion", *2001 International Conference on Dependable Systems and Networks*, pp. 453–462, 2001.

[181] "Singer.io", `https://www.singer.io/`, June 2023.

[182] "Node-RED", `https://nodered.org/`, June 2023.

[183] P. Gkikopoulos, P. G. Kropf, V. Schiavoni, and J. Spillner, "AVOC: history-aware data fusion for reliable IoT analytics", K. Zhang, A. Gherbi, and P. Bellavista, Eds., *Proceedings of the 23rd International Middleware Conference: Industrial Track, Middleware 2022, Quebec, Quebec City, Canada, November 7-11, 2022*, pp. 1–7, ACM, 2022.

[184] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, p. 226–231, AAAI Press, 1996.

[185] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[186] D. Pelleg and A. Moore, "X-means: Extending K-means with Efficient Estimation of the Number of Clusters", *In Proceedings of the 17th International Conf. on Machine Learning*, pp. 727–734, Morgan Kaufmann, 2000.

[187] "Wireless VINT Hub", `https://www.phidgets.com/?tier=3&catid=64&pcid=57&prodid=1143`, 2022.

[188] "Light Phidget", `https://www.phidgets.com/?tier=3&catid=8&pcid=6&prodid=707`, 2022.

[189] "Graphic LCD Phidget", https://www.phidgets.com/?tier=3&catid=48&pcid=41&prodid=963, 2022.

[190] "Bluetooth Technology Overview", https://www.bluetooth.com/learn-about-bluetooth/tech-overview/, 2022.

[191] N. Minaei, "Future Transport and Logistics in Smart Cities: Safety and Privacy", *Smart Cities*, pp. 113–142, CRC Press, 2022.

[192] "Cargo Sous Terrain", https://www.cst.ch/, 2022.

[193] "Lego Mindstorms EV3", https://education.lego.com/en-us/products/lego-mindstorms-education-ev3-intelligent-brick/45500, 2022.

[194] Y. Wu, S. Cheng, and X. Yan, "Study on Improved Algorithm of RSSI Correction and Location in Mine-Well Based on Bluetooth Positioning Information", *Proceedings of the 4th International Conference on Computer Science and Application Engineering*, CSAE 2020, New York, NY, USA, 2020, Association for Computing Machinery.

[195] Q. Li, R. Li, K. Ji, and W. Dai, "Kalman Filter and Its Application", *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, pp. 74–77, 2015.

[196] J. J. Moré, "The Levenberg-Marquardt algorithm: Implementation and theory", G. A. Watson, Ed., *Numerical Analysis*, pp. 105–116, Berlin, Heidelberg, 1978, Springer Berlin Heidelberg.

[197] V. Cantón Paterna, A. Calveras Augé, J. Paradells Aspas, and M. A. Pérez Bullones, "A Bluetooth Low Energy Indoor Positioning System with Channel Diversity, Weighted Trilateration and Kalman Filtering", *Sensors*, 17(12), 2017.

[198] C. Zhou, J. Yuan, H. Liu, and J. Qiu, "Bluetooth Indoor Positioning Based on RSSI and Kalman Filter", *Wirel. Pers. Commun.*, 96(3):4115–4130, Kluwer Academic Publishers, USA, oct 2017.

[199] M. Sakman., P. Gkikopoulos., F. Martella., M. Villari., and J. Spillner., "Indoor Navigation for Personalised Shopping: A Real-Tech Feasibility Study", *Proceedings of the 20th International Conference on Smart Business Technologies - ICSBT*, pp. 43–53, INSTICC, SciTePress, 2023.

[200] R. Faragher and R. Harle, "Location Fingerprinting With Bluetooth Low Energy Beacons", *IEEE Journal on Selected Areas in Communications*, 33(11):2418–2428, 2015.