TBA

Marco Anisetti *Senior Member, IEEE,*, Claudio A. Ardagna *Senior Member, IEEE,* Chiara Braghin, Ernesto Damiani *Senior Member, IEEE,*, Antongiacomo Polimeno

Abstract—The conflict between the need of protecting and sharing data is hampering the spread of big data applications. Proper security and privacy assurance is required to protect data owners, while proper data access and sharing are fundamental to implement smart big data solutions. In this context, access control systems assume a central role for balancing the need of data protection and sharing. However, given the software and technological complexity of big data ecosystems, existing solutions are not suitable because they are neither general nor scalable, and do not support a dynamic and collaborative environment. In this paper, we propose an access control system that enforces access to data in a distributed, multi-party big data environment. It is based on data annotations and secure data transformations performed at ingestion time. We show the feasibility of our approach with a case study in a smart city domain using an Apache-based big data engine.
In today's data landscape, the coexistence of data quality and data privacy is critical to support high-value services and pipelines. Our approach seeks to harmonize these objectives by establishing a data governance framework that balances privacy and data quality.

Index Terms—Access Control, Big Data, Data Transformation, Data Ingestion

1 Introduction

The usage of multitenancy coupled with cloud infrastructure represents a paradigm shift in the big data scenario, redefining scalability and efficiency in data analytics. Multitenancy enables multiple users to share resources, such as computing power and storage, optimizing resource utilization and reducing operational costs. Leveraging cloud infrastructure further enhances flexibility and scalability. The flip side of multitenancy is the increased complexity of data governace: the shared model introduces unique security challenges, as tenants may have different security requirements, levels of access, and data sensitivity. Adequate measures such as encryption, access control mechanisms, and data anonymization techniques must be implemented to safeguard data against unauthorized access and ensure compliance with regulatory requirements such as GDPR or HIPAA. As a consequence, achieving a balance between data protection and data quality is crucial, as the removal or alteration of personally identifiable information from datasets to safeguard individuals' privacy can compromise the accuracy of analytics results.

So far, all research endeavors have been concentrated on exploring these two issues separately: on one hand, the concept of data quality, encompassing accuracy, reliability, and suitability, has been investigated to understand the implications in analytical contexts. Although extensively studied, these investigations often prioritize enhancing the quality of source data rather than ensuring data quality throughout the entire processing pipeline, or the integrity of outcomes derived from data. On the other hand, there is a focus on data privacy and security, entailing the protection of confidential information and adherence to rigorous privacy regulations. There are very few solutions that that find a good balance between them since it requires a holistic approach that integrates technological solutions, organizational policies, and ongoing monitoring and adaptation to emerging threats and regulatory changes.

- *M. Anisetti, C.A. Ardagna, E. Damiani, are with the Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy. E. Damiani is also with.*
E-mail: {firstname.lastname}@unimi.it

To this aim, we propose a data governance framework tailored to contemporary data-driven pipelines, which aims to limit the privacy and security risks. The primary objective of this framework is to facilitate the assembly of data processing services, with a central focus on the selection of those services that optimize data quality, while upholding privacy and security requirements.

The key contributions of this study are as follows:

1) each service in the pipeline is tagged with *annotations* to specify data protection requirements expressing transformation on data to enforce data protection, as well as functional specifications on services expressing data manipulations carried out during services execution;
2) the annotated pipeline, called *pipeline template*, now acts as a skeleton, specifying the structure of the pipeline and both the functional and non-functional requirements;
3) the *pipeline instance* is built by instantiating the template with services according to the specified requirements. Our service selection approach focuses on maximizing data quality by retaining the maximum amount of information when applying data protection transformations;
4) The composite selection problem is NP-hard, but we present a parametric heuristic tailored to address the computational complexity. We evaluated the performance and quality of the algorithm by running some experiments on a dataset.

The rest of the paper is organized as follows.

2 System Model and Service Pipeline

~~Big data is highly dependent on cloud-edge computing, which makes extensive use of multitenancy. Multitenancy permits sharing one instance of infrastructures, platforms or applications by multiple tenants to optimize costs. This leads to common scenarios where a service provider offers subscription-based analytics capabilities in the cloud, or a single data lake is accessed by multiple customers. Big data pipelines then mix data and services which belong to various organizations, posing a serious risk of potential privacy and security violations. We propose a data governance framework tailored to contemporary data-driven pipelines, which aims to limit the privacy and security risks. The primary objective of this framework is to facilitate the assembly of data processing services, with a central focus on the selection of those services that optimize data quality, while upholding privacy and security requirements.~~

In the following of this section, we present our system model (Section 2.1) and our reference scenario (Section 2.2).

2.1 System Model

~~In today's data landscape, the coexistence of data quality and data privacy is critical to support high-value services and pipelines. The increase in data production, collection, and usage has led to a split in scientific research priorities. First, researchers are exploring methods to optimize the usage of valuable data. Here, ensuring data quality is vital, and requires accuracy, reliability, and soundness for analytical purposes. Second, there is a need to prioritize data privacy and security. This involves safeguarding confidential information and complying with strict privacy regulations. These two research directions are happening at the same time, though there are not many solutions that find a good balance between them.~~

~~Our approach seeks to harmonize these objectives by establishing a data governance framework that balances privacy and data quality.~~ Our system model is derived by a generic big-data framework and is composed of the following parties:

Service, a software distributed by a **service provider** that performs a specific task ~~according to access control privileges on data~~;

Pipeline, a sequence of connected services that collect, prepare, process, and analyze data in a structured and automated manner. ~~We distinguish between a **pipeline template** that acts as a skeleton, specifying the structure of the pipeline and the (non-)functional requirements driving service selection and composition, and a **pipeline instance** instantiating the template with services according to the specified requirements~~;

Data Governance Policy, a structured set of privacy guidelines, rules, and procedures regulating data access and protection;

User, executing an analytics pipeline on the data. We assume that the data target of the analytics pipeline are ready for analysis, i.e., they underwent a preparatory phase addressing issues such as missing values, outliers, and formatting discrepancies. This ensures that the data are in an optimal state for subsequent analysis.

We distinguish between a **pipeline template** that acts as a skeleton, specifying the structure of the pipeline, i.e., the chosen sequence of desired services, and both the functional and non-functional requirements driving service selection and composition, and a **pipeline instance** instantiating the template with services according to the specified requirements. The user first selects a pipeline template among a set of functionally-equivalent templates according to its non-functional requirements. It then instantiates the template in a pipeline instance. To this aim, for each component service in the template, it retrieves a set of candidate services that satisfy the functional requirements of the component service. Candidate services are filtered to retrieve a list of compatible services that comply with the policies specified in the template.

The user first selects a pipeline template among a set of functionally-equivalent templates according to its non-functional requirements. It then instantiates the template in a pipeline instance. To this aim, for each component service in the template, it retrieves a set of candidate services that satisfy the functional requirements of the component service. Candidate services are filtered to retrieve a list of compatible services that comply with the policies specified in the template.

Compatible services are ranked based on their ability to retain the maximum amount of information (*data quality* in this paper), while maintaining a minimum level of privacy; the best service is then selected to instantiate the corresponding component service in the template. Upon selecting the most suitable service for each component service in the pipeline template, the pipeline instance is completed and ready for execution. It is important to note that our data governance approach builds on the following assumption: *upholding a larger quantity of data is linked to better data quality.* While this assumption is not true in all settings, it correctly represents many real-world scenarios. We leave a solution that departs from this assumption to our future work.

2.2 Service Pipeline and Reference Scenario

We consider a service-based environment where a service pipeline is designed to analyze data. We define a service pipeline as a graph defined as follows.

Definition 2.1 (Pipeline)**.** A Pipeline is as a direct acyclic graph G(V,E), where V is a set of vertices and E is a set of edges connecting two vertices v_i,$v_k \in V$. The graph has a root $v_r \in V$, a vertex $v_i \in V_S$ for each service s_i, an additional vertex $v_f \subset V$ for each parallel ($\oplus$) structure modeling the contemporary execution (*fork*) of services.

We note that $\{v_r,v_f\} \cup V_S = V$, vertices v_f model branching for parallel structures, and root v_r possibly represents the orchestrator. We also note that, for simplicity but no lack of generality, alternative structures modeling the alternative execution of services are specified as alternative service pipelines, that is, there is no alternative structure in a single service pipeline.

Our reference scenario considers a service pipeline analyzing a dataset of individuals detained in Department of Correction facilities in the state of Connecticut while awaiting trial [?]. In particular, the user, a member of the Connecticut Department of Correction (DOC), seeks to compare admission trends in Connecticut prisons with DOCs in New York and New Hampshire. We assume DOCs to be partners and share data according to their privacy policies. The user's preferences align with a predefined pipeline template that orchestrates the following sequence of operations: (i) *Data fetching*, including the download of the dataset from other states; (ii) *Data preparation*, including data merging, cleaning, and anonymization; (iii) *Data analysis*, including statistical measures like average, median, and clustering-based statistics; (iv) *Data storage*, including the storage of the results; (v) *Data visualization*, including the visualization of the results.

We note that the template requires the execution of the entire service within the Connecticut Department of Correction. If the data needs to be transmitted beyond the boundaries of Connecticut, data protection measures must be implemented. A visual representation of the flow is presented in Figure 1. Table 1 presents a sample of the adopted dataset.[1] Each row represents an inmate; each column includes the following attributes: date of download, a unique identifier, last entry date, race, gender, age of the individual, the bound value, offense, entry facility, and detainer. To serve the objectives of our study, we have extended this dataset by introducing randomly generated first and last names.

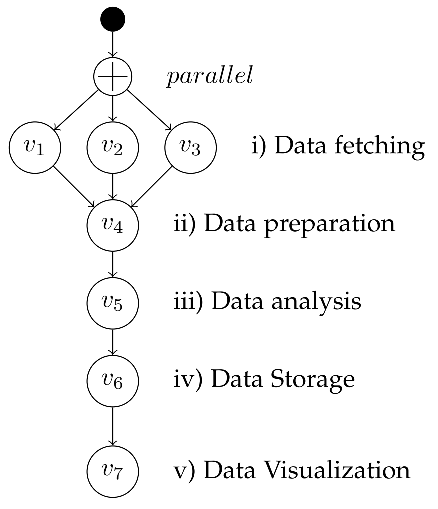


Fig. 1: Reference Scenario

3 Pipeline Template

Our approach integrates data protection and data management into the service pipeline using annotations. To this aim, we extend the service pipeline in Definition 2.1 with: *i)* data protection annotations expressing transformations on data to enforce data protection requirements, *ii)* functional annotations expressing data manipulations carried out during services execution. These annotations permit to implement an advanced data lineage, tracking the entire data lifecycle by monitoring changes arising from functional service execution and data protection requirements.

1. https://data.ct.gov/Public-Safety/Accused-Pre-Trial-Inmates-in-Correctional-Faciliti/b674-jy6w

In the following, we first introduce the annotated service pipeline, called pipeline template (Section 3.1). We then present functional annotations (Section 3.3) and data protection annotations (Section 3.2). We finally provide an example of a pipeline template (Section **??**).

3.1 Pipeline Template Definition

Given the service pipeline in Definition 2.1, we use annotations to express data protection requirements to be enforced on data and functional requirements on services to be integrated in the pipeline. Each service vertex in the service pipeline is labeled with two mapping functions forming a pipeline template: *i*) a labeling function $\lambda : V_S \rightarrow P$ that associates a set of data protection requirements, in the form of policies $p \in P$, with each vertex $v_i \in V_S$; *ii*) a labeling function $\gamma : V_S \rightarrow F$ that associates a functional service description $F_i \in F$ with each vertex $v_i \in V_S$.

The template is formally defined as follows.

Definition 3.1 (Pipeline Template)**.** Given a service pipeline G(V,E), a pipeline template $G^{\lambda,\gamma}$(V,E,λ,γ) is a direct acyclic graph with two labeling functions:

i λ that assigns a label $\lambda(v_i)$, corresponding to a set P_i of policies p_j to be satisfied by service s_i represented by v_i, for each vertex $v_i \in V_S$;

ii γ that assigns a label $\gamma(v_i)$, corresponding to the functional description F_i of service s_i represented by v_i, for each vertex $v_i \in V_S$.

We note that, at this stage, the template is not yet linked to any service. We also note that policies $p_j \in P_i$ annotated with $\lambda(v_i)$ are ORed, meaning that the access decision is positive if at least one policy p_j is evaluated to *true*. An example of pipeline template is depicted in Fig. 2

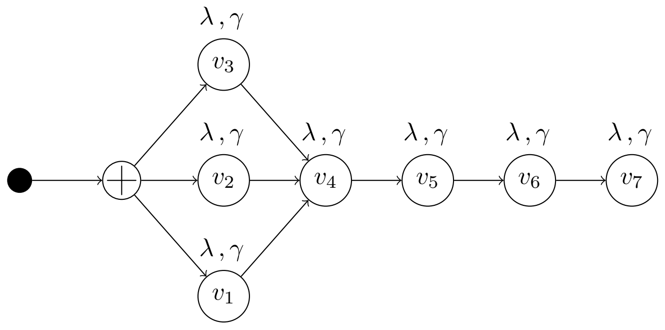


Fig. 2: Pipeline Template

3.2 Data Protection Annotation

Data Protection Annotation λ expresses data protection requirements in the form of access control policies. We consider an attribute-based access control model that offers flexible fine-grained authorization and adapts its standard key components to address the unique characteristics of a big data environment. Access requirements are expressed in the form of policy conditions that are defined as follows.

Definition 3.2 (Policy Condition)**.** A *Policy Condition pc* is a Boolean expression of the form (*attr_name* op *attr_value*),

TABLE 1: Dataset sample

DOWNLOAD DATE	ID	FNAME	LNAME	LAD	RACE	GENDER	AGE	BOND	OFFENSE	...
05/15/2020	ZZHCZBZZ	ROBERT	PIERCE	08/16/2018	BLACK	M	27	150000	CRIMINAL POSS ...	...
05/15/2020	ZZHZZRLR	KYLE	LESTER	03/28/2019	HISPANIC	M	41	30100	VIOLATION OF P...	...
05/15/2020	ZZSRJBEE	JASON	HAMMOND	04/03/2020	HISPANIC	M	21	150000	CRIMINAL ATTEM...	...
05/15/2020	ZZHBJLRZ	ERIC	TOWNSEND	01/15/2020	WHITE	M	36	50500	CRIM VIOL OF P...	...
05/15/2020	ZZSRRCHH	MICHAEL	WHITE	12/26/2018	HISPANIC	M	29	100000	CRIMINAL ATTEM...	...
05/15/2020	ZZEJCZWW	JOHN	HARPER	01/03/2020	WHITE	M	54	100000	CRIM VIOL OF P...	...
05/15/2020	ZZHJBJBR	KENNETH	JUAREZ	03/19/2020	HISPANIC	M	35	100000	CRIM VIOL ST C...	...
05/15/2020	ZZESESZW	MICHAEL	SANTOS	12/03/2018	WHITE	M	55	50000	ASSAULT 2ND, V...	...
05/15/2020	ZZRCSHCZ	CHRISTOPHER	JONES	05/13/2020	BLACK	M	43	10000	INTERFERING WIT...	...

with op$\in\{<,>,=,\neq,\leq,\geq\}$, *attr_name* an attribute label, and *attr_value* the corresponding attribute value.

An access control policy then specifies who (*subject*) can access what (*object*) with action (*action*), in a specific context (*environment*) and under specific obligations (*data transformation*), as formally defined below.

Definition 3.3 (Policy)**.** A *policy p*$\in$*P* is 5-uple <*subj*, *obj*, *act*, *env*, T^P>, where:

Subject *subj* defines a service s_i issuing an access request to perform an action on an object. It is of the form <*id*, $\{pc_i\}$>, where *id* defines a class of services (e.g., classifier), and $\{pc_i\}$ is a set of *Policy Conditions* on the subject, as defined in Definition 3.2. For instance, <*service*,{(classifier = "SVM")}> refers to a service providing a SVM classifier. We note that *subj* can also specify conditions on the service owner, such as, <*service*,{(owner_location = "EU")}> and on the service user, such as, <*service*,{(service_user_role = "DOC Director")}>.

Object *obj* defines any data whose access is governed by the policy. It is of the form <*type*, $\{pc_i\}$>, where: *type* defines the type of object, such as a file (e.g., a video, text file, image, etc.), a SQL or noSQL database, a table, a column, a row, or a cell of a table, and $\{pc_i\}$ is a set of *Policy Conditions* defined on the object's attributes. For instance, <*dataset*,{(region = CT)}> refers to a dataset whose region is Connecticut.

Action *act* defines any operations that can be performed within a big data environment, from traditional atomic operations on databases (e.g., CRUD operations varying depending on the data model) to coarser operations, such as an Apache Spark Direct Acyclic Graph (DAG), Hadoop MapReduce, an analytics function call, or an analytics pipeline.

Environment *env* defines a set of conditions on contextual attributes, such as time of the day, location, IP address, risk level, weather condition, holiday/workday, emergency. It is a set $\{pc_i\}$ of *Policy Conditions* as defined in Definition 3.2. For instance, <*env*,{(time = "night")}> refers to a policy that is applicable only at night.

Data Transformation T^P defines a set of security and privacy-aware transformations on *obj*, which must be enforced before any access to data. Transformations focus on data protection, as well as compliance to regulations and standards, in addition to simple format conversions. For instance, let us define three transformations that can be applied to the Table 1: i) *level0* (t_0^p): no anonymization is carried out; ii) *level1* (t_1^p): The data has been partially anonymized with only the first name and last name being anonymized; iii) *level2* (t_2^p): The data has been fully anonymized with the first name, last name, identifier, and age being anonymized.

Access control policies $p_j\in P_i$ annotating vertex v_i in a pipeline template $G^{\lambda,\gamma}$ are used to filter out those candidate services $s\in S^c$ that do not match data protection requirements. Specifically, each policy $p_j\in P_i$ is evaluated to verify whether a candidate service $s\in S^c$ for vertex v_i is compatible with data protection requirements in P_i ($\lambda(v_i)$). Policy evaluation matches the profile *prf* of candidate service $s\in S^c$ with the policy conditions in each $p_j\in P_i$. If the credentials and declarations, defined as a set of attributes in the form (*name*, *value*), in the candidate service profile fails to meet the policy conditions, meaning that no policies p_j are evaluated to *true*, the service is discarded; otherwise it is added to the set S' of compatible service, which is used in Section 4 to generate the pipeline instance G'. No policy enforcement is done at this stage.

3.3 Functional Annotations

A proper data management approach must track functional data manipulations across the entire pipeline execution, defining the functional requirements of each service operating on data. To this aim, each vertex $v_i\in V_S$ is annotated with a label $\gamma(v_i)$, corresponding to the functional description F_i of the service s_i represented by v_i. F_i describes the functional requirements on the corresponding service s_i, such as API, inputs, expected outputs. It also specifies a set T^F of data transformation functions t_i^f, possibly triggered during execution of the connected service s_i.

Each $t_i^f\in T^F$ can be of different types as follows: *i*) an empty function t_ϵ^f that applies no transformation or processing on the data; *ii*) an additive function t_a^f that expands the amount of data received, for example, by integrating data from other sources; *iii*) a transformation function t_t^f that transforms some records in the dataset without altering the domain; *iv*) a transformation function t_d^f (out of the scope of this work) that changes the domain of the data by applying, for instance, PCA or K-means.

For simplicity but with no loss of generality, we assume that all candidate services meet functional annotation F and that T^F=t^f. As a consequence, all candidate services apply the same transformation to data during execution.

3.4 Example

Let us consider our reference scenario in Section 2.1. Fig. 2 presents an example of pipeline template consisting of five stages, each one annotated with a policy in Table 2.

The first stage consists of three parallel vertices v_1, v_2, v_3 for data collection. Data protection annotations $\lambda(v_1)$, $\lambda(v_2)$, $\lambda(v_3)$ refer to policy p_0 with an empty transformation t_0^p. Functional requirement F_1, F_2, F_3 prescribes a URI as input and the corresponding dataset as output.

The second stage consists of vertex v_4, merging the three datasets obtained stage 1. Data protection annotation $\lambda(v_4)$ refers to policies p_1 and p_2, which apply different data transformations depending on the relation between the dataset and service owners. If the service owner is also the dataset owner ($\langle service\_owner = dataset\_owner\rangle$), the dataset is not anonymized (t_0^p). We note that if the service owner has no partner relationship with the dataset owner, no policies apply. If the service owner is a partner of the dataset owner ($\langle service\_owner = partner(dataset\_owner)\rangle$), the dataset is anonymized at level l_1 (t_1^p). Functional requirement F_4 prescribes n datasets as input and the merged dataset as output.

The third stage consists of vertex v_5 for data analysis. Data protection annotation $\lambda(v_5)$ refers to policies p_1 and p_2, as for stage 2. Functional requirement F_5 prescribes a dataset as input and the results of the data analysis as output. Data protection annotation $\lambda(v_5)$ refers to policy p_4 with data transformation t_2^p, that is, anonymization level l_2 to prevent personal identifiers from entering into the machine learning algorithm/model. Functional requirement F_6 prescribes a dataset as input, and the trained model and a set of inferences as output.

The fourth stage consists of vertex v_6, managing data storage. Data protection annotation $\lambda(v_6)$ refers to policies p_5 and p_6, which apply different data transformations depending on the relation between the dataset and service region. If the service region is the dataset origin ($\langle service\_region = dataset\_origin"\rangle$) ($p_5$), the dataset is anonymized at level l_1 (t_1^p). If the service region is in a partner region ($\langle service, region = NY, NH"\rangle$) ($p_6$), the dataset is anonymized at level l_2 (t_2^p). Functional requirement F_7 prescribes a dataset as input and the URI of the stored data as output.

The sixth and last stage consists of vertex v_7, responsible for data visualization. Data protection annotation $\lambda(v_7)$ refers to policies p_7 and p_8, which anonymize data according to the environment where the service is executed. A *risky* environment is defined as a region outside the owner or partner facility. If the environment is risky (p_7), the data are anonymized at level r_0 (t_3^p). If the environment is not risky (p_8), the data are anonymized at level r_1 (t_4^p). Functional requirement F_8 prescribes a dataset as input and data visualization interface (possibly in the form of JSON file) as output.

4 PIPELINE INSTANCE

A Pipeline Instance $G'(V', E, \lambda)$ instantiates a Pipeline Template $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$ by composing services in the instance according to data protection and functional annotations in the template. It is formally defined as follows.

Definition 4.1 (Pipeline Instance)**.** Let $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$ be a pipeline template, a pipeline Instance $G'(V', E, \lambda)$ is an isomorphic directed acyclic graph where: *i)* $v'_r{=}v_r$; *ii)* for each vertex $v \in V_{\otimes} \cup V_{\oplus}$, there exists a corresponding vertex $v' \in V'_{\otimes} \cup V'_{\oplus}$; *iii)* for each $v_i{\in}V_S$ annotated with policy P_i, there exists a corresponding vertex $v'_i{\in}V'_S$ instantiated with a service s'_i, such that:

1) s'_i satisfies data protection annotation $\lambda(v_i)$ in $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$;
2) s'_i satisfies functional annotation $\gamma(v_i)$ in $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$.

Condition 1 requires that each selected service s'_i satisfies the policy requirements P_i of the corresponding vertex v_i in the Pipeline Template, whereas Condition 2 is needed to preserve the process functionality, as it simply states that each service s'_i must satisfy the functional requirements F_i of the corresponding vertex v_i in the Pipeline Template.

We then define a *pipeline instantiation* function that takes as input a Pipeline Template $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$ and a set S^c of candidate services, with a specific set of services S^c_i for each vertex $v_i{\in}V_S$, and returns as output a Pipeline Instance $G'(V', E, \lambda)$. Recall from Section 3.3 that all candidate services meet the functional annotation in the template, meaning that Condition 2 in Definition 4.1 is satisfied for all candidate services.

The Pipeline Instance is generated by traversing the Pipeline Template with a breadth-first search algorithm, starting from the root vertex v_r. Then, for each vertex $v \in V_{\oplus} \bigcup V_{\otimes}$ in the pipeline template, the corresponding vertex $v' \in V'_{\oplus} \bigcup V'_{\otimes}$ is generated. Finally, for each vertex $v_i{\in}V_S$, a two-step approach is applied as follows.

1) *Filtering Algorithm* – The filtering algorithm checks if the profile prf_j of each candidate service $s_j{\in}S^c_i$ satisfies the policies $p_k{\in}P_i$ corresponding to $\lambda(v_i)$. If prf_j satisfies at least one policy, service s_j is compatible, otherwise it is discarded. The filtering algorithm finally returns a subset $S'_i{\subseteq}S^c_i$ of compatible services for each vertex $v_i{\in}V_S$.
2) *Selection Algorithm* – The selection algorithm selects one service s'_i for each set S'_i of compatible services and instantiates the corresponding vertex $v'_i{\in}V'$ with it. There are many ways of choosing s'_i, we present our approach based on the minimization of quality loss in Section **??**.

When all vertices $v_i{\in}V$ have been visited, the Pipeline Instance G′ is finalized, with a service instance s'_i for each $v'_i{\in}V'$. Vertex v'_i is still annotated with policies $p_k{\in}P_i$ according to λ, because policies in P_i are evaluated and enforced only when the pipeline instance is triggered, before any service is executed. In case policy evaluation returns *true*, data transformation $T^P{\in}P_i$ is applied, otherwise a default transformation that removes all data is applied.

4.1 Example

TBD (mettere un esempio in cui non sia stato scelto l'ottimo che invece cercheremo e metteremo come esempio nella prossima sezione)

Example 4.1 (Pipeline Instance). Let us consider a subset $\{v_5,\ v_6,\ v_7\}$ of the pipeline template $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$ in Section 4.1.

TABLE 2: Anonymization policies (a) and data transformations (b)

Vertex	Policy	$\langle$subject, $object, action, environment, transformation\rangle$
v_1,v_2,v_3	p_0	$\langle$ANY, $dataset, READ, ANY, t^p_0\rangle$
v_4,v_5	p_1	$\langle\langle service\_owner = dataset\_owner\rangle, dataset, READ, ANY, t^p_0\rangle$
v_4,v_5	p_2	$\langle\langle service\_owner = partner(dataset\_owner)\rangle, dataset, READ, ANY, t^p_1\rangle$
v_6	p_5	$\langle\langle service\_region = dataset\_origin\rangle, dataset, WRITE, ANY, t^p_0\rangle$
v_6	p_6	$\langle\langle service\_region = “\{NY, NH\}”\rangle, dataset, WRITE, ANY, t^p_1\rangle$
v_7	p_7	$\langle$ANY, $dataset, READ$,environment = risky, $t^p_3\rangle$
v_7	p_8	$\langle$ANY, $dataset, READ$,environment = not_risky, $t^p_4\rangle$

(a)

t^p_i	Level	Data Transformation
t^p_0	l_0	$anon(\varnothing)$
t^p_1	l_1	$anon(fname, lname)$
t^p_2	l_2	$anon(fname, lname, id, age)$
t^p_3	r_0	$aggregation(cluster = \infty)$
t^p_4	r_1	$aggregation(cluster = 10)$

(b)

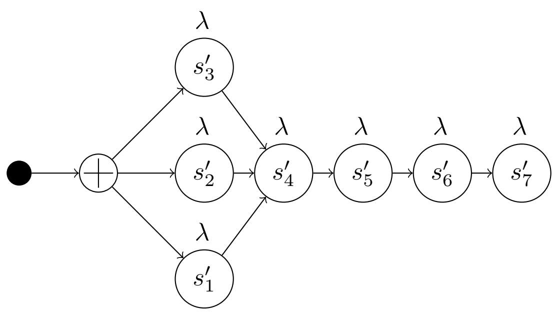


Fig. 3: Service composition instance

Each vertex is associated with three candidate services, each having a profile. The filtering algorithm matches each candidate service's profile with the policies annotating the corresponding vertex. It returns the set of services whose profile matches a policy. *i*) For v_6, the filtering algorithm produces the set $S'_1 = \{s_{61}, s_{62}\}$; assuming that the dataset owner is “CT”, the service profile of $s61$ matches p_1 and the one of $s62$ matches p_2. For $s63$, there is no policy match and, thus, it is discarded. *ii*) For v_7, the filtering algorithm returns the set $S'_2 = \{s_{72}, s_{73}\}$; assuming that the dataset region is “CT”, the service profile of $s72$ matches p_5 and the one of $s73$ matches p_6. For $s71$, there is no policy match and, thus, it is discarded. *iii*) For v_8, the filtering algorithm returns the set $S'_3 = \{s_{81}, s_{82}, s_{83}\}$. Since policy p_7 matches with any subjects, the filtering algorithm does not discard any service.

For each vertex, we could select the initial matching servic from each set S'_* and incorporate it into the instance. For instance, for v_6, we select $s61$; for v_7, $s72$ is chosen, and for v_8, $s81$ is the preferred option. The instance thus formulated is depicted in Table 3. It is imperative to acknowledge that this instance is valid it satisfies all the policies in the pipeline template. However, it does not represent the optimal instance achievable. o determine the optimal instance, it is essential to evaluate services based on specific quality metrics that reflect their impact on data quality. In the next sections, we will introduce the metrics that we use to evaluate the quality of services and the results of the experiments conducted to evaluate the performance of our approach.

5 Maximizing the Pipeline Instance Quality

Our goal is to generate a pipeline instance with maximum quality, which addresses data protection requirements with the minimum amount of information loss across the pipeline execution. To this aim, we first discuss the role of some metrics (Section 5.1) to specify and measure data quality, and describe the ones used in the paper. Then, we prove that the problem of generating a pipeline instance with maximum quality is NP-hard (Section 5.2). Finally, we present a parametric heuristic (Section 5.3) tailored to address the computational complexity associated with enumerating all possible combinations within a given set. The primary aim of the heuristic is to approximate the optimal path for service interactions and transformations, particularly within the landscape of more complex pipelines composed of numerous vertexes and candidate services. Our focus extends beyond identifying optimal combinations, encompassing an understanding of the quality changes introduced during the transformation processes.

5.1 Quality Metrics

Ensuring data quality is mandatory to implement data pipelines that provide accurate results and decision-making along the whole pipeline execution. To this aim, quality metrics evaluate the quality loss introduced at each step of the data pipeline, and can be classified as *quantitative* or *qualitative* [?]. Quantitative metrics monitor the amount of data lost during data transformations as the quality difference between datasets X and Y. Qualitative metrics evaluate changes in the properties of datasets X and Y. For instance, qualitative metrics can measure the changes in the statistical distribution of the two datasets.

In this paper, we provide two metrics, one quantitative and one qualitative, that compare the input dataset X and dataset Y generated by enforcing data protection requirements (i.e., our policy-driven transformation in Section [?]) on X at each step of the data pipeline.

5.1.1 Jaccard coefficient

The Jaccard coefficient is a quantitative metric that can be used to measure the difference between the elements in two datasets. It is defined as:

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

where X and Y are two datasets of the same size.

The Jaccard coefficient is computed by dividing the cardinality of the intersection of two sets by the cardinality of their union. It ranges from 0 to 1, where 0 indicates no similarity and 1 indicates complete similarity between the datasets.

TABLE 3: Instance example

Vertex→Policy	Candidate	Profile	Filtering	Instance	Candidate	Ranking
	$s41$	service_owner = ”CT”	✓	✓	$s41$	1
$v_4 \rightarrow p_1,p_2$	$s42$	service_owner = ”NY”	✓	✗	$s42$	2
	$s43$	service_owner = ”CA”	✗	✗	$s43$	–
	$s51$	service_region = ”CA”	✗	✗	$s51$	–
$v_5 \rightarrow p_5,p_6$	$s52$	service_region = ”CT”	✓	✓	$s52$	1
	$s53$	service_region = ”NY”	✓	✗	$s53$	2
	$s61$	visualization_location = ”CT_FACILITY”	✓	✓	$s61$	1
$v_6 \rightarrow p_7,p_8$	$s62$	visualization_location = ”CLOUD”	✓	✗	$s62$	2

The Jaccard coefficient has several advantages. Unlike other similarity measures, such as Euclidean distance, it is not affected by the magnitude of the values in the dataset. It is suitable for datasets with categorical variables or nominal data, where the values do not have a meaningful numerical interpretation.

5.1.2 Jensen-Shannon Divergence

The Jensen-Shannon divergence (JSD) is a quantitative metric that can be used to measure the dissimilarity between the probability distributions of two datasets. It is a symmetrized version of the KL divergence [**?**] .

The JSD between X and Y is defined as:

$$JSD(X,Y) = \frac{1}{2}\left(KL(X||M) + KL(Y||M)\right)$$

where X and Y are two datasets of the same size, and M=0.5*(X+Y) is the average distribution.

JSD incorporates both the KL divergence from X to M and from Y to M. It provides a balanced measure of dissimilarity that is symmetric and accounts for the contribution from both datasets. JSD can compare the dissimilarity of the two datasets, providing a symmetric and normalized measure that considers the overall data distribution. It provides a more comprehensive understanding of the dissimilarity between X and Y, taking into account the characteristics of both datasets.

We note that our metrics can be applied either to the entire dataset or to specific features only. The features can be assigned with equal or varying importance, providing a weighted version of the metrics, thus enabling the prioritization of important features that might be possibly lost during the policy-driven transformation in Section [**?**]. A complete taxonomy of possible metrics is however outside the scope of this paper and will be the target of our future work.

5.2 NP-Hardness of the Max Quality Pipeline Instantiation Process

se lo definiamo in maniera formale come il problema di trovare un'istanza valida in accordo alla definizione di istanza tale che non ne esiste una con un loss piu' piccolo?

Definition 5.1 (Max Quality Pipeline Instantiation Process)**.** Given $dtloss_i$ the value of the quality metric computed after applying the transformation of the policy matching the service selected to instantiate vertex $v_i \in V_S$, the Max quality Pipeline Instantiation Process is the case in which the *pipeline instantiation* function returns a Pipeline Instance where the $dtloss_i$ sum is maximized.

The Max Quality Pipeline Instantiation Process is a combinatorial selection problem and is NP-hard, as stated by Theorem 5.1. However, while the overall problem is NP-hard, there is a component of the problem that is solvable in polynomial time: matching the profile of each service with the node policy. This can be done by iterating over each node and each service, checking if the service matches the node's policy. This process would take $O(|N|*|S|)$ time. This is polynomial time complexity.

Theorem 5.1. The Max Quality Pipeline Instantiation Process is NP-Hard.

Proof: The proof is a reduction from the multiple-choice knapsack problem (MCKP), a classified NP-hard combinatorial optimization problem, which is a generalization of the simple knapsack problem (KP) []. In the MCKP problem, there are t mutually disjoint classes $N_1, N_2, \ldots, N_t$ of items to pack in some knapsack of capacity C, class N_i having size n_i. Each item $j \in N_i$ has a profit p_{ij} and a weight w_{ij}; the problem is to choose one item from each class such that the profit sum is maximized without having the weight sum to exceed C.

The MCKP can be reduced to the Max quality Pipeline Instantiation Process in polynomial time, with $N_1, N_2, \ldots, N_t$ corresponding to $S^c_1, S^c_1, \ldots, S^c_u$, $t{=}u$ and n_i the size of S^c_i. The profit p_{ij} of item $j \in N_i$ corresponds to $dtloss_{ij}$ computed for each candidate service $s_j \in S^c_i$, while w_{ij} is uniformly 1 (thus, C is always equal to the cardinality of V_C).

Since the reduction can be done in polynomial time, our problem is also NP-hard. (non è sufficiente, bisogna provare che la soluzione di uno e' anche soluzione dell'altro)

Example 5.1 (Max-Quality Pipeline Instance)**.** Let us consider a subset $\{v_5,\ v_6,\ v_7\}$ of the pipeline template $G^{\lambda,\gamma}(V,E,\lambda,\gamma)$ in Section 4.1. Each vertex is associated with three candidate services, each having a profile. The filtering algorithm matches each candidate service's profile with the policies annotating the corresponding vertex. It returns the set of services whose profile matches a policy.

The comparison algorithm is then applied to the set of services S'_* and it returns a ranking of the services. The ranking is based on the amount of data that is anonymized by the service. The ranking is listed in Table 3 and it is based on the transformation function of the policies, assuming that a more restrictive transformation function anonymizes more data affecting negatively the position in the ranking. For example, $s11$ is ranked first because it anonymizes less data than $s12$ and $s13$. The ranking of $s22$ and $s23$ is based on the same logic. Finally, the ranking of $s31$, $s32$ is influenced by the environment state at the time of the ranking. For example, if the environment in which the visualization is performed is a CT facility, then $s31$ is ranked first and $s32$ second; thus because the facility is considered a less risky environment than the cloud.

5.3 Heuristic

HO RIVISTO IL PARAGRAFO VELOCEMENTE GIUSTO PER DARE UN'INDICAZIONE. DOBBIAMO USARE LA FORMALIZZAZIONE E MAGARI FORMALIZZARE ANCHE LO PSEUDOCODICE. We design and implement a heuristic algorithm for computing the pipeline instance maximizing data quality. Our heuristic is built on a *sliding window* and aims to minimize information loss according to quality metrics. At each step, a set of vertexes in the pipeline template $G^{\lambda,\gamma}(V,E,\lambda,\gamma)$ is selected according to a specific window w=[i,j], where i and j are the starting and ending depth of window w. Service filtering and selection in Section 4 are then executed to minimize information loss in window w. The heuristic returns as output the list of services instantiating vertexes at depth i. A new window w=[i+1,j+1] is considered until j+1 is equal to the max depth of $G^{\lambda,\gamma}(V,E,\lambda,\gamma)$, that is the window reaches the end of the template. This strategy ensures that only services with low information loss are selected at each step, minimizing the overall information loss. Pseudo-code for the sliding window algorithm is presented in Algorithm 1.

```
1   selectedServices = empty
2   for i from 0 to length(serviceCombinations):
3       minMetricCombination = None
4       minMetric = +∞
5      M = JSD or J //JSD or Jaccard coefficient
6
7       for j from i to i + windowSize:
8           totalMetric = 0
9           for service in serviceCombinations[j]:
10              totalMetric += M(service)
11              currentMetric = totalMetric / length(
      serviceCombinations[j])
12              if currentMetric < minMetric:
13                  minMetric = currentMetric
14                  minMetricCombination =
      serviceCombinations[j]
15                  firstService = serviceCombinations
      [j][0]
16      add firstService to instance
17  return instance
18
19
```

Listing 1: Sliding Window Heuristic with Selection of First Service from Optimal Combination

The pseudocode implemets function *SlidingWindowHeuristic*, which takes a sequence of vertexes and a window size as input and returns a set of selected vertexes as output. The function starts by initializing an empty set of selected vertexes (line 3). Then, for each node in the sequence (lines 4–12), the algorithm iterates over the vertexes in the window (lines 7–11) and selects the node with the lowest metric value (lines 9-11). The selected node is then added to the set of selected vertexes (line 12). Finally, the set of selected vertexes is returned as output (line 13).

We note that a window of size 1 corresponds to the *greedy* approach, while a window of size N, where N represents the total number of vertexes, corresponds to the *exhaustive* method.

The utilization of heuristic in service selection can be enhanced through the incorporation of techniques derived from other algorithms, such as Ant Colony Optimization or Tabu Search. By integrating these approaches, it becomes feasible to achieve a more effective and efficient selection of services, with a specific focus on eliminating paths that have previously been deemed unfavorable.

6 EXPERIMENTS

We experimentally evaluated the performance and quality of our methodology, and corresponding heuristic implementation in Section 5.3, and compare them against the exhaustive approach in Section **??**. In the following, Section 6.1 presents the simulator and testing infrastructure adopted in our experiments, as well as the complete experimental settings; Section 6.2 analyses the performance of our solution in terms of execution time; Section 6.3 presents the quality of our heuristic algorithm in terms of the metrics in Section 5.1.

6.1 Testing Infrastructure and Experimental Settings

Our testing infrastructure is a Swift-based simulator of a service-based ecosystem, including service execution, comparison, and composition. The simulator first defines the pipeline template as a sequence of vertexes in the range $3-7$. We recall that alternative vertexes are modeled in different pipeline templates, while parallel vertexes only add a fixed execution time that is negligible and do not affect the quality of our approach. Each vertex is associated with a (set of) policy with transformations varying in three classes:

roman* *Confident*: Adjusts data removal to a percentage within $[0.8,1]$. roman* *Diffident*: Sets data removal percentage to $[0.33,0.5]$. roman* *Average*: Modifies data removal percentage within $[0.33,1]$. set of functionally-equivalent candidate services is randomly generated.

Upon setting the sliding window size, the simulator selects a subset of vertexes along with their corresponding candidate services. It then generates all possible service combinations for the chosen vertexes. For each combination, the simulator calculates a metric, selecting the first service from the optimal combination before shifting the sliding window. When the end of the vertex list is reached, or when the window size equals the vertex count, the simulator computes the optimal service combination for the remaining vertexes.

An hash function is to simulate the natural interdependence between services. This is particularly important when the removal of data by one service may impact another. By assigning weights to the services using this function, the system aims to reflect the interconnected dynamics among the services.

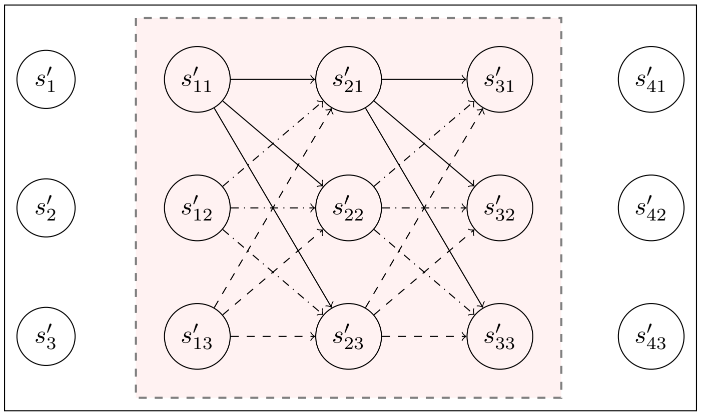


Fig. 4: Execution example

The simulator is used to assess the performance and quality of our sliding window heuristic in Section 5 for the generation of the best pipeline instance (Section 4). Our experiments have been run on a workstation equipped with a 2.40GHz i5-8279U CPU with 16GB RAM and a 512GB SSD. Each experiment was repeated ten times and the results averaged to improve the reliability of the data.

6.2 Perfomance

We first calculated the execution time required by our exhaustive solution. We incrementally varied the number of vertexes and the number of services per vertex. The results of these evaluations are presented in Fig. 8. As anticipated, the trend in execution times is exponential. Fig. 8 displays the execution time plots, clearly showing that as the number of vertexes increases, the execution time grows exponentially. Execution times for up to 5 vertexes and 6 services were computed directly, while the remaining data points were obtained through interpolation. Subsequently, the logical extension of this empirical inquiry involves evaluating the execution time efficiency attributable to the implementation of the sliding window heuristic.

We then evaluated our heuristics to quantify the execution time reduction achieved through the application of heuristics. In this context, the number of vertexes and services per vertex was incrementally increased, with the addition of a sliding window whose size was progressively enlarged in each experiment. The outcomes are depicted in **??**, and as expected, we observed a marked reduction in execution times with the implementation of the sliding window heuristic. This empirical evidence highlights the heuristic's ability to reduce computational demands, an aspect that becomes increasingly pivotal as the problem's complexity grows. The use of a logarithmic scale to illustrate the results linearizes the exponential growth associated with the exhaustive method, offering a clear visual confirmation of the heuristic's efficiency in decreasing computational time.

6.3 Quality

We finally evaluated the quality of our heuristic comparing, where possible, its results with the optimal solution retrieved by executing the exhaustive approach. The latter executes with window size equals to the number of vertexes in the pipeline template, and provides the best, among all possible, solutions.

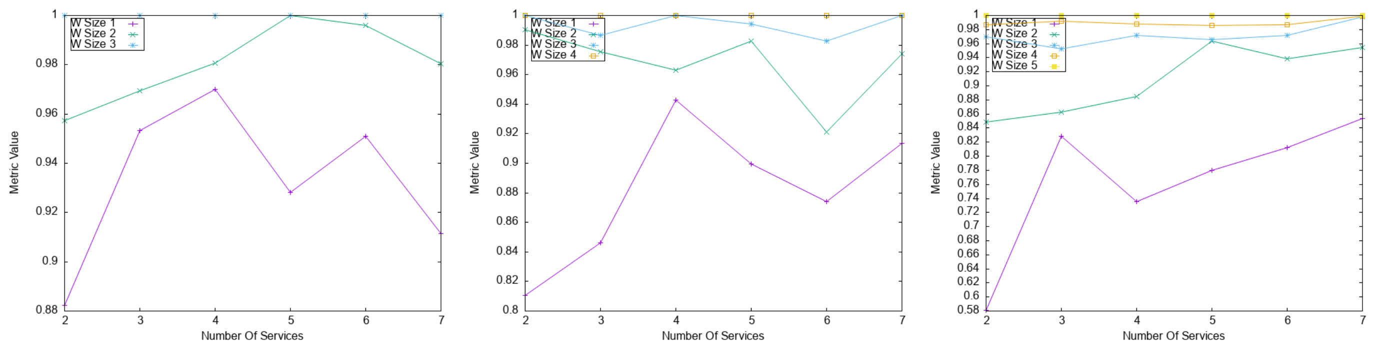
We run our experiments in the three settings in Section Section 6.1, namely, confident, diffident, average, and varied: *i)* the number of vertexes in the pipeline template in [3,6], *ii)* the window size in [1,$|\max_v|$], where $\max_v$ is the number of vertexes in the pipeline template, and *iii)* the number of candidate services for each vertex in the pipeline template in [2, 6].

?? and Figs. 5 and 6 presents our results using metric Jensen-Shannon Divergence. When a diffident setting is used, **??**, the quality range from 0.7 to 0.9, with the highest quality retrieved for the pipeline template with 3 vertices and the lowest with 7 vertices. In particular, the quality ranges from 0.88 (greedy approach) to 0.93 (exhaustive approach) for a 3-vertex pipeline with a loss of 5,38% in the worst case, from 0.84 to 0.92 for a 4-vertex pipeline with a loss of 8,7%, from 0.84 to 0.89 for a 5-vertex pipeline with a loss of 5,61%, from 0.8 to 0.89 for a 6-vertex pipeline with a loss of 10,11%, and from 0.72 to 0.88 for a 7-vertex pipeline with a loss of 18,18%. We note that the benefit of an increasing window size can be appreciated with lower numbers, reaching a sort of saturation around the average length (e.g., window of length 4 with a 7-vertex pipeline) where the quality with different length almost overlaps. The only exception is for 6-vertex pipeline where the overapping starts with window size 2. However, this might be due to the specific setting and therefore does not generalize. We also observe that, as the window size increase, the quality increase as well. This suggests that the heuristic performs better when it has a broader perspective of the data it is governing.

QUESTO E' PIU' DA CONCLUSIONE FINALE. Finally, the data suggest that while larger window sizes generally lead to better performance, there might exist a point where the balance between window size and performance is optimized. Beyond this point, the incremental gains in metric values may not justify the additional computational resources or the complexity introduced by larger windows.

7 Related Work

8 Conclusions


(a) 3 vertices (b) 4 vertices (c) 5 vertices

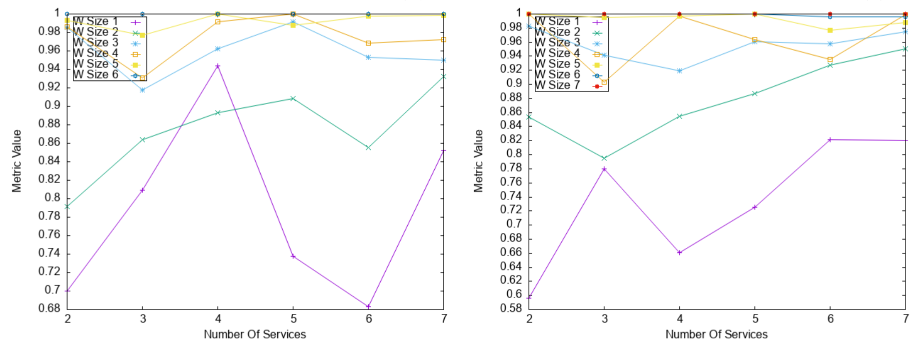
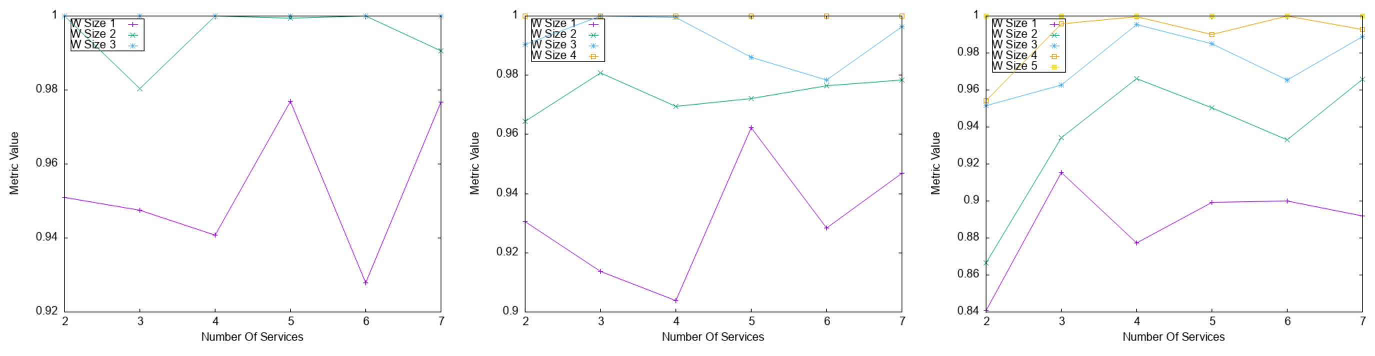

(d) 6 vertices (e) 6 vertices

Fig. 5: Quality evaluation with *Confident* profile.


(a) 3 vertices (b) 4 vertices (c) 5 vertices

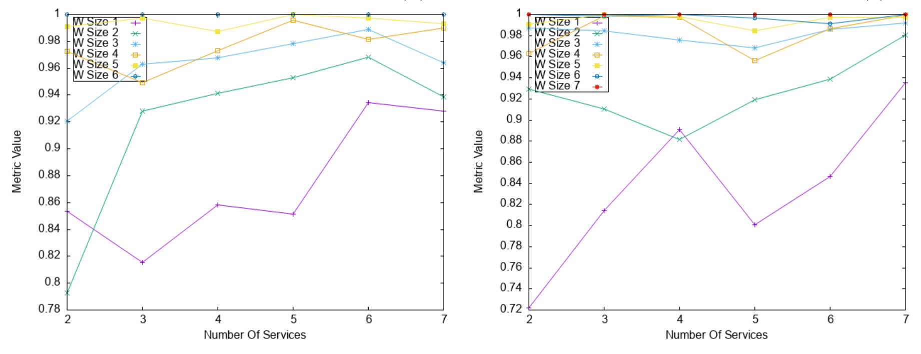
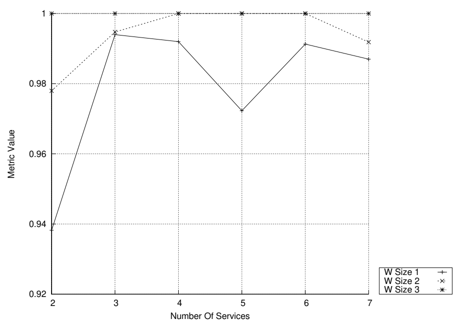
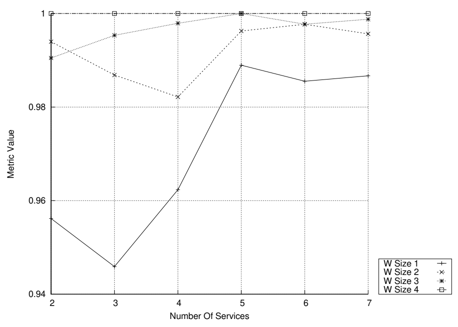

(d) 6 vertices (e) 6 vertices

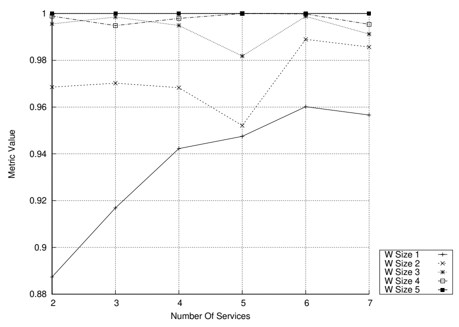
Fig. 6: Quality evaluation with *Confident* profile.



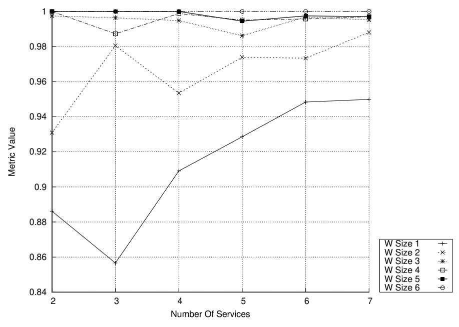
(a) 3 vertices



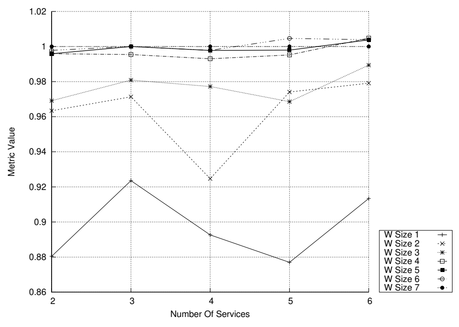
(b) 4 vertices



(c) 5 vertices



(d) 6 vertices



(e) 6 vertices

Fig. 7: Quality evaluation with *Confident* profile.

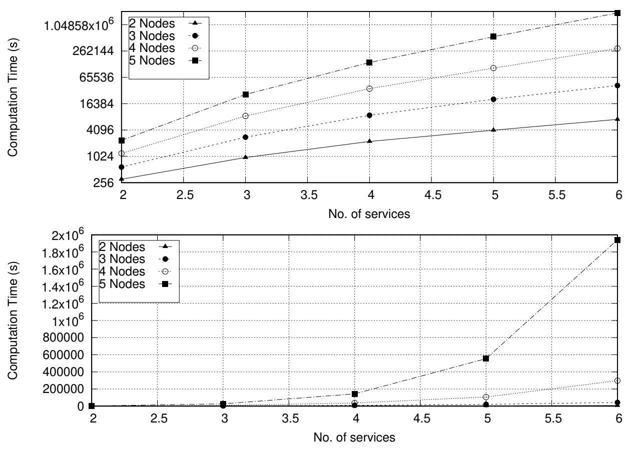


Fig. 8: Exhaustive execution time evaluation. The x-axis represents the number of services, while the y-axis represents the execution time in seconds. The execution time is expressed both in linear and logarithmic scales.

REFERENCES