Design of Cloud Data Platforms

Alex Baiardi[1], Matteo Francia[1,*], Enrico Gallinucci[1], Matteo Golfarelli[1] and Manuele Pasini[1]

Abstract

Data platforms are state-of-the-art solutions to implement data-driven applications and analytics, since they facilitate the ingestion, storage, management, and exploitation of big data. Data platforms are built on top of complex ecosystems of services answering different data needs and requirements; such ecosystems are offered by different providers (e.g., Amazon AWS and Microsoft Azure). However, when it comes to engineering data platforms, no unifying strategy and methodology is available yet, and the design is mainly left to the expertise of practitioners in the field. Service providers simply expose a long list of interoperable and alternative engines, making it hard to select the optimal subset without a deep knowledge of the ecosystem. A more effective approach to the design starts from the knowledge of the data transformation and exploitation processes that should be supported by the platform. In this paper, we sketch a computer-aided design methodology and then focus on the selection of the optimal services needed to implement such processes. We show that our approach lightens the design of data platforms and enables an unbiased selection and comparison of solutions even through different service ecosystems.

Keywords

Data Platform, Methodology, Big Data, Cloud Computing

1. Introduction

A data platform is a centralized infrastructure that facilitates the ingestion, storage, management, and exploitation of large volumes of heterogeneous data. It provides a collection of independent and composable services meeting the end-to-end needs of data pipelines, where: *centralized* means that a data platform is conceptually a single and unified component; *independent* means that changes in the implementation of a service do not affect other services; *composable* means that services have interfaces that enable an easy and frictionless composition; and *end-to-end* means that services cover the entire data life cycle. Data platforms foster collaboration and shared governance (being centralized, data is unified following some integration and it is easier to ensure compliance with data protection and privacy laws through shared security and access control), and scalability (being implemented on a distributed infrastructure, it is easy to add storage and computing resources as needed).

When it comes to building data platforms, no unifying strategy and methodology is there yet: building data platforms is mainly left to the expertise of practitioners or consultancy companies.

SEBD 2025: 33rd Symposium on Advanced Database Systems, June 16-19, 2025, Ischia, Italy

*Corresponding author.

alex.baiardi2@unibo.it (A. Baiardi); m.francia@unibo.it (M. Francia); enrico.gallinucci@unibo.it (E. Gallinucci); matteo.golfarelli@unibo.it (M. Golfarelli); manuele.pasini@unibo.it (M. Pasini)

0000-0002-0805-1051 (M. Francia); 0000-0002-0931-4255 (E. Gallinucci); 0000-0002-0437-0725 (M. Golfarelli); 0009-0005-7023-6742 (M. Pasini)

On the one hand, cloud service providers[1] offer ecosystems of services composed of many engines interoperable with each other; however, choosing the optimal set of services is hard since multiple solutions could fulfill the desiderata (e.g., whether disjoint databases and data warehouses or a single Lakehouse [1] should be used) and could require vertical knowledge on the design of data pipelines. On the other hand, several abstract big data architectures have been introduced (e.g., NIST [2], Lambda [3], and Kappa [4]). However, while they provide the necessary functionalities to enable big-data applications, their implementation and adoption require understanding which services should be used.

Neither providers of service ecosystems nor abstract architectures answer a crucial question: *given an ecosystem of services and the data-driven processes to support, which is the optimal subset of services enabling such processes?* We believe that to ease the data platform design, the description of data-driven processes should drive such activity. Indeed, data pipelines are the backbone of a *data* platform as they determine information-rich representations, and are congenial to designers since they encode many constraints on the choices to be made.

This paper discusses the following contributions (its extended version can be found at [5]): (i) a methodology to guide the selection of blueprints for (cloud) data platforms driven by clients' processes; (ii) the formulation of such selection as an optimization problem where preferences, affinity, and further constraints can be applied; and (iii) the extraction of blueprints from real case studies and validation with experts from consultancy companies. The paper is organized as follows: Section 2 sketches the overall methodology; Section 3 and Section 4 describe and formalize the design steps; Section 5 evaluates the effectiveness and efficiency of the contribution; and Section 6 analyzes the related literature. Finally, Section 7 draws the conclusions.

2. Methodology Overview

Our methodology (Figure 1) involves three types of users: *cloud service providers* (IT experts with in-depth knowledge of services available in the ecosystem); data platform *designers* (consultants or people with expertise in designing data flows but with no vertical knowledge on cloud/service ecosystems) and *clients* asking for the design of the data platform blueprint (e.g., partners involved in the same European project with expertise on the applicative domain —such as precision agriculture— but no expertise on IT and architectures).

The methodology aims to assist designers in selecting the services necessary to implement the clients' data pipelines out of the "unstructured" lists of services provided by cloud providers.

(1) Define the service ecosystem. A cloud service provider identifies *una tantum*: (i) the alternative candidate services to compose the blueprints of data platforms, and (ii) a taxonomy of tags that describe and characterize such services. The identified services are organized in a service graph describing the preferences and relationships between them as well as the tags characterizing each service. Assuming that cloud service providers do not offer redundant services, our guideline for designing the tag taxonomy emphasizes ensuring sufficient expressiveness.

[1]While data platforms are not mandatorily coupled with cloud computing, cloud computing is proving to be a winning business model since deploying and maintaining such a variety of computational resources and assets requires advanced technical skills that companies hardly have "in-house".

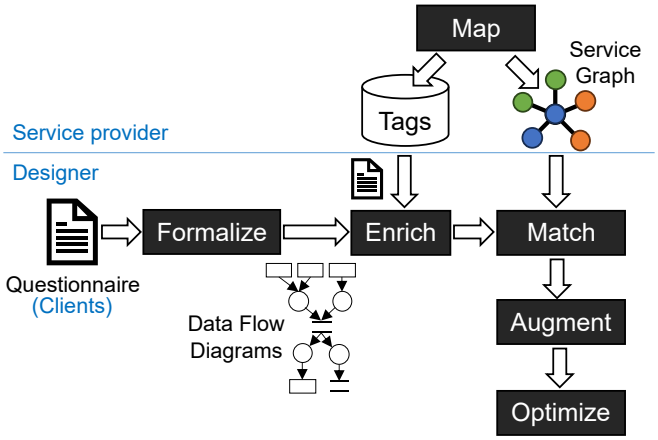

Figure 1: Overview of the methodology.

This means that each service should be uniquely identifiable by a set of tags.

(2) Formalize the requirements through a Data Flow Diagram. When building the blueprint of a new data platform, clients compile questionnaires that collect information about their data-driven processes and the main steps, subjects, and goals of their analysis. Note that the data platform should support (possibly independent) processes from multiple clients. Designers then refine the answers to the questionnaires and formalize the processes into a Data Flow Diagram (DFD); during the process, a few interviews with the clients might be necessary. We choose the DFD formalism [6] since it represents flows of data through an information system at a high level of abstraction, emphasizing the movement and transformation of data while hiding details such as decision points and interactions: knowing which repositories and processes compose the data-driven processes is enough to return a blueprint. To build the DFD, designers decompose the data flows into agents, processes, and repositories. Starting from an aggregated overview, our guideline is to recursively split candidate processes and repositories until each of them is characterized by homogeneous tags (e.g., if a repository contains both unstructured images and relational tables, split it into two homogeneous repositories). Repositories and processes that should not be imported into the data platform (e.g., legacy systems deployed on-premises) should be treated as external agents.

(3) Enrich the DFD with service tags. To match the DFD with the services deployed in a cloud ecosystem, the two must share the same characterization. Each process and repository in the DFD is enriched with the tags from the taxonomies previously identified by the cloud service provider. To aid designers in the characterization of each process/repository (agents are not subjects of enrichment, since they are out of the scope of the platform), clients answer an additional set of questions; we recall that clients are not required to have vertical knowledge about computer and data science or engineering. Such questions are defined by the cloud service providers and driven by the tag taxonomies. Note that since our final goal is selecting the most appropriate services, we are not interested in identifying and tracking the data/processes but rather their types and their flows.

Example 1. *Given a data repository, we ask the question: "What are the main types of collected*

data?"
☐ *Sensor data* ☐ *Images* ☐ *Videos* ☐ *Satellite observations* ☐ *Tables*
Answering "Satellite observations" tags the repository with the properties (Volume, Big), (Data Model, File), *and* (Data Nature, Raster) *(since earth observations are around 1 GB for 100* km^2*) and tags the process to download such data as* (Collection, Pull) *since files are downloaded from an FTP server [7].*

(4) Match the DFD and service graphs.
Once the DFD and service graphs are characterized by tags from the same taxonomies, it is possible to join them in a matched graph. A DFD process or repository matches (i.e., can be implemented by) a service only if the service has the same or more functionalities to fully implement it.

(5) Augment the matched graph. Augmentation is a semi-automatic step atop the matched graph. On the one hand, the system automatically recognizes architectural design patterns in the data pipelines to either automatically apply or simply highlight advanced compositions of services. On the other hand, designers can manually inject their knowledge to enrich the matched graph and steer the subsequent optimization. Designers could force the selection of services in case they are required by clients, prefer some services in place of others for performance or compatibility reasons, and refine the match (e.g., consider only services supporting programming languages required by legacy software).

(6) Select the optimal services. Out of all the services that are candidate implementations, it is necessary to select the minimal blueprint of the data platform that covers all the DFD entities (the fewer the services, the lower the cost and management efforts). Furthermore, dependencies and compatibilities between services have to be taken into account.

Case Study: Agritech Within the Agritech spoke of the NRRP European project [8], we deploy a data platform supporting prescriptive analytic tasks from 8 clients in the field of precision agriculture. We will use one of the partners as a working case study throughout the paper. Here follows a qualitative description of one of the clients' data flows that the platform must support.

Example 2. *The analysis entails a flow that is structured as follows. (i) Data comes from soil moisture sensor grids, weather stations, and SENTINEL-2 satellites; (ii) Sensor and weather data is uploaded every 15 minutes to the platform, while satellite data is periodically downloaded; (iii) Soil moisture data is interpolated using mathematical (e.g., bilinear interpolation) and machine learning (e.g., neural networks) techniques; (iv) Vegetation indexes are computed out of the raw satellite observations and integrated with the enriched sensor data; (v) Reports are periodically generated out of the integrated data; (vi) Given an optimal soil moisture matrix, the integrated data is used to decide how much to irrigate the soil.*

After gathering questionnaires from the clients, we (designers) iteratively refined the interview. Figure 2 depicts the tasks from Example 2 using the DFD formalism. More details are described in [9, 10].

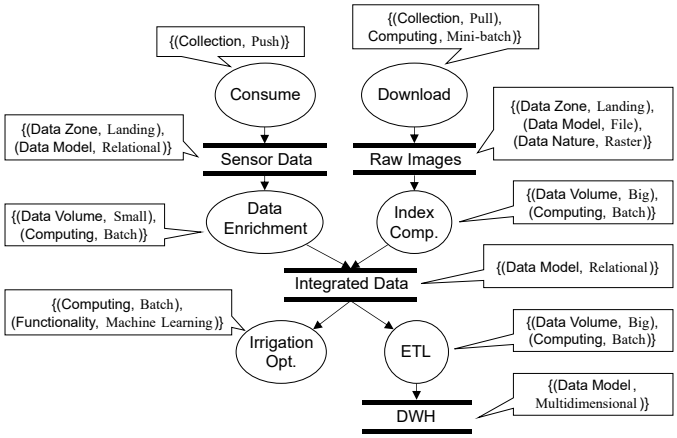

Figure 2: DFD from Example 2.

Table 1
Examples of taxonomies of tags

Data Model (All)	Structured	Relational
		Multidimensional
	Semi-structured	Document
		Wide-column
	Unstructured	Key-value
		Graph
		File (text, binary)
Data Nature (All)	Spatial	Vectorial
		Raster
	Temporal	

3. Mapping the Service Ecosystem

Tags (Table 1) are organized as a collection of hierarchies that can be fueled bottom-up from the documentation of cloud services providers (i.e., the set of tags that are attached to each service or that are inferrable through the service description), and top-down from the literature; e.g., the big data V's.

Definition 1 (Hierarchy)**.** *A* hierarchy h *is a taxonomy of categorical values* $Dom(h)$*.* $\geq_h$ *is the partial order that defines the taxonomy;* $\geq_h$ *indicates whether a value is more specific than another. We denote with* H *the set of hierarchies.*

Example 3 (Tags and hierarchies)**.** *Table 1 depicts an example of hierarchies* $H = \{\text{Data Model}, \text{Volume}, ...\}$*. In the partial order of the hierarchy* $h = \text{Data Model}$ *it is, for instance,* Relational $\geq_{\text{Data Model}}$ Structured $\geq_{\text{Data Model}}$ Data Model (All)*.* Relational *is more specific than* Structured *in the* Data Model *hierarchy.*

The services available on a specific ecosystem differ for each provider, and no single and shared organization of services is available. Also, it is necessary to consider the relationships between services (e.g., whether the adoption of a service requires another) and preferences in

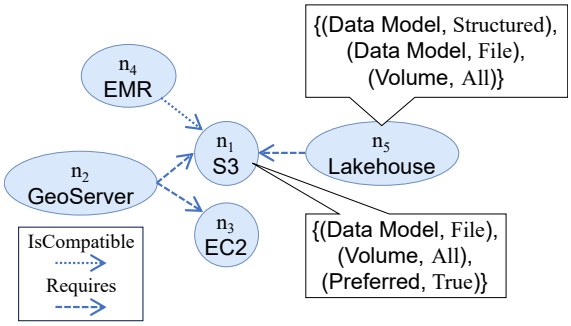

Figure 3: Excerpt of a service graph and its properties.

their choice (e.g., due to differences in performance reasons). That is why we map the services into a directed property graph.

Definition 2 (Directed property graph)**.** *A* directed property graph *is a tuple* $G = (N, A, P, L)$ *where* $N = \{..., n_i, ...\}$ *is a finite set of* nodes*,* $A = \{..., a_{ij}, ...\}$ *is a finite set of* arcs *connecting nodes* n_i *and* n_j*,* $P = \{..., (h, v), ...\}$ *is a set of key-value* properties *with* $h \in H$ *and* $v \in Dom(h)$*, and* L *is a set of* labels*.* $props : (N \cup A) \rightarrow P$ *returns the properties of a node or arc.* $label : (N \cup A) \rightarrow L$ *returns the labels of a node or arc.*

Definition 3 (Service graph)**.** *A* service graph *is a directed property graph* G^S *where nodes are labeled as* Service*, while arcs are alternatively labeled as* {Requires, IsCompatible, IsAkin}*. The semantics of the labels is the following.*

Service*: is any engine from the service ecosystem that can be optionally tagged with the property* (Preferred, True) *to specify whether it should be considered more than others;*

Requires*: represents whether a service mandatorily relies on another;*

IsCompatible*: represents whether a service natively interfaces with another (i.e., their interaction is supported by default and does not require custom/additional libraries or connectors);*

IsAkin*: represents whether a service not only is compatible with another but also if their adoption together is encouraged.*

An excerpt of the service graph G^S from the AWS ecosystem is depicted in Figure 3.

4. Process-Driven Match, Augmentation, and Optimization

We now describe three steps composing our approach. *Match* extracts the services that can support the execution of the data pipeline, *Augment* refines the potential candidates and, finally, *Optimize* picks the optimal (minimal) subset. We mainly focus on the optimization of the platform design rather than on how, starting from questionnaires, designers refine the DFD of such flows.

Since data pipelines are the backbone of data platforms, everything starts from the DFD describing the processes of clients' analysis.

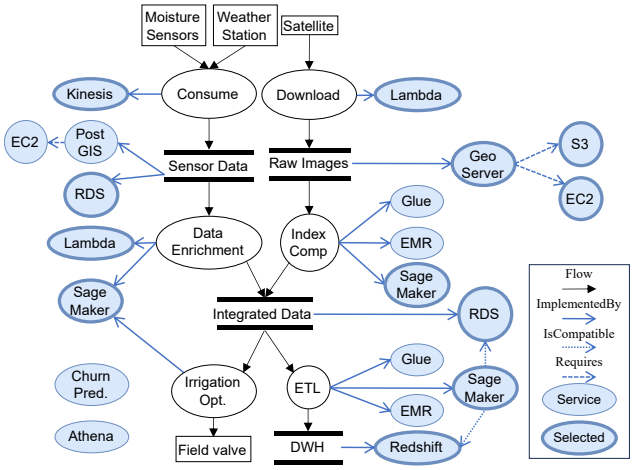

Figure 4: Matching the DFD (white) and service (blue) graphs; in bold the services selected as the optimal blueprint for the data platform design.

Definition 4 (Data Flow Diagram). *A* Data Flow Diagram *(DFD) is a directed property graph* G^D *where nodes are alternatively labeled as* $\{\text{Agent}, \text{Repository}, \text{Process}\}$*, while arcs are labeled as* Flow*. Arcs must connect with at least one* Process *node. The semantics of the labels is the following.* Agent*: an external entity that communicates with the system and stands outside of the system;* Process*: transform inputs to outputs;* Repository*: store data for later use;* Flow*: transfer data from one part of the system to another.*

Matching the DFD and Service Graphs Given the service graph and the DFD, we can automatically join them by matching their tags. A match represents whether a DFD node can be implemented using a specific service (solid arcs in Figure 4); i.e., services that have the same or more generic values for the properties specified in the DFD node. For instance, a DFD repository tagged as Relational could be implemented by services supporting Relational or all Structured data.

Definition 5 (Matched graph). *Given a DFD* $G^D = (N^D, A^D, P^D, L^D)$ *and a service graph* $G^S = (N^S, A^S, P^S, L^S)$*, a matched graph* $G^M = (N^D \cup N^S, A^D \cup A^S \cup A, P^D \cup P^S, L^D \cup L^S \cup \{\text{ImplementedBy}\})$ *is a directed property graph obtained as the union of* G^S *and* G^D*.* A *is an additional set of arcs, one for every match between nodes in* N^S *and* N^D*. Arcs in* A *are labeled as* ImplementedBy *and can have an optional property* (Mandatory, True) *to specify that such implementation must be selected in the final blueprint.*

Example 4 (Matched graph). *Figure 4 depicts an excerpt of the matched graphs for the DFD from Figure 2; for the sake of clarity, not all the arcs have been presented. Nodes from the DFD have a white background, while services are represented in blue. Solid arrows represent arcs labeled as* ImplementedBy *(e.g.,* Sensor Data *can be implemented by either* `PostGIS` *or* `RDS`*). Dotted arrows represent arcs labeled as* IsCompatible *(e.g.,* `SageMaker` *reads from and writes to* `Redshift`*). Dashed arrows represent arcs labeled as* Require *(e.g.,* `GeoServer` *requires* `EC2` *since it is deployed*

$$min \sum_i w_i s_i \text{ such that} \tag{1}$$

$$s_i \in \{0,1\} \text{ for all } n_i \in N, label(n_i) = \text{Service} \tag{2}$$

$$s_{ij} \in \{0,1\} \text{ for all } a_{ij} \in A, label(a_{ij}) = \text{ImplementedBy} \tag{3}$$

$$s_j \geq s_{ij} \text{ for all } s_{ij} \tag{4}$$

$$\sum_{a_{ij} \in A, label(a_{ij}) = \text{ImplementedBy}} s_{ij} = 1 \text{ for all } n_i \in N, label(n_i) \in \{\text{Process}, \text{Repository}\} \tag{5}$$

$$s_j \geq s_i \text{ for all } a_{ij} \in A, label(a_{ij}) = \text{Requires} \tag{6}$$

$$s_{ij} + s_{kh} \leq 1 \text{ for all } a_{ik} \in A, a_{jh} \notin A, label(a_{ik}) = \text{Flow}, label(a_{jh}) = \text{IsCompatible} \tag{7}$$

$$s_{ij} = 1 \text{ for all } a_{ij} \in A, (\text{Mandatory}, \text{True}) \in props(a_{ij}) \tag{8}$$

$$s_{ij} \cdot s_{kh} \geq s_h \text{ for all } a_{ik}, a_{jh} \in A, label(a_{ik}) = \text{Flow}, label(a_{jh}) = \text{IsAkin} \tag{9}$$

Figure 5: Selecting the optimal services.

on it). Of course, it is unlikely that all services of the cloud ecosystem will be matched, `Athena` *and* `Churn Prediction` *are examples of unmatched services.*

Augmenting and Pruning the Matched Graph Designing a platform is not just a technical task but encompasses various organizational, business, and political aspects that must be carefully considered. The successful creation of a platform requires the expertise and experience of designers to manipulate the matched graph and aid in the subsequent optimization. This ensures that the platform is not only technically sound but also aligns with broader strategic goals and user needs. Designers have several tools at their disposal to influence and refine the design process: (i) force the selection of specific services; (ii) prefer certain services over others; and (iii) refine the match by preliminarily removing services that do not meet specific criteria or adding services that were initially unmatched.

Example 5 (Mandatory and preferred services)**.** *With reference to Figure 4, imagining that our methodology is used to lift-and-shift an on-premises solution that already used* `GeoServer` *to AWS,* `GeoServer` *could be tagged with the property* (Mandatory, True) *since clients already have experience with it (and not with alternative engines such as* `GoogleEarth`*). Also, since* `S3` *is cheaper than other storage services (such as* `EBS` *and* `EFS`*),* `S3` *could be tagged with the property* (Preferred, True)*.*

Finally, this step supports the recognition of design patterns, detailed in the extended version [5], to ensure that the data platforms adhere to best practices and maintain a high standard maintainability.

Optimizing the Blueprint We use the minimization of the number of services as an objective function since this indicator correlates with both cost minimization and management simplicity.

More formally, given a matched graph $G^M = (N, A, P, L)$, the selection of the optimal blueprint can be modeled as a linear programming problem inspired by the standard *facility location problem.* Services are mapped to the potential locations while DFD nodes are mapped to the demand points.

The formulation in Figure 5 reads as follows. (1) The optimization function minimizes the weighted sum of the selected services. Weights w_i specify preferences for services s_i, we set $w_i = 0.5$ if (Preferred, True) $\in props(n_i), 1.0$ otherwise. (2) s_i are binary variables modeling services. $s_i = 1$ if the service is selected, 0 otherwise. (3) s_{ij} are binary variables modeling the services implementing DFD processes and repositories. $s_{ij} = 1$ if $n_i \in N$ s.t. $label(n_i) \in \{\text{Repository}, \text{Process}\}$ is implemented by $n_j \in N$ s.t. $label(n_j) = \text{Service}$. (4) Binding variables s_j and s_{ij}: service s_j is selected if it implements ($s_{ij} = 1$) a DFD node $n_i \in N$ s.t. $label(n_i) \in \{\text{Repository}, \text{Process}\}$. (5) *Coverage* constraints: every repository and process must be implemented by exactly one service. (6) *Dependency* constraints: if a service is selected, all the services it depends on must be selected. (7) *Compatibility* constraints: if two services are incompatible, they cannot be selected to implement two consecutive processes/repositories linked through a data flow in the DFM. (8) *Mandatory* constraints: if an implementation is mandatory and tagged with the property (Mandatory, True), s_{ij} is set to 1. (9) *Affinity* constraints: if the service s_h is selected and has affinity with another service s_j, then s_j is selected too.

Example 6 (Optimization)**.** *Given the matched graph from Figure 4, the optimal blueprint of the data platform is represented by the services highlighted in bold (with a cost of 7.5), namely* `Kinesis`, `Lambda`, `RDS`, `EC2`, `SageMaker`, `GeoServer`, `S3` *(tagged as preferred), and* `Redshift`. `GeoServer` *has been selected since it is the only available implementation for* Raw Images, `EC2` *and* `S3` *have been selected since they are required by* `GeoServer`. `SageMaker` *has been selected and preferred to* `EMR` *and* `Glue` *since three DFD processes can be implemented on it.*

5. Evaluation

For the exhaustive evaluation and implementation, refer to [5].

Validation with real use cases We applied our approach to two real companies: one of the largest Italian consulting firms specialized in data analysis and cloud data platforms and an international company leading in the production of smart and digitalized sports equipment. We (i) asked their experts for two use cases of data platforms, (ii) generated a blueprint for each use case, and (iii) asked the experts to evaluate and compare our blueprints against the ones they implemented.

Blueprints effectiveness. The blueprints returned by our system are compatible with the ones actually adopted by the companies and their overlap is 85%. Although the blueprints are not perfectly overlapping, experts claim that our blueprints are correct from a technical and technological point of view. However, what could be improved in our approach is the integration of more organizational, political, or even familiarity constraints that could bias the service selection.

Methodology effectiveness. The requirement analysis was smooth and the staff from the two companies had no problem describing the data-processes and answering questions needed for tagging. Both companies acknowledged that selecting services for a project is highly complex and that staying up to date about service provider offerings requires considerable time and effort. Overall the experts were enthusiastic about the approach and highlighted the following

strengths: (i) it can support and train young designers for the creation of blueprints, (ii) it can remove bias in the selection of services, and (iii) it is an objective approach to compare different blueprints.

Efficiency We test how our approach scales up to bigger cloud ecosystems and data pipelines by implementing synthetic DFDs with increasing nodes $|N^D|$ that share a service ecosystem of 200 services and we ensure that every node in the DFD is implemented by at least one service. With $|N^D| = 250$ the total computational time is 7.76 seconds, of which 7.23 for match and 0.53 for selection. Matching could be improved by further optimizing its implementation on GraphDB.

6. Related Work

Understanding which "building blocks" should be part of a data platform is nontrivial. NIST designed the Big Data Reference Architecture [2] which comprised vendor-, technology- and infrastructure-independent logical functional components that are necessary for managing and processing big data (data provider, data consumer, system orchestrator, big data application provider, and big data framework provider). Lambda [3] is an architecture designed to handle massive quantities of data by taking advantage of both batch and stream-processing methods. Kappa [4] overcomes some limitations (e.g., "redundant" batch and streaming implementations) of Lambda by using a pure streaming approach with a single code base. While these architectures define functional components necessary for (big) data platforms, they are abstract and do not address the problem of selecting the optimal services necessary for deploying working data platforms.

Cloud service providers such as Amazon [11], Google [12], and Microsoft [13] provide ecosystems of independent and interoperable services. While ecosystems enable easy deployment of data pipelines, choosing the *optimal* services is hard: (i) each provider offers different services, (ii) a shared categorization and organization of services is missing, and (iii) it is hard for the users to understand which services are needed based on their data-driven processes [14]. Some multi-objective optimization techniques have been proposed, where users express requirements and preferences (e.g., minimal QoS) about single services [15, 16, 17], cloud deployment models (public and private) [18], and service providers [19, 20]. Finally, domain languages and ontologies have been introduced (e.g., [21]) to enable some service composition (e.g., [22]).

7. Conclusion and Future Works

In this paper, we have discussed a methodology to aid designers in selecting the optimal services (out of a service ecosystem) supporting data-driven processes from multiple stakeholders (e.g., partners in a research project), and we have addressed such selection as a facility location optimization problem. Our approach can match and select single services as well as design patterns (e.g., Lakehouse to replace both data lakes and warehouses), it supports additional constraints (e.g., mandatory and affinity), and the produced blueprints have been successfully compared with the ones recommended by expert designers.

We believe that the expressiveness of the approach can still be improved as follows. *Expressiveness*: introduce more advanced constraints to support organizational and political factors in the design of the blueprints. *Resource provisioning*: additionally to selecting the services, a complete approach should also consider how many instances of a service are required (to do so, a cost model should be studied). *Metadata integration*: while catalog and meta-data management services do not directly introduce functionalities for data transformation and exploitation, the design should also recommend services helping in the management of the platform itself [23].

References

[1] M. Zaharia, A. Ghodsi, R. Xin, M. Armbrust, Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics, in: CIDR, www.cidrdb.org, 2021, p. 8. URL: http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf.

[2] W. Chang, N. Grady, N.-P. NIST, NIST big data interoperability framework: Volume 2, big data taxonomies [version 2], 2018. doi:https://doi.org/10.6028/NIST.SP.1500-2r1.

[3] M. Kiran, P. Murphy, I. Monga, J. Dugan, S. S. Baveja, Lambda architecture for cost-effective batch and speed big data processing, in: BigData, IEEE, Santa Clara, CA, USA, 2015, pp. 2785–2792. doi:10.1109/BIGDATA.2015.7364082.

[4] J. Kreps, Questioning the Lambda Architecture, volume 2, O'Reilly, 2014.

[5] M. Francia, M. Golfarelli, M. Pasini, Process-driven design of cloud data platforms, Information Systems (2025) 102527.

[6] T. DeMarco, Structured Analysis and System Specification, Yourdon press, 1978.

[7] Earth observation guide, https://business.esa.int/newcomers-earth-observation-guide, 2020. Accessed 2024-07-26.

[8] Agritech, https://agritechcenter.it/, 2024. Accessed 2024-07-26.

[9] M. Francia, J. Giovanelli, M. Golfarelli, Multi-sensor profiling for precision soil-moisture monitoring, Computers and Electronics in Agriculture 197 (2022) 106924. doi:10.1016/j.compag.2022.106924.

[10] E. Baldi, M. Quartieri, G. Larocca, M. Golfarelli, M. Francia, J. Giovanelli, E. Xylogiannis, M. Toselli, Smart irrigation system for precision water management: effect on yield and fruit quality of yellow fleshed kiwifruit in northern italy, in: ECPA, Wageningen Academic Publishers, 2023, pp. 59–66. doi:10.3920/978-90-8686-947-3_5.

[11] Amazon web services, https://aws.amazon.com/, 2024. Accessed 2024-07-26.

[12] Google cloud platform, https://cloud.google.com/, 2024. Accessed 2024-07-26.

[13] Microsoft azure, https://azure.microsoft.com/en-us, 2024. Accessed 2024-07-26.

[14] L. Sun, H. Dong, F. K. Hussain, O. K. Hussain, E. Chang, Cloud service selection: State-of-the-art and future research directions, J. Netw. Comput. Appl. 45 (2014) 134–150. doi:10.1016/J.JNCA.2014.07.019.

[15] E. Cavalcante, F. Lopes, T. V. Batista, N. Cacho, F. C. Delicato, P. F. Pires, Cloud integrator: Building value-added services on the cloud, in: NCCA, IEEE, Toulouse, France, 2011, pp. 135–142. doi:10.1109/NCCA.2011.29.

[16] M. H. Nejat, H. Motameni, H. Vahdat-Nejad, B. Barzegar, Efficient cloud service ranking

based on uncertain user requirements, Clust. Comput. 25 (2022) 485–502. doi:10.1007/S10586-021-03418-W.

[17] L. Lu, Y. Yuan, A novel TOPSIS evaluation scheme for cloud service trustworthiness combining objective and subjective aspects, J. Syst. Softw. 143 (2018) 71–86. doi:10.1016/J.JSS.2018.05.004.

[18] R. Garg, Mcdm-based parametric selection of cloud deployment models for an academic organization, IEEE Trans. Cloud Comput. 10 (2022) 863–871. doi:10.1109/TCC.2020.2980534.

[19] M. B. Kar, R. Krishankumar, D. Pamucar, S. Kar, A decision framework with nonlinear preferences and unknown weight information for cloud vendor selection, Expert Syst. Appl. 213 (2023) 118982. doi:10.1016/J.ESWA.2022.118982.

[20] A. Tomar, R. R. Kumar, I. Gupta, Decision making for cloud service selection: a novel and hybrid MCDM approach, Clust. Comput. 26 (2023) 3869–3887. doi:10.1007/S10586-022-03793-Y.

[21] M. Noura, M. Gaedke, Wotdl: Web of things description language for automatic composition, in: Int. Conf. on Web Intelligence, ACM, Thessaloniki, Greece, 2019, pp. 413–417. doi:10.1145/3350546.3352558.

[22] H. B. Mahfoudh, A. Caselli, G. D. M. Serugendo, Learning-based coordination model for on-the-fly self-composing services using semantic matching, J. Sens. Actuator Networks 10 (2021) 5. doi:10.3390/JSAN10010005.

[23] M. Francia, E. Gallinucci, M. Golfarelli, A. G. Leoni, S. Rizzi, N. Santolini, Making data platforms smarter with MOSES, Future Generation Computer Systems 125 (2021) 299–313. doi:10.1016/j.future.2021.06.031.