Received: 1 October 2018 | Revised: 12 December 2018 | Accepted: 6 February 2019
DOI: 10.1002/cpe.5212

WILEY

SPECIAL ISSUE PAPER

Interoperable and network-aware service workflows for big data executions at internet scale

Pradeeban Kathiravelu[1,2,3] | Peter Van Roy[3] | Luís Veiga[2]

[1]Emory University, Atlanta, Georgia
[2]INESC-ID Lisboa/Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
[3]Université Catholique de Louvain, Louvain-la-Neuve, Belgium

Correspondence
Pradeeban Kathiravelu, Emory University, Atlanta, GA 30322; or INESC-ID Lisboa/Instituto Superior Técnico, Universidade de Lisboa, 1049-001 Lisbon, Portugal; or
Université Catholique de Louvain, 1348 Louvain-la-Neuve, Belgium.
Email: pradeeban.kathiravelu@emory.edu

Present Address
Pradeeban Kathiravelu, 101 Woodruff Circle, Atlanta, GA 30322

Funding information
National funds through Fundação para a Ciência e a Tecnologia, Grant/Award Number: UID/CEC/50021/2013; PhD grant offered by the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC), Grant/Award Number: 2012-0030

Summary

Sharing of computing resources and workload across different big data frameworks is challenging due to their lack of interoperable interfaces. In contrast, web services natively support an interoperable execution. Therefore, an increasing number of big data workflows are composed of data services and web service implementations that access and process big data. On the other hand, big data execution in the wide area networks needs to minimize latency and communication overheads to be able to scale seamlessly. Lack of network-awareness of classic web service execution beyond data centers significantly challenges the scope of data services.

Software-Defined Networking (SDN) offers better control and management to the network, by unifying the control plane centrally, away from the distributed data plane devices. In this paper, we propose Software-Defined Data Services (SDDS), an SDN-based distributed service composition and workflow placement approach for data services in wide area networks. We first present the design of an SDDS framework that models the big data executions as composable data service workflows in multi-domain network environments. We then evaluate the performance of a prototype SDDS framework through microbenchmarks. The benchmarks highlight the efficiency of SDDS in data service execution inside and beyond data centers.

KEYWORDS

big data, service composition, Software-Defined Networking (SDN), web services

1 | INTRODUCTION

Big data frameworks consist of diverse storage media, heterogeneous data formats, and a large volume of data from several scientific and enterprise domains.[1] Resource scarcity in any single server prevents processing of a vast amount of data on time in a centralized manner in a server. Therefore, big data processing is often performed in a distributed and parallel manner, due to the volume and variety of big data.[2] Big data is processed either on top of in-memory frameworks such as In-Memory Data Grids (IMDGs),[3] on the disk in data stores such as relational databases, or on hybrid architectures consisting of persistent data on disk as well as cached data and computations in-memory.[4] Such an execution leads to a significant amount of data processed and transferred across multiple computing nodes.[5] Thus, it often incurs a considerable bandwidth cost and communication overhead, even inside a cluster or a data center.[6] Distributed execution frameworks such as IMDGs aim to minimize the performance degradation associated with communication and coordination overheads, by reducing, if not avoiding, unnecessary data movements between the execution and storage nodes.[7]

Enterprise big data executions are typically confined to their platforms, and cannot be shared with other frameworks, due to the independent development of the big data platforms and incompatibility across their executions and interfaces.[4] Even though a big data application is often built for a cluster or a data center network, big data workflows can indeed extend beyond the boundaries of a data center, and continue to do so, more and more in recent times.[8] On the other hand, multiple network providers manage the wide area networks such as the inter-cloud and edge networks. Thus, centralized and unified control is challenging concerning both technologies as well as administration and policies. This state of affairs indeed hinders the potential of the big data execution frameworks to efficiently distribute the workload and share the resources beyond a data center or a cloud.

While research efforts such as volunteer computing,[9] namely, at the edge, have leveraged resource sharing and workload dissemination across independent and decentralized execution nodes, their use is limited to specific applications, and they are usually not aimed at enterprise big data executions. Initially proposed for CPU-bound workloads, volunteer computing and cycle sharing have recently been extended for data-intensive applications.[10] However, volunteer computing approaches significantly lack the potential to distribute data-intensive workloads in a network-aware manner, as the central coordinators in such approaches dare developed with minimal control over the execution nodes for less intrusion[11] and easy integration with computing resources of independent participants.[12]

Service compositions

Service compositions[13] facilitate complex computation-intensive eScience workflows and enterprise business processes by chaining the outputs of several web services[14] and microservices.[15] Service composition workflows can, and often should, execute on computational nodes that are geographically distributed, due to the increasing scale and distribution of big data and the services that access and process it. With the abundance of service deployments, service workflows face the challenge of finding the optimal service instances for their execution among these various geo-distributed nodes. Web services are typically designed for and deployed over web services engines. These web services engines periodically monitor and store the web service health statistics such as how many requests a service deployment executed, and how many are on the fly. Service workflows typically exploit this information for their context-aware execution.[16] Service execution should be able to perform inter-domain migration between possible service deployments, during a service failure or congestion, for resilient workflow execution. However, such a migration causes communication overhead when the workflows involve big data, due to massive data migrations, therefore making the classic service composition approaches less suitable for big data workflows.

While web services are developed following standards and descriptive languages such as the Web Application Description Language (WADL)[17] and Web services description language (WSDL)[18] as well as protocols such as the Representational State Transfer (REST)[19] and SOAP (formerly, Simple Object Access Protocol),[20] such practices and protocols are lacking in the development of big data execution frameworks. Data services are designed as web service implementations that access and process big data to bring the standardization and interoperability of the web services to big data. Data services offer service-based Application Programming Interfaces (APIs) to big data storage and execution frameworks and provide a standardized interoperable execution model across various frameworks. Their interoperable APIs facilitate composable data service chains, where the output of one or more data services is fed as the input for one or more data services. This potential for service compositions supports extensible and reusable big data workflows. Consequently, data services offer the best of both worlds from web services and big data frameworks to the execution of big data workflows in the wide area networks.

However, these services are often developed and deployed independently at several locations that are neither connected nor centralized. Finding the service deployments and chaining their execution outcomes for a large-scale workflow is impossible without prior knowledge of their existence and access mechanisms. Service registry specifications[21] such as Universal Description, Discovery, and Integration (UDDI)[22] and Electronic Business XML (ebXML)[23] identify and store the web service engines and service instances to enable easy discovery of the service endpoints. Service registry implementations such as Apache jUDDI[24] implement the registry specifications for web services following SOAP and REST protocols. However, current service registry implementations lack network-awareness at Internet scale. Therefore, we cannot entirely rely on them to find the service instances to compose data service workflows.

Software-defined networking

Software-defined networking (SDN) has facilitated programmability and control of the network and executions inside and beyond data centers.[25] It equips a logically unified controller with a global view of the network data plane devices. Through their service-based approach, data services increase the controllability and interoperability of big data workflows, compared to the classic data management frameworks. Given the potential of SDN and its extension to control multi-domain wide area networks,[26] we posit that big data workflows can execute in a decentralized and distributed environment spanning data centers, by extending and leveraging SDN for managing the decentralized data services. Several enterprises such as IBM,[27] NEXION Networks,[28] and Catalogic[29] offer Software-Defined Data Services (SDDS), a data service execution model to manage the big data workflows in a centralized and unified manner, while abstracting the execution of data services away from the data storage. Nevertheless, the existing SDDS offerings focus only on one or a just few specific data services. They do not propose a generic software-defined approach for data services. Furthermore, a vast majority of current SDDS frameworks do not leverage SDN. While enterprises and research exploit SDN for big data, typical applications are limited to the data centers and are confined to a single big data application due to the interoperability constraints. The research findings on leveraging SDN for service compositions[30] are not optimized for big data workflows, as they focus more on the standard web services. Subsequently, comprehensive research to understand how data services and big data workflows can benefit from SDN is still lacking.

Big data workflows at internet scale

Recent developments in SDN and Software-Defined Wide Area Networks (SD-WAN)[31] have opened up the potential for network-aware service compositions in the edge and multi-cloud environments. While geographical proximity is one deciding factor in picking the service composition

for the workflows, the existence of dedicated connectivity between two geographically distributed servers plays a significant role in finding the best instances. Cloud providers such as Amazon Web Services (AWS) offer Cloud Direct Connects between the server of a user and the data centers of a cloud region, typically the nearest to the user server. Moreover, Multiprotocol Label Switching (MPLS)[32] providers connect the enterprise users to their remote servers with high bandwidth, without having to go through the public Internet, which is typically slow and thus hinders the performance of the organizational workflows between the distributed locations. Previous research by Kathiravelu et al[33] has presented an architecture for a cloud-based overlay connectivity provider with up to 30% latency reduction in data transfers. These findings highlight the potential for the service providers to leverage their dedicated networks across the Internet regions.

Direct connect services that offer the Internet fast paths[34] are getting more mainstream, prominently for data-intensive workflows. Cloud providers such as AWS leverage their backbone to route their entire data traffic, without relying on external Internet paths or other connectivity providers, in most of the cloud regions. While such networks are readily available for each vendor, currently, they are not shared across the vendors or third-party users. Even though Internet Service Providers (ISPs) and Internet eXchange Points (IXPs)[35] are more equipped with the potential to alter the network paths across the autonomous systems and differentiate the network flows, social aspects of the Internet, and net neutrality regulations prevent them from implementing these research avenues globally, without hindering a large segment of the existing end-users.

There have been increasing proposals for a *network softwarization*,[36] a complete virtualization of the network, to facilitate dynamic formation, efficient configuration, incremental deployment, and seamless migration of network architectures through software constructs.[37] While network softwarization has made promising improvements to network management,[38] it limits its focus mainly to data center networks. To reap the benefits of network softwarization for Internet services, we need a "bridge" between the networking management capabilities of network softwarization, and the service composition and scheduling capabilities handled at the application level by the service providers. Currently, the potential for extending the paradigm of network softwarization for data services, and its impact beyond networks on service composition and workflow placement at Internet scale, remain mostly unexplored.

Motivation

Given the aforementioned premises, we aim at addressing the following research questions in this paper.

- (RQ_1) Can we extend and adopt SDN and web services paradigms as a generic software-defined approach for interoperable and network-aware big data executions?
- (RQ_2) Can such a software-defined approach enhance the performance, management, and scheduling of the data service workflows at various scales from data centers to the Internet?
- (RQ_3) Can network domains advertise and share their dedicated network links among each other for dynamic data service compositions, through enterprise communication protocols such as Message-Oriented Middleware (MOM)[39] protocols?

Contributions

The goal of this paper is to answer the identified research questions. The main contributions of this paper are as follows:

1. Software-Defined Data Services (SDDS), a big data execution model, to generalize the big data applications, including storage and processing, as data services, and execute them across multi-domain wide area networks (RQ_1).
2. A network-aware execution of big data workflows, leveraging the Internet paths as well as the direct links provided by various enterprises, such as the cloud direct connects (RQ_3).
3. *Mayan-DS*,* an SDDS framework that aims to solve the challenges that hinder efficient and interoperable big data executions inside and beyond data center networks (RQ_2).

First, *Mayan-DS* defines the big data workflows as data services, adopting the interoperability offered by the web service definitions across heterogeneous big data environments. It enables the execution to be agnostic to the storage media, format, and location, by defining each step of the execution as interoperable data services. Second, it exploits SDN for the performance, scalability, and bandwidth efficiency of the data services execution in the wide area network. By deploying a federated multi-domain controller over a wide area network, the SDDS controller receives a global overview of the data service instances comprising the workflows. By leveraging both the service statistics of the web service engines and the network status from the SDN controller, the *Mayan-DS* controller architecture schedules the data services composing the big data workflows in an interoperable and network-aware manner.

In our previous work,[40] we presented a preliminary version of SDDS for interoperable and network-aware big data executions in data center environments. In this paper, we address the extension of SDDS from data centers to multi-domain wide area networks and elaborate the architecture and algorithms of our SDDS framework. We further evaluate the performance of SDDS through microbenchmarks on a prototype implementation of *Mayan-DS*. Our evaluations highlight that *Mayan-DS* enhances the scalability and performance of big data executions inside data centers and wide area networks, through interoperability facilitated by data services, and network-awareness of an extended SDN architecture.

* *Mayan* is a mythical architect of Tamil literature known for his skills in constructing resilient large-scale structures and building foundations.

Paper organization

In the following sections, we further discuss the SDDS approach and the *Mayan-DS* framework. Section 2 elaborates the SDDS model and *Mayan-DS* architecture. Section 3 presents the algorithms of *Mayan-DS*. Section 4 describes the prototype implementation of our SDDS framework. Section 5 evaluates the performance of SDDS by benchmarking the prototype implementation of *Mayan-DS*. Section 6 presents the related work on SDDS. Finally, Section 7 presents the open challenges in implementing and deploying a complete SDDS infrastructure at the Internet scale, and Section 8 concludes this research with the summary and future work.

2 | *MAYAN-DS* MIDDLEWARE FOR SOFTWARE-DEFINED DATA SERVICES

In this section, we look into the core of the *Mayan-DS* architecture, and how *Mayan-DS* executes the data services in a network-aware manner in a wide area network as an SDDS framework. Section 2.1 presents our SDDS model. Section 2.2 elaborates the *Mayan-DS* middleware. Section 3 presents the core *Mayan-DS* algorithms that provide the network-aware data service executions.

2.1 | SDDS model for network-aware big data workflows

Due to the multiple service implementations and deployments, a workflow can execute using one of the several service composition options. Therefore, there are numerous alternative network paths for the workflow execution. An SDDS framework must avoid unnecessarily dispersing the related data and execution across several nodes, or pulling the data to the execution. Data should be distributed across nodes or data centers only when such distribution yields better performance. Therefore, when all the services composing a service composition are present in a given data center or a cloud, the workflow typically executes inside the data center or the cloud, without resorting to inter-domain data transfer and communication. *Mayan-DS* builds on this principle, along with its network-awareness inside and across data centers, for an efficient SDDS execution.

Mayan-DS aims at minimizing the distance *l* between any related data objects {i, j} among the data sets of interest *D* that are typically identified by the user. It further aims at the placement of the service execution *e* on a node *n* such that the sum of the distance between the data and the execution remains minimal. The distance is determined by a utility function including the network properties such as bandwidth, throughput, execution time, and latency, rather than a mere physical distance between the service endpoints. Equation (1) illustrates the distribution of the data and execution across the servers in *Mayan-DS*

$$\forall i, j \in D; \textbf{minimize}\left(A(n, D) : A(n, D) = \iint_{|D| \subseteq \xi} \left(l_{i,j} + l_{n,j}\right) d\xi\right). \tag{1}$$

Here, ξ represents the entire area covered by the data relevant to a given data service. ξ measures how far the execution and the related datasets (such as the data stored and accessed by a particular service, a given user, or data sets that are frequently accessed together by the data services) are spread out. The spread of the set of data D is denoted by $|D|$, which is a subset of ξ, as the data can spread only across the servers identified by the SDDS framework. Therefore, the SDDS framework attempts to minimize the total area A(n, D), denoting the distance between the execution server *n* and the related objects from the set of data objects *D*. We depict the total area covered by the data objects and the execution node as a double integral of distances between the related objects as well as the execution server for the considered data sets. Previous work has proposed storage efficiency through efficient data placement to minimize data migration.[41] Sharing the same motivation of efficient data placement, SDDS proposes storing of data in indexed data structures in unified storage and accessing it via service interfaces to ensure that A(n, D) is effectively minimized. While Equation (1) presents the efficient data placement as a minimization of a surface area, represented by a double integral of the lengths (ie, distances between pairs of data sets or between data and the execution), the lengths can be extended to include weights that represent network parameters such as bandwidth and jitter, as well as additional properties such as monetary cost.

Figure 1 depicts a sample representation of data and execution placement in *Mayan-DS*. By default, *Mayan-DS* measures the distance l by end-to-end latency (which can be continuously monitored and computed by a ping). However, this utility function that determines the perceived distance between any two service endpoints can be extended to consider other network properties such as the number of hops, throughput, and jitter, as we observed earlier. We note that the geographical distance indeed plays a decisive role; for instance, a pair of servers inside a single data center offers better latency in serving a single big data workflow, compared to two identical but geographically distributed cloud data centers. On the other hand, a direct dedicated link between two geographically distributed servers can offer minimal latency compared to two servers connected via the public Internet despite the geographical distance, due to factors such as the bandwidth of the network, the number of network users and flows sharing the route, and the unreliability of the Internet paths.

A distance value of 0 is given to the placement inside the same server, whereas positive values of α, β, and γ are given to placement in different servers of the same rack, servers of different racks, and servers of different data centers, respectively. In the sample representation of Figure 1, $\alpha = 1; \beta = 10; \gamma = 100$. *Mayan-DS* aims to minimize the total inter-distance among the data objects and the execution through its data and service placement. Even though the sample representation in Figure 1 gives a constant value to γ, in practice, it remains a variable. For example,

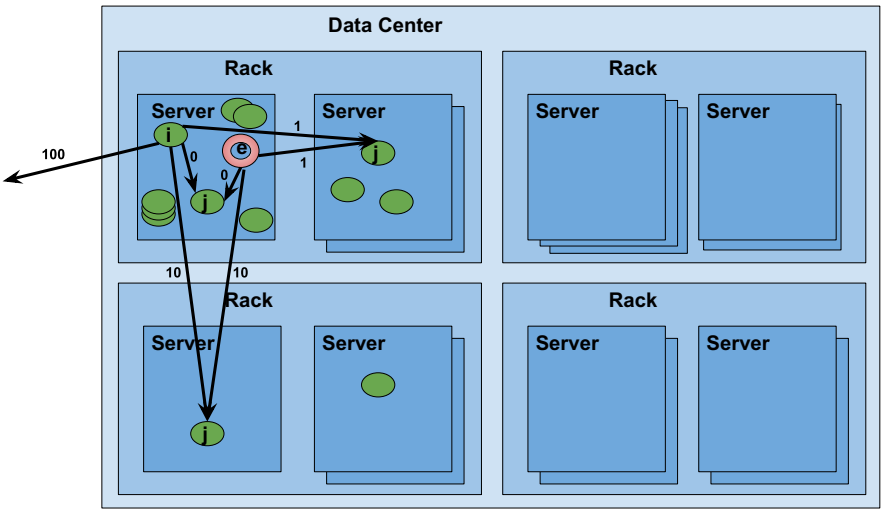

FIGURE 1 Sample representation of data and execution placement

cloud providers own several servers spanning data centers. Some servers are in the same data center (D_o). Some are in the same availability zone, ie, data centers (D_o and D_d) connected by high-bandwidth low-latency network links. Each network region has multiple availability zones (a_o and a_d). Network traffic between regions (r_o and r_d) incur a large latency that depends on the number of hops between the regions and the nature of the backbone network that connects the regions.[33] Moreover, if the different regions are connected by a low-latency dedicated link (distance value of $\gamma(r_o, r_d)_D$), they perform better than being connected via the public Internet (distance value of $\gamma(r_o, r_d)_I$). Equation (2) illustrates a simplified version of the network distance. In practice, these distances are substituted by weights based on the network link properties

$$(\gamma : \gamma(r_o, r_d)_I > \gamma(r_o, r_d)_D > \gamma(a_o, a_d) > \gamma(D_o, D_d)) > \beta > \alpha > 0. \tag{2}$$

When an inter-domain service execution is required, *Mayan-DS* seeks optimal service scheduling, including network properties such as low latency and high throughput. We denote an Internet-based path between the services s_o and s_d by $s_o \cdot s_d$. We denote a high-speed or low-latency dedicated link between any service s_i and another Internet server[†] $s_{\bar{i}}$ by $\overrightarrow{s_i \cdot s_{\bar{i}}}$. We define the set of data flow paths $P_{(o,d)}$ of the service composition workflow w(s) by its origin (o) and destination (d) servers in the service composition $s_o \circ s_d$, assuming the servers are connected via the Internet paths, as shown by Equation (3)

$$P_{(o,d)} = s_o \cdot s_d. \tag{3}$$

The path $P_{(o,d)}$ can be replaced with an alternative path $P'_{(o,d)}$, including dedicated links as well as Internet paths. Thus, we expand $P'_{(o,d)}$ as shown by Equation (4), considering the potential existence of direct connectivity between an endpoint and an intermediate node. Thus, we can iteratively expand a path into sub-paths consisting of direct dedicated paths and Internet paths

$$E(P_{(o,d)}) = P'_{(o,d)} = s_o \cdot s_{\bar{o}} + \overrightarrow{s_{\bar{o}} \cdot s_{\bar{d}}} + s_{\bar{d}} \cdot s_d. \tag{4}$$

Equation (5) denotes that, if the server $s_{\bar{i}}$ is the same as the server of s_i, the value represented by the path $s_i \cdot s_{\bar{i}}$ is reduced to 0

$$\forall s_i \in E(P_{(o,d)}); s_i \equiv s_i \Rightarrow s_i \cdot s_{\bar{i}} = \vec{0}. \tag{5}$$

Thus, we observe $P'_{(o,d)}$ as a generic form of an execution path of the data service workflow, concerning two service endpoints, as illustrated by Equation (6)

$$P_{(o,d)} \subset E(P_{(o,d)}). \tag{6}$$

Consequently, the execution path of a larger service composition workflow, consisting of more than two services, can be expanded as shown by Equation (7)

$$W = s_o \circ s_{i_1} \circ \ldots \circ s_{i_n} \circ s_d \Rightarrow P'_W = P'_{(o,i_1)} + P'_{(i_1,i_2)} + \ldots + P'_{(i_n,d)}. \tag{7}$$

Mayan-DS consists of an initialization procedure and a data scheduling procedure that leverage these equations for efficient data placement and data service workflow execution. First, *Mayan-DS* initialization procedure (presented in Section 3.1) exploits the constructs provided by the platforms such as IMDG in ensuring related objects stay closer in a locality-aware manner, as illustrated by Equation (1). The initialization

[†] In calculating the efficient paths, we use the notions "server" and "service" interchangeably, assuming any server in a data service ecosystem hosts a service, at least a unity service that returns or forwards the same input as its output, as in a cloud router. Furthermore, we assume each server to host a single data service or a set of related services that form a related data service composition.

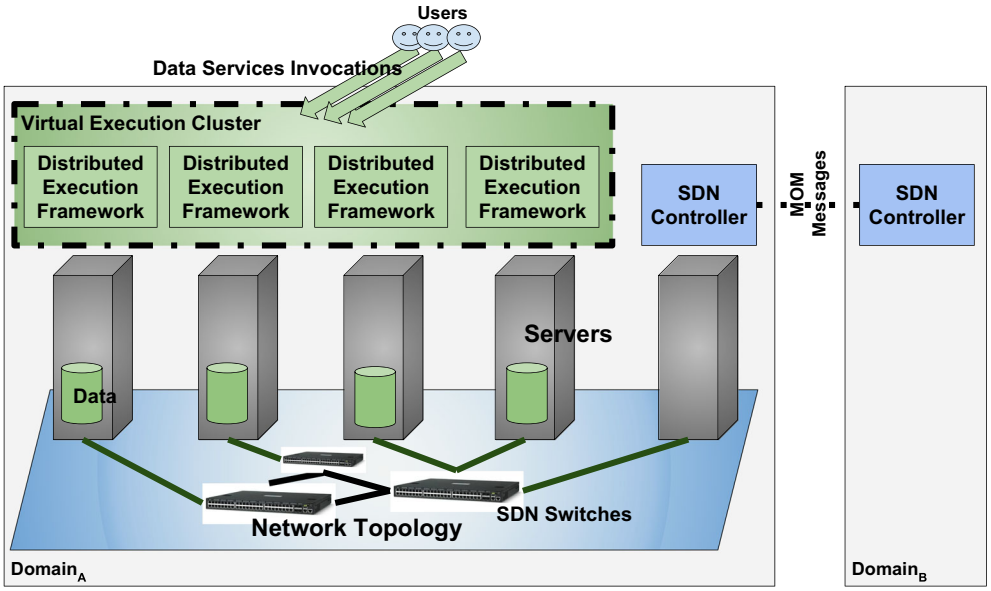

FIGURE 2 A sample *Mayan-DS* deployment

procedure then identifies the potential workflow execution paths as a set $E(P_{(o,d)})$ as denoted by Equation (6) by exploiting MOM. Finally, *Mayan-DS* data service scheduling procedure (presented in Section 3.2) leverages the best-performing option among the potential paths identified by the Equation (7) to schedule the data service execution in a network-aware manner, exploiting SDN in the wide area environment. *Mayan-DS* focuses on latency on deciding which of the options is the best for a big data workflow. However, the workflow scheduling approach can be extended to consider other parameters and user policies.

2.2 | *Mayan-DS* solution architecture

In its core, the *Mayan-DS* framework consists of a federated deployment of SDN controllers that spans multiple network domains. Typically, a domain represents a data center or a networked environment that is managed by a single vendor. The federated deployment denotes a set of servers and topology of SDN switches that are controlled by an SDN controller in each domain. The controllers communicate with the controllers of the other domains through MOM messages to propagate status updates on the network and bandwidth health statistics. The messages are limited to the ones relevant for each domain, based on the subscriptions of each controller. Thus, each controller achieves a limited access to the network topologies beyond its domain.

Figure 2 illustrates a sample deployment of *Mayan-DS* in a data center along with inter-domain communication between the controllers via MOM messages. Users invoke the data services individually, or as part of a larger workflow. The data services are hosted on the **Distributed Execution Frameworks** to support dynamic execution of the service requests on the instance with the best available resources. As the core of the *Mayan-DS* execution, the SDN controller is typically hosted on a dedicated server to avoid overloading the server with other computing workflows and applications. The other servers in the domain host the distributed execution frameworks such as an IMDG. These IMDG instances form a **Virtual Execution Cluster**, an execution environment at the application level. Distributed big data executions are usually stateful. Therefore, once a given instance serves a client request, it continues to receive and serve all the subsequent requests. This stateful execution enables lossless and continuous big data workflows with minimal communication overhead.

It is still possible to host the controller in a server that is also shared by the distributed execution framework, especially when either the number of servers or the workload of the distributed execution framework is minimal. The virtual execution cluster is aware of the network status, through its communication with the SDN controller. Status changes such as an interruption in data service availability and congestion in a workflow execution path are identified through the web services engines and passed to the controllers that have subscribed to the topic.

Mayan-DS leverages the global network knowledge of the SDN controller to find the best service instance among various potential service deployments for the service workflow execution. Each controller instance monitors and stores several metrics, including the service execution completion time, the number of service failures, and latency between the endpoints. The controllers compute and propagate the inter-domain network properties including the nature of the link between two endpoints in different domains such as bandwidth and latency and propagate them via MOM messages between controllers across the network domains. These messages include the update on the availability of a server or a data service initialized in the server. These metrics and events can be extended with the information received via the SDN protocols of the controller as well as the web service engines. Figure 3 elaborates the deployment architecture of *Mayan-DS* consisting of the controllers and services in a wide area network.

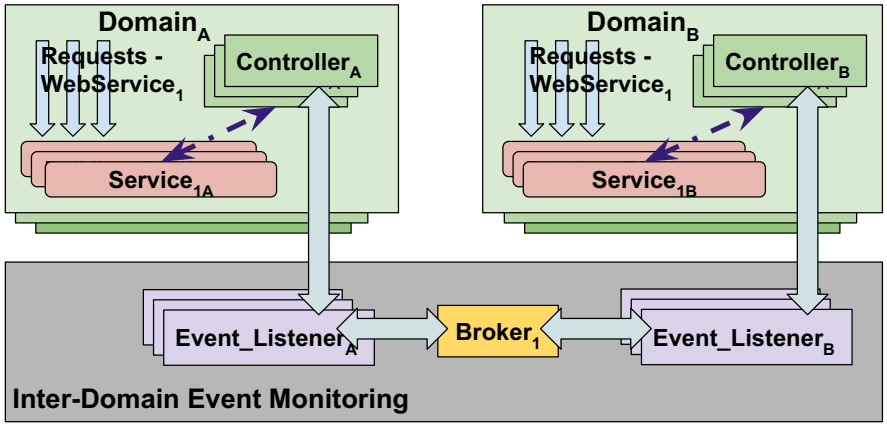

FIGURE 3 *Mayan-DS* controller instances in a federated deployment architecture

The core of *Mayan-DS* is constituted by the communication between inter-domain SDN controllers, facilitated by MOM protocols such as Advanced Message Queuing Protocol (AMQP)[42] and MQTT (Message Queuing Telemetry Transport).[43] The service requests are mapped to the network through SDN, and the resource provisioning is managed with the assistance of the SDN controller. Hence, each domain is aware of the services that are served by its services. Congestion and failure of the underlying computing nodes and links can be monitored by leveraging SDN, such that malfunctioning nodes can be dynamically removed or demoted among the alternatives to fine tune the services placement and support dynamic execution migrations.

Controller$_A$ controls the *Domain*$_A$, while *Controller*$_B$ controls the *Domain*$_B$. Service instances such as *Service*$_{1A}$ and *Service*$_{1B}$ are multiple deployments of the same service, *Service*$_1$. The *Service*$_{1A}$ instances satisfy the web service requests to the *Service*$_1$ in the *Domain*$_A$. Similarly, the *Service*$_{1B}$ instances satisfy the respective requests in the *Domain*$_B$. An entire workflow execution sequence is carried out in a single original composition unless any given threshold (such as the load on any of the service instance) is met. When a threshold is met, an event is triggered, and the controller is notified. If there is no service instance to satisfy the request in the current domain, *Mayan-DS* sends the request for service provisioning to an alternative deployment from another service domain. The other service deployment receives this request and continues to execute the remaining of the workflow, without disrupting the workflow execution.

3 | *MAYAN-DS* ALGORITHMS

The data storage in *Mayan-DS* is performed at the initial stage, with later continuous updates. *Mayan-DS* defines all of its executions, including the data storage task, as data services. *Mayan-DS* consists of two core algorithms - one to initialize the framework, and the other to schedule the execution of data services of each workflow.

3.1 | Initializing the *Mayan-DS* framework

Algorithm 1 presents the initialization procedure of the *Mayan-DS* framework in each domain. First, as shown in line 2, *initController()* initializes the SDN controller with bootstrapping information consisting of the endpoints of the MOM brokers. The MOM brokers can be local brokers managed by the domain of the controller, or remote brokers managed by other domains that the controller has subscribed to. The bootstrapping information enables the controller to communicate with the brokers for the inter-domain communications regarding the wide area network status as well as the migrations including the data storage and data service executions. Then the flag *controller.initialized* is set. If the domain manages any broker instances, *broker.initialize()* initializes those instances next if they have not been initialized already by a prior initialization procedure (lines 4 and 5). Then, the flag *broker.initialized* is set (line 6).

Once the controller and the brokers are initialized, *init(server)* (line 9) initializes each server (precisely a subset of servers, the servers in a dedicated cluster in the data center, rather than all the servers of the data center, since *Mayan-DS* does not necessarily span the entire data center) and its respective persistent storage (such as SQL and NoSQL databases) in the data center. Then, *initImdgCluster()* (line 12) initializes all the IMDG clusters. Each IMDG cluster spans the entire execution nodes as a virtual in-memory cluster. An instance that initializes a cluster becomes the master instance of the cluster. Since we can have several IMDG clusters for each IMDG (such as Hazelcast[44] and Infinispan[45]), the procedure initializes at least one cluster per IMDG. Once the IMDG clusters are initialized, *initServiceEndpoints()* (line 14) initializes the service endpoints to receive the client requests for the data services and APIs from the clients. Then, *startIMDG()* (line 17) starts the IMDGs in each of the servers. Once an IMDG instance is launched, *joinImdgCluster(id, server)* joins the instance to the respective IMDG cluster (line 18). Finally, *initServices()* (line 20) starts scheduling the initial set of services, indicated in the bootstrap information, in each server.

Algorithm 1 Initialize the *Mayan-DS* Framework

```
1:  procedure InitSDDS()
2:      controller ← initController(brokers)                    ▷ Initialize the controller with the bootstrap information
3:      controller.initialized ← TRUE
4:      if (∃ broker ∈ controller.brokers; (broker.isLocal() ∧ ¬broker.initialized)) then   ▷ Presence of uninitialized local brokers
5:          broker.initialize()
6:          broker.initialized ← TRUE
7:      end if
8:      for all (server ∈ cluster) do
9:          init(server)
10:     end for
11:     for all (imdg ∈ IMDGs) do
12:         imdgIDs ← initImdgCluster(imdg)
13:     end for
14:     initServiceEndpoints()
15:     for all (server ∈ cluster) do
16:         for all (id ∈ imdgIDs) do
17:             startIMDG(id, server)
18:             joinImdgCluster(id, server)
19:         end for
20:         initServices(server)
21:     end for
22:     while (controller.initialized) do                       ▷ Periodic update to the SDDS environment of the domain
23:         if (broker.getEvent(t) ≠ ∅) then                    ▷ A non-null event received at current time t
24:             ℰ(t) ← broker.getEvent(t)                       ▷ Set the value of ℰ(t) from the event
25:             if (ℰ(t) = SIGINT) then                         ▷ Interrupt Signal Received
26:                 controller.initialized ← ¬controller.initialized   ▷ Negate the controller.initialized flag
27:             else
28:                 update(∇ℰ(t))   ▷ An event composed of updates to the controller, as well as the broker and service instances
29:             end if
30:         end if
31:     end while
32: end procedure
```

Once *Mayan-DS* is initialized for the domain (typically a data center), the *Mayan-DS* controller continues to monitor for events from the broker (line 22). If a non-null event is received at time t (line 23), it is stored asynchronously in an event queue of $\mathcal{E}$, with the timestamp t (line 24). If the event is an interrupt signal *SIGINT* (line 25), the flag *isControllerInitialized* is inverted (line 26). Otherwise, the update function is invoked to update the controller, broker, and service instances appropriately via *update*($\nabla\mathcal{E}(t)$) (line 28). Equation (8) defines the change propagated through the event $\mathcal{E}(t)$ as $\nabla\mathcal{E}(t)$. $\nabla\mathcal{E}(t)$ is a combination of changes in controller (c), local broker instances ($b \in B_c$), as well as the service instances ($s \in S_c$) managed by the controller c

$$\nabla\mathcal{E}(t) = \frac{\partial\mathcal{E}(t)}{\partial c} + \frac{\partial\mathcal{E}(t}{\partial b}\Big|\forall b \in B_c + \frac{\partial\mathcal{E}(t)}{\partial s}\Big|\forall s \in S_c. \tag{8}$$

Thus, the update procedure keeps the environment updated in each domain until the controller is terminated. As the controller termination itself is defined as an event, the event of a controller (and consequently, the domain controlled by the controller) leaving the network is immediately propagated to the other controllers as a MOM event, before the controller terminates.

3.2 | Scheduling data services

The data services comprise various data actions, such as data storing, data deduplication, data aggregation, data analysis, and data manipulation. Some of these data services can be composite, with a series of data services chained to each other as a service composition. Algorithm 2 presents the overall scheduling of a data service *s* on a cluster (or a data center) following the SDDS approach. First, an empty set named *potentialServers* is created to track of the potential servers for the given service scheduling (line 2). Then, while the controller is in the initialized state (as illustrated in the Algorithm 1), the scheduling process continues to wait for the service invocations (line 3).

Algorithm 2 Schedule Data Service

```
1:  procedure SCHEDULE(CLUSTER, S)
2:      potentialServers ← ∅
3:      while (controller.initialized) do
4:          if ((potentialServers.getBest() = ∅) ∨ (update(s) ≠ ∅)) then        ▷ ∂ℰ(t)/∂s ≠ 0
5:              potentialServers ← cluster.getServer(s)                          ▷ minimize(A(n, D))
6:              if (potentialServers = ∅) then
7:                  broker.update(s, status)          ▷ Service status as a MOM message to the associated brokers
8:                  serviceEndpoint ← broker.get(s).endpoint   ▷ Receive potential service instance to migrate the workflow
9:                  s.migrate(serviceEndpoint)
10:                 return
11:             end if
12:             s.schedule(potentialServers.getBest()))   ▷ Choose the first in the ordered set of best servers in the cluster
13:         end if
14:     end while
15: end procedure
```

Network congestion or resource scarcity in the current execution node can interrupt the execution in the form of control flows. As illustrated by Equation (8), *update(s)* will trigger an update to the scheduled service executions, if $\frac{\partial \mathcal{E}(t)}{\partial s} \neq 0$. If the set *potentialServers* does not have the best (typically the first one, since the set of *potentialServers* is often stored as an ordered list) element or if an update is triggered for the given service (line 4), the *potentialServers* set is initialized with the servers that host the majority of the data (line 5). The algorithm leverages the constructs offered by the underlying distributed execution framework to identify the servers to host data to minimize the communication overhead of migrating data back and forth inside the data center or cluster, as illustrated by Equation (1). The algorithm avoids dispersing data and services belonging to the same workflow across several domains by first trying to get the potential servers inside the current cluster or data center (line 5).

After initialization, if the set remains an empty set, the algorithm identifies that the requirements for the service execution are not met by the servers in the current domain (ie, the cluster or the data center) (line 6). When a big data workflow execution requires execution of services spanning several data centers, *Mayan-DS* aims to find the service instances in a network-aware manner, finding the best execution paths from the set of potential execution paths identified in Equation (6). Thus, if the current domain does not have a suitable data service instance available, the controller updates the brokers that it has subscribed to, with this status for the data service (line 7). Then, the controller receives a potential service endpoint for the service invocation (line 8). Once an alternative service endpoint from another network domain is identified, the data service invocation is migrated to the newly identified service instance, and the execution of the data service in the current domain terminates (line 9). Finally, *Mayan-DS* schedules the data service in the chosen server(s) using *s.schedule()* (line 12), until its completion. The chosen service instances continue to receive the client requests until another interrupt is received.

4 | PROTOTYPE IMPLEMENTATION

We prototyped *Mayan-DS* as an SDDS framework to assess the feasibility of the proposed architecture with Apache Axis2 1.7.0 and Apache CXF 3.2.1 web services engines. Using these web service engines, we created the data service APIs on top of Hazelcast 3.9.2 and Infinispan 9.1.5 IMDGs. The underlying persistent storage was composed of MySQL server and MongoDB. OpenDaylight Beryllium was leveraged as the SDN controller. We implemented and extended Messaging4Transport, an AMQP northbound for inter-domain control flows, as the core of the *Mayan-DS* architecture.[‡]

We used ActiveMQ 5.15.3 as the default broker. We deployed the *Mayan-DS* framework on a cluster with Infinispan IMDG initialized on them. Due to the limitations of access to data centers and multi-domain network environments, we resorted to emulations and simulations in evaluating larger execution environments. To model a larger execution environment, we deployed *Mayan-DS* over a data center of leaf-spine topology emulated with Mininet network emulator.[46] We simulated multi-domain wide area networks on the cluster and executed big data workflows across them. We developed the *Mayan-DS* framework following a layered architecture with a bottom-up approach, consisting of a data plane, storage plane, control plane, and execution plane, as illustrated by Figure 4.

The data plane consists of the switches and servers. The storage plane includes the various SQL and NoSQL databases. The control plane comprises deployment of the OpenDaylight controller and IMDG clusters. The OpenDaylight controller and the IMDG clusters control the network data plane devices and data placement accordingly. An AMQP implementation supports the inter-domain workflows by offering them as

[‡]https://github.com/pradeeban/mayan-ds

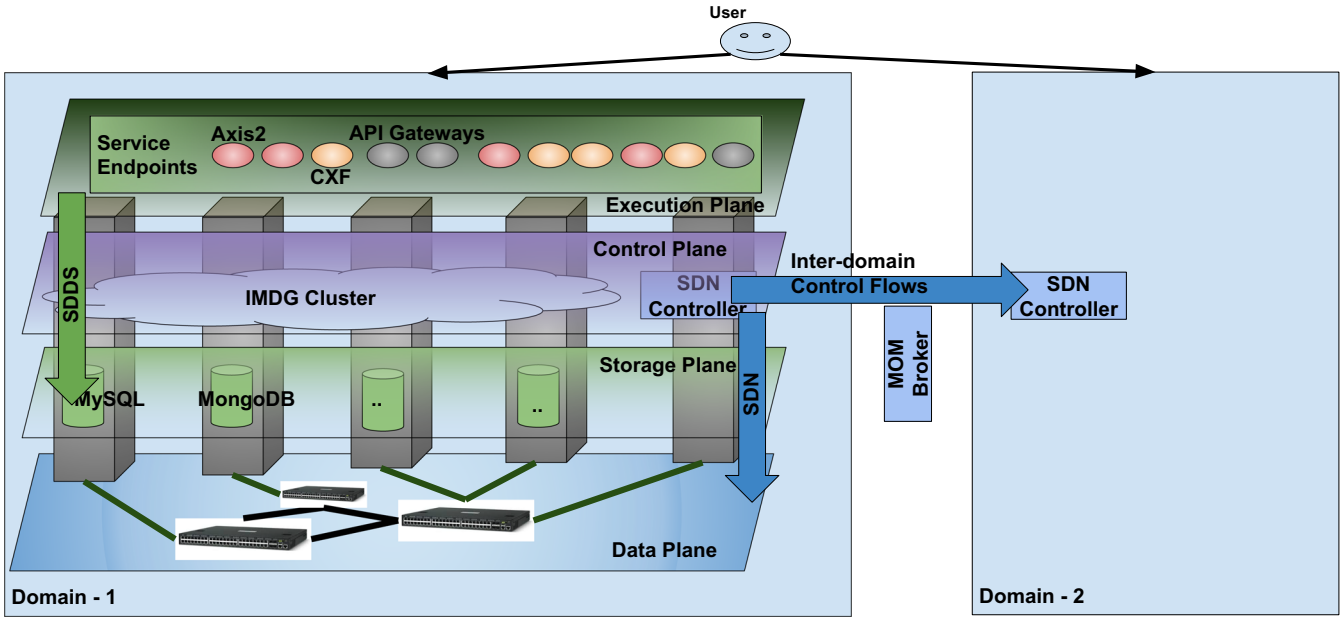

FIGURE 4 A three-dimensional view of the *Mayan-DS* implementation

subscriptions between the controllers. Execution plane consists of the service APIs of web service engines and API gateways such as Kong, Tyk, and API Umbrella. The API gateways host the API endpoints of the multi-domain deployments to serve the user requests seamlessly. The controller consists of the data service scheduling algorithm, whereas the initialization algorithm serves as the core of the framework across all the planes.

The control plane enables inter-plane communications. The SDN controller facilitates communication between the execution plane and data plane and functions as the core enabler of the cross-layer communication. IMDGs facilitate interactions across the execution plane and the storage plane. The separation of control from the data and storage planes facilitates cross-control communications. On the other hand, supporting inter-domain communications through MOM messages enables a dynamic multi-domain network, without requiring a static topology. Thus, *Mayan-DS* offers a platform for scalable and interoperable data service workflows.

5 | PROTOTYPE EVALUATION

Our evaluation seeks to answer thee following two questions: (1) under what conditions we can further improve the performance of an optimized data execution framework with an SDDS approach in a data center or a cluster? and (2) can an SDDS approach significantly enhance the big data workflow performance in multi-domain wide area networks at Internet scale? We thus evaluate the scalability of the SDDS approach from data centers to the Internet. We assessed the efficiency of the SDDS approach in improving the big data workflow performance, regarding throughput and end-to-end latency.

Our evaluation environment consists of a cluster of 6 servers, each with an AMD A10-8700P Radeon R6 processor, 10 Compute Cores 4C+6G×4, 8 GB of memory, Ubuntu 16.04 LTS 64 bit operating system, and 1 TB of disk space. We built a base framework with Infinispan IMDG for a distributed in-memory execution of data services and a MySQL and MongoDB-based persistent storage. We optimized the IMDG cluster using the Infinispan's programming constructs for efficient locality-aware data storage. We then extended this base framework to build a *Mayan-DS* prototype for SDDS execution with cross-layer data management and inter-domain coordination.

5.1 | Software-defined data service execution in a cluster

We first evaluated the performance enhancements of the SDDS approach inside a cluster. We composed a big data workflow with data storage and update services, first with the base framework, and then with *Mayan-DS*. While the workflow is data intensive, it is not high in its processing demands. We measured the throughput from the service plane by the total amount of data processed through the data services per unit time, for both the base as well as *Mayan-DS*, with an increasing volume of data to be processed.

We deployed the *Mayan-DS* prototype for SDDS execution as well as the base implementation with a network-agnostic data service execution inside the Infinispan IMDG cluster. The base implementation ensures that the related objects stay together in the same node, as much as possible, using a user-defined key as an index. However, the base approach engages all the instances into the execution, oblivious of the actual network load. *Mayan-DS* monitors the system workload and adaptively scales the deployment by extending the base approach. Figure 5 benchmarks the performance of the *Mayan-DS* against that of the base approach, with the growing volume of the data.

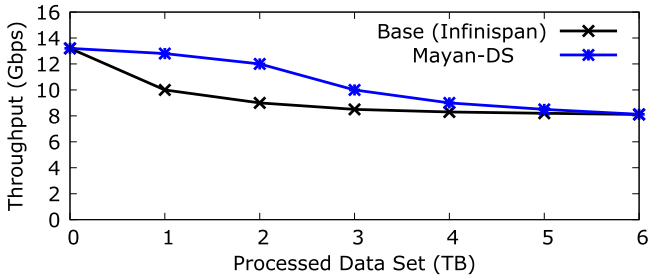

FIGURE 5 Performance comparison of *Mayan-DS* against the base

The base approach performed and scaled well itself on clustering the related data together based on the keys (the indices that Infinispan uses to group the related data objects internally). However, due to the data-heavy nature of the services, incorporating additional instances early on for the distributed execution indeed harmed the performance of the execution, by making the IMDG to naively distribute data among the instances, even though not uniformly. We note that a computation-heavy service workflow would indeed favor a scaled execution early on, as the performance benefits of the distributed execution would typically be more prominent than the performance degradation cost by the communication overhead. Therefore, we note that the awareness of the network and the executed workflows offer smart scaling capabilities to *Mayan-DS* based on the nature of the workload.

Mayan-DS aims to minimize the data distribution unnecessarily and prematurely, inside a data center or a cluster. It complements the scaling out of the execution offered by the IMDG to consider the workload in using the available nodes, instead of distributing the data and execution to all the instances. Thus, it effectively reduces the number of servers that are involved in the execution at any given time. Up to 1 TB, *Mayan-DS* leveraged a single server for the storage, as each server consists of 1 TB of free disk space. It distributed data to more servers as the data increased beyond 1 TB. Consequently, *Mayan-DS* harnessed all six servers when we use 6 TB of data, due to the limitation in the persistent storage. When the total data reaches 6 TB, the throughput of *Mayan-DS* converges to that of the base. For the smaller data sizes (as long as the storage space in the servers is not exhausted), *Mayan-DS* offered better performance by avoiding early premature distribution of data in the big data workflows. We further observe that, for negligible data volumes ($\approx$ 0 TB), both the base and *Mayan-DS* approaches offered the same throughput.

We also observed that a brute-force approach distributes data equally among the six servers. Such an equal distribution leads to poor performance due to communication overhead caused by uniformly and evenly scaling out the workloads. Both our base and *Mayan-DS* approaches performed equally better than such naive approaches in data distribution. However, lack of coordination between the execution and storage leads to lower throughput at the application level in the base approach. Our prototype evaluations highlight the data service performance efficiency of *Mayan-DS*. Furthermore, it elaborates that even the platforms finetuned for performance can benefit from the coordination and collaboration facilitated by a software-defined approach. We note that exposing existing IMDG executions or other big data applications as data services is a significant undertaking. However, we believe that the performance benefits and enhanced control associated with such data service execution justify the effort over time.

5.2 | Network-aware data service execution at internet scale

Comparing the performance of big data workflows on a global scale is a challenging task due to the limited access to geo-distributed servers. We simulated and modeled a geo-distributed network with RIPE ATLAS Probes[47] and our physical servers to evaluate the performance enhancements of a network-aware data service workflow execution at Internet scale. Table 1 elaborates a part of the deployment architecture of our modeled environment, with several servers (ie, nodes, forming a connected graph) spanning the globe. The nodes are connected to the Internet, whereas a few selected links are also connected to each other via direct links. The direct links are bidirectional, although we list them as origin and destination pair for the ease of reference. We realistically chose the links that are geographically close to each other in forming a direct connect. For example, we modeled a direct connection between a smaller town such as Svalbard, Norway against the capital city, ie, Oslo, Norway (rather than connecting to a far away region of Singapore, as assuming the existence of such long-haul direct links will hinder realistic evaluations). We modeled the direct dedicated connects between two nodes with a 10 Gbps bandwidth, the maximum bandwidth offered by the AWS Direct Connect to connect the user servers to the cloud servers directly.

To assess the Internet path performance, we compared the Round-Trip Time (RTT) between two endpoints that connect through *Mayan-DS* against that of using the Internet-based connectivity of ISPs. We modeled the RTT (typically measured by a ping) of *Mayan-DS* with realistic values. We used a constant fiber path adjustment of 10% and metro fiber and a local loop length of 100 km. We considered the speed of light in fiber as 200 km/ms. We considered the equipment latency of 1 ms in our evaluation nodes. We thus estimated the latency between two directly connected links on the Internet.

We then benchmarked *Mayan-DS* against the regular Internet routes for network-aware big data workflows. We sent pings between the endpoints, first entirely utilizing the public Internet paths and then through *Mayan-DS*, where *Mayan-DS* chooses the best-path considering the

TABLE 1 The simulated *Mayan-DS* deployment environment (with modeled latency)

Origin node	Nodes connected via a direct link
Svalbard	{Moscow (30.94), Oslo (25.33), … }
Vladivostok	{Tokyo (10.62), … }
Hobart	{Sydney (20.55), … }
Cape Town	{Windhoek (19.29), … }
Mangilao	{San Francisco (44.36), … }
Manila	{Seoul (31.82), … }
⋮	{⋮}

TABLE 2 Ping times (ms) between two nodes, ISP-based versus *Mayan-DS* connectivity

No.	Origin (o) → Destination (d)	ISP (o → d)	*Mayan-DS*	*Mayan-DS* Path
1	Svalbard/Norway → SÃ→ Paulo/Brazil	339.164	246.436	o ⇒ Oslo/Norway → d
2	Vladivostok/Russia → SÃ→ Paulo/Brazil	373.712	282.451	o ⇒ Tokyo/Japan → d
3	Atlanta/USA → Hobart/Tasmania/Australia	255	255	o → d
4	Hobart/Tasmania/Australia → SÃ→ Paulo/Brazil	413.869	340.617	o ⇒ Sydney/Australia → d
5	Atlanta/USA → Svalbard/Norway	164	164	o → d
6	Atlanta/USA → Vladivostok/Russia	287	194.68	o → Tokyo/Japan ⇒ d
7	Cape Town/South Africa → Colombo/Sri Lanka	345.889	342.877	o ⇒ Windhoek/Namibia → d
8	Atlanta/USA → Windhoek/Namibia	301	299.29	o → Cape Town/South Africa ⇒ d
9	Mangilao/Guam → SÃ→ Paulo/Brazil	968.149	252.111	o ⇒ San Francisco/USA → d
10	Singapore → Seoul/South Korea	200.581	75.907	o → Manila/Philippines ⇒ d
⋮	⋮ → ⋮	⋮	⋮	⋮

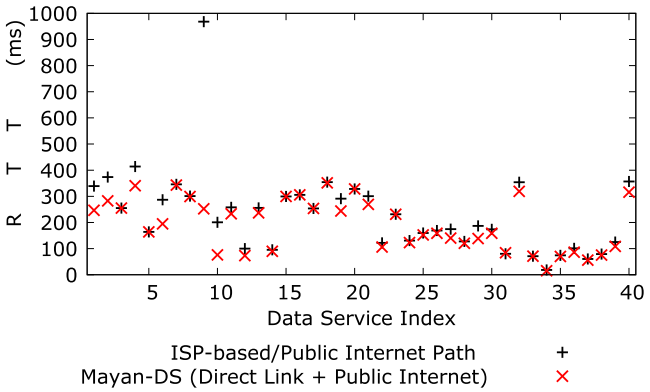

FIGURE 6 Ping times of *Mayan-DS* against the public internet-based connectivity

available direct connections and the Internet paths, in composing the end-to-end connection between the origin and the destination nodes. We list a part of the ping times (in milliseconds) between two endpoints via the Internet-based routes as well as via *Mayan-DS* in Table 2.

The arrow → indicates a public Internet path (ie, ISP-based connectivity) between two nodes, whereas ⇒ indicates a direct link. Therefore, ($o \Rightarrow i \rightarrow d$) indicates that the origin node o and the destination node d are connected via an intermediate node i, where there is a direct connect between o and i whereas the rest of the path (ie, i to d) is a public Internet path. Similarly, ($o \rightarrow i \Rightarrow d$) indicates that i and d are connected by a direct link, while o and i are connected via Internet paths. Figure 6 depicts the ping times of *Mayan-DS* against that of data service scheduling entirely based on the public Internet paths. The plot illustrates the service invocation times (assuming the delay caused by the service execution itself to be negligible), depicting 40 instances of the same data service implementation deployed at several geographical locations and invoked from various geographical locations.

Each data service consists of a data flow between an origin node o and a destination node d, which are geographically distributed. Each such pair, identified by the *Data Service Index*, is unique, as shown by Table 2. Therefore, each point in Figure 6 depicts the invocation of a data service deployed in d from the origin o. o can either be a user or a previous service whose output is sent to the service in d to compose the big data workflow. We present this as a representative measure for a data service invocation.

Mayan-DS exploits the available direct links that connect either the origin or the destination to an intermediate node. By routing via such direct links, *Mayan-DS* minimizes the latency of the remote service endpoints significantly. *Mayan-DS* was able to achieve up to 33% reduction in

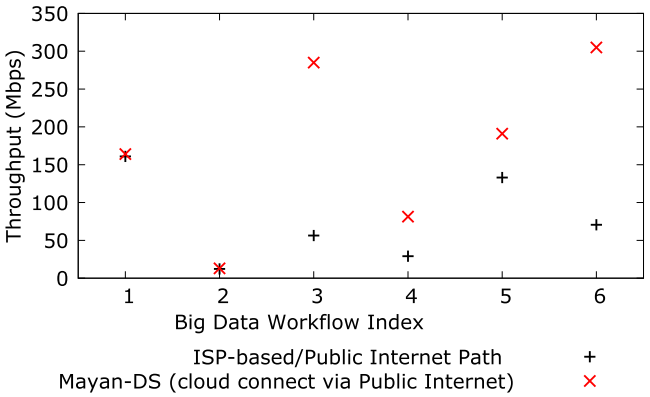

FIGURE 7 Throughput of *Mayan-DS* against the public internet-based connectivity

latency with just a little fraction of the path routed through a high-bandwidth direct link. When the direct link covers a substantial portion of the route, apparently, the latency reduced significantly up to 75% or more. We further observe that the availability of such direct links also minimized the outliers (ie, the pairs of endpoints that consume an abnormally large amount of time for the data transfer) by reducing the dependency on long-haul public Internet-based links from the remote regions to the nearby regional Internet hubs. A big data workflow consists of several such data service invocations chained together. Therefore, the latency improvements of *Mayan-DS* in big data workflows will be even more prominent.

We note the prevalence of such direct connection between a smaller city with a nearby cloud region or a metropolitan city. However, our observations indicate that sometimes smaller towns have faster connectivity than a bigger metropolitan neighbor. While this might be due to the local congestions or other external factors, it is also difficult to predict which of the two cities will benefit more by leveraging direct links between two servers in the cities. Furthermore, a direct connection between a remote island, such as those belonging to Hawaii to a major Internet hub such as San Francisco, reduces the latency of a service invocation from the remote location significantly as we observed. Therefore, we note that the performance benefits entirely depend on how the origin or destination regions differ from the intermediate region that they connect to via the direct connect. We also observed that the strategic positioning of the intermediate node between the origin and destination nodes was a deciding factor. The *Mayan-DS* controller thus exploits its awareness of such direct links for a faster data service execution across the globe.

5.3 | *Mayan-DS* big data workflow scheduling via a cloud server

We then measured the end-to-end throughput of a big data workflow using *Mayan-DS* against that of using the public Internet access provided by the ISPs. We first measured the throughput of data flows between two endpoints o and d directly via the public Internet ($T(o,d)$). Then, we scheduled the data flow between the endpoints via *Mayan-DS* ($T'(o,d)$) that chooses the best route among the options of sending the data either entirely through the public Internet or via an Intermediate cloud server from the cloud region r close to o. With a network-aware selection of an intermediate node (in this case, a cloud server), *Mayan-DS* aims at mitigating congested paths in the data service execution. As *Mayan-DS* chooses the best option among the two (ie, leveraging the public Internet paths without the intervention of *Mayan-DS* versus *Mayan-DS* choosing a cloud server as an intermediate node in routing the data flow), effective network bandwidth achieved by *Mayan-DS* is limited by $max(T(o,d), min(T(o,r), T(r,d)))$.

As the big data workflows are composed of two or more data services, the end-to-end throughput of the workflow is also limited by the throughput of the least-performing data service, $min(T(s) : s \in W)$. We assume that the services ($s \in W$) composing the workflow W have a higher throughput than the throughput of the network path (public Internet path, as well as the path modified with *Mayan-DS*), as shown by Equation (9)

$$min(T(s) : s \in W) \gg max(T(o,d), min(T(o,r), T(r,d))). \tag{9}$$

Therefore, the service endpoints themselves do not pose a bottleneck to the performance of the workflow. Thus, the *Mayan-DS* throughput can be simplified to $max(T(o,d), min(T(o,r), T(r,d)))$, the maximum bandwidth achieved by the *Mayan-DS* network. However, if there are data services that offer a throughput lower than the bandwidth of the network links, the service throughputs will also be a deciding factor on choosing the data service instances in the big data workflow execution.

We composed a typical big data workflow consisting of a service composition that chains two data services, ie, one service performing a simple check and the other one indexing its input data and outputs it into persistent storage asynchronously. Figure 7 illustrates the throughput of the same workflow sequence performed at different deployment locations, achieved with the *Mayan-DS* deployment as well as entirely with the public Internet paths. In the plot, the workflow index identifies a workflow sequence in a specific route, exploiting the same implementation of the data service chains in different deployments in geo-distributed endpoints.

We observe that *Mayan-DS* exploits the cloud paths and the public Internet paths efficiently to find the best route for the data service execution. By replacing the public Internet path between the slower Internet region and the close by region with a faster connection with an

existing direct connect or a cloud path, *Mayan-DS* avoids the bottleneck in the data service workflows. Furthermore, when exploiting such direct links to an intermediary node was indeed not favorable in latency for any big data workflow, *Mayan-DS* sends the data entirely through the public Internet. Therefore, *Mayan-DS* always performs either better than or equal to using only the public Internet paths for the big data workflows. Thus, it offers a higher throughput for the data service execution, compared to a network-agnostic workflow scheduling over the Internet.

Our evaluations highlight that an SDDS approach always offers equal or better performance in terms of latency and throughput compared to a network-agnostic execution. Our observations confirmed the efficiency of SDDS in scaling from clusters and data centers to multi-domain wide area networks such as the Internet.

6 | RELATED WORK

In this section, we look into the related work to our SDDS framework. We specifically discuss the resource sharing for data service workflows, network-aware service composition workflows including data-intensive as well as compute-intensive workflows, and enterprise SDDS offerings and related Software-Defined Systems proposed by the research.

Exploiting services to offer unified access interfaces to big data has been proposed in previous research. Remarkably, OGSA-DAI (Open Grid Services Architecture - Data Access and Integration)[48] facilitates federation and management of various data sources through its web service interface. While the research on resource sharing, volunteer computing,[9] and Service Oriented Architecture (SOA)[49] researched their peak a decade ago, wide-spread adoption of SDN and research on expanding SDN for wide area networks are recent. Therefore, despite their promising outcomes regarding interoperability, the research efforts on data services have not been extended to the Internet scale in a network-aware manner. *Mayan-DS* exploits the recent advancements in network softwarization such as SDN in enabling interoperable and network-aware data service workflows in wide area networks.

6.1 | Network-aware resource allocation for data service workflows

Previous research by Rupprecht[50] categorized big data traffic in data centers into aggregatable traffic, non-aggregatable traffic, and storage traffic. Aggregatable traffic represents the data traffic where the receivers consist of a service that performs an aggregation function such as sum, join, sort, or average. On the other hand, non-aggregatable traffic denotes the traffic that cannot be aggregated. Storage traffic represents the network flows where data is written to or read from the storage devices. Researchers propose solutions such as middlebox-based aggregation, efficient computation of aggregation, and allowing queries to reroute the data flows without impacting the existing queries to support network-aware big data processing.[50] However, such a distinction of services become vague in workflows that consume and chain several data services. Instead of differentiating the big data flows into categories, *Mayan-DS* develops them all as web services, thus inherently supporting interoperable execution and chaining of the services, regardless of their nature.

Fine-tuning the data placement in the distributed data stores inside and beyond data centers have attracted a considerable research interest.[51] DIANA offers network-aware scheduling for data-intensive workloads in grid environments.[52] DIANA is more efficient than pulling data to the execution node in computation-intensive workloads or pushing the execution to the data in data-intensive workloads in a network-agnostic manner. However, unlike the grid and cloud environments, inter-cloud and edge environments are managed by multiple vendors. Big data workflow execution across services spanning such multi-vendor multi-domain environments need to consider additional parameters such as the possible interconnect between the network domains and the high-latency network links (typically, the public Internet routes) that connect the domains. *Mayan-DS* takes into account both the data service composition executions inside a single center as well as the executions that span several domains by representing them in a unified notion. While previous work has proposed data-aware and network-aware workflow scheduling in data centers and clouds,[53] they mostly narrow their focus to a given domain managed by a single vendor. *Mayan-DS* aims at the efficient scheduling of big data workflows in multi-domain wide area networks consisting of several federated clouds and data centers.

NetMIP considers network properties and resource consumptions in the cloud network nodes for service compositions, rather than merely addressing the Quality of Service (QoS) of the services as a static property.[54] It presents optimization problems to maximize the QoS utility value while minimizing network resource consumption. NetMIP limits its focus to data centers and does not cater for data-intensive workloads in wide area networks. Research has proposed network-aware service compositions in wide area networks such as cloud environments.[55] Vivaldi[56] leverages network coordinates in estimating latency between two endpoints. While Vivaldi and similar approaches have been used to predict and model latency in service composition workflows on the Internet, their use in wide area networks consisting of several domains is mostly limited. Models based on game theory have been proposed to understand dynamic service placements in cloud environments comprised of multiple service providers.[57] Similarly, significant research efforts have been made to identify an optimal service placement in wide area networks that are shared by several users, considering the dynamic availability of the resources.[58] Mayan[59] proposes Software-Defined Service Compositions in the multi-domain networks by extending and exploiting SDN for context-aware workflow execution. However, Mayan focuses on web services that are typically computation-intensive.

A computation-intensive workflow is typically smaller in duration and volume. Therefore, a quick decision is essential for minimal overhead, causing an additional constraint on service instance selection in multi-domain wide area networks. On the other hand, data service workflows have

substantial demand for data volume and data rate, and are typically long-running. Furthermore, overheads caused by the control flows in big data workflows are negligible, as these workflows are composed of elephant flows of data services. Therefore, it is feasible to have a more involved and time-consuming service instance selection procedure in the controller, in favor of optimal data and execution placement, for big data workflows.

6.2 | Software-defined systems for big data

Software Defined Internet of Things (SDIoT)[60] proposes a software-defined architecture for IoT devices by handling the security,[61] storage,[62] and network aspects in a software-defined approach. Software-Defined Systems such as SDIoT can indeed be extended as an SDDS framework. However, existing SDN-based approaches such as SDIoT focus on networks and systems, rather than the data itself. Big data applications require prominence to data for a high-performant and latency-aware execution in the wide area networks. Therefore, it is essential to develop SDDS with data placement and execution as the primary focus. *Mayan-DS* leverages and exploits the existing constructs in place in frameworks such as IMDGs and web service engines to facilitate this complex undertaking.

Many enterprises such as SAP[63] and Oracle[64] provide data services as part of their middleware frameworks. Several enterprises propose SDDS by bringing a software-defined approach to data services. Portworx is an open source SDDS framework that follows a container-based approach.[65] However, Portworx mostly limits its focus to dev-ops by exploiting Kubernetes, Mesos, and Docker swarms. PureStorage offers various data services, with a software-defined approach.[66] These data services include (1) a *reduce* service, (2) an *assure* service with high availability, QoS guarantees, and encryption, and (3) a *protect* service with snapshots and replication. PureStorage covers several aspects of data storage through numerous data services. However, it does not offer a generic software-defined framework for big data. IBM Spectrum Copy Data Management[67] and Catalogic ECX[68] are Copy Data Management (CDM) frameworks that aim to modernize the IT infrastructure with a Software-Defined Copy Data Management, an instance of SDDS. They centrally orchestrate how data is copied and shared among the users of various data sources.

Many SDDS enterprise offerings limit their focus to storage, or a Software-Defined Storage, rather than focusing on service execution. RedHat defines SDDS as a storage orchestration enabled by full lifecycle management consisting of provisioning, installing, configuring, tuning, and monitoring the data.[69] PrimaryIO APA is a storage solution with a higher performance achieved through virtualization, intelligent caching, and an SDDS approach.[70,71] HPE proposes SDDS, available in storage and servers, as well as embedded in systems and software. HPE SDDS aims to offer scalable, secure, and highly available virtualized storage solution.[72] Commvault mitigates vendor lock-in through its service-based data access and Software-Defined Storage.[73] Commvault proposes an SDDS approach for backup, archival, and recovery, with performance, scalability, and agility.[74] CTI offers fast incremental updates, per-application storage, and efficient computation agnostic to the physical location of the resource, through its SDDS approach.[75] Commvault and CTI aim at offering performance regardless of the physical location of the data in the data center.

Due to the unified control offered by the SDN controllers, data centers can well provision the resources for their workflow execution. The recent surge in the enterprise offerings of SDDS frameworks highlights the demand for an SDN-based or an SDN-like approach for big data. However, the current enterprise SDDS offerings (1) limit their focus to certain data services rather than providing a generic data services framework, (2) focus entirely on a single-domain network such as their cloud offerings or enterprise data centers, (3) lack interoperability between execution frameworks due to their closed vendor-specific implementation, or (4) do not leverage the existing research efforts on SDN and SD-WAN for network-aware service workflows. These limitations highlight the need for a complete network-aware and interoperable SDDS framework. *Mayan-DS* is the first to propose a generic SDDS framework, with a fine-grain control inside data centers, as well as coordination between domains, for data service workflows.

7 | DISCUSSION

In this paper, we proposed SDDS as a unified approach to data storage and processing, by separating the data, storage, control, and execution planes. While we limit our implementation effort to specific frameworks (such as Infinispan and MongoDB) as a prototype, in practice, researching and implementing such an interoperable approach as a global framework is a challenging task. We looked into the way of addressing the research challenges through our proposed *Mayan-DS* architecture and its service-based approach.

Mayan-DS achieves the challenge of storing related data closer to each other through its network-awareness achieved by SDN extended with a MOM deployment. However, storing of data inside a cluster ensuring that related objects stay closer is specific to the storage plane. This task is achieved with the support of the human user in the data sources, eg, with careful indexing, to group the related objects together. IMDGs and database frameworks already offer such efficient storage for quick query and access capabilities. *Mayan-DS* proposes to exploit such existing constructs in ensuring minimal data movements inside a data center or a cluster. On the other hand, *Mayan-DS* proposes to utilize inter-domain control flows in identifying the best path for workflow execution.

SDDS aims to exploit the data awareness offered by the big data applications together with the network awareness of an extended SDN hierarchy in the wide area network. However, the realization of such networks for heterogeneous data sources is a complex problem due to several implementation challenges. Deploying an in-memory cluster on top of a persistent storage plane is a relatively trivial engineering task. However, porting existing big data workflows to use an SDDS approach involves writing the respective services to execute on top of the IMDG cluster. An increasingly complex workflow would require more and more data services to be developed and chained. The practicality of

disseminating the service instances themselves across the nodes requires automation, as expecting to configure layers of platforms and software on top of the servers is questionable. One potential solution is to distribute the platform stack itself container instances, supported through frameworks such as Docker.[75] However, a detailed discussion of such possible implementation alternatives is a future work.

Expanding the scope of the data service execution can be seamless once such a deployment is established across several servers, thanks to the recent advancement of network softwarization research. In this paper, we proposed a federated SDN deployment extended with MOM as a SOA to enable network-aware big data workflows at Internet scale. However, the practicality and success of these approaches heavily depend on the adoption of the proposed framework by several infrastructure providers (including data center providers as well as independent nodes as in the case of volunteer computing). While we limited our evaluations to our globally distributed servers, AWS cloud instances, and RIPE Atlas Probes, a latency-aware execution requires several millions of nodes in each region to offer redundancy in execution paths with multiple alternative routes between two endpoints. We introduced the concept of SDDS at Internet scale with the fair assumptions on community adaptation of the SDDS framework. However, the realization of a complete interoperable SDDS framework and overcoming its operational challenges are left as future work.

8 | CONCLUSION

The high demand as well as the focus on the data storage and processing are the core aspect of big data workloads. Big data processing requires resources beyond what can be offered by a single server or even a data center, due to the volume, variety, and geo-distribution of the data. Big data frameworks including IMDGs and NoSQL data sources facilitate distributed execution of big data workflows. Such a distributed execution is usually heavy in its bandwidth demand. Moreover, a big data application procedure is generally confined to its own framework as different frameworks consist of diverse storage and incompatible interfaces to process the workload, thus preventing interoperability. The big data frameworks are designed to work in an environment such as a cluster, data center, or a cloud, which offer consistent network connectivity and topology. Due to these factors, chaining the execution of a big data application between several servers spanning multi-domain data centers and edge nodes in a wide area network is a significant undertaking.

Data services aim at offering interoperability in a wide area network by exploiting the standardization of web services for big data access and processing. Nevertheless, the proliferation of data services has caused a management challenge in placing them and scaling them. Complex eScience workflows aggregate and chain various geo-distributed web services, microservices, and data services for their execution. Context-aware execution of data services in multi-domain networks is essential for incorporating more data services into the scientific and enterprise workflows.

In this paper, we have proposed SDDS, an integration of SDN into the composition of data service workflows. As an SDDS framework, *Mayan-DS* extends data services with the management and resource allocation capabilities offered by the network-awareness of SDN. By leveraging the global awareness of the network topology and network flow statistics of an SDN controller, *Mayan-DS* scales and distributes the data services in a network-aware manner. It minimizes the overhead caused by frequent inter-node communications through an adaptive and context-aware execution supported by a federated deployment of SDN controllers in a wide area network. Thus, *Mayan-DS* reduces the bandwidth overhead common in distributed big data processing. Our evaluations on the *Mayan-DS* prototype indicate how an SDDS approach could be leveraged as a reusable, scalable, and resilient distributed execution framework for the big data workflows on a global scale. As a future work, we propose to implement SDDS as a complete interoperable and network-aware execution platform for various real-world big data applications.

ACKNOWLEDGMENTS

We are thankful to Prof Etienne Riviere for the cloud resources that partly supported our experiments. This work was supported by national funds through Fundação para a Ciência e a Tecnologia with reference UID/CEC/50021/2013 and a PhD grant offered by the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) under grant agreement 2012-0030.

ORCID

Pradeeban Kathiravelu https://orcid.org/0000-0002-0335-0458

REFERENCES

1. Chen CP, Zhang CY. Data-intensive applications, challenges, techniques and technologies: a survey on big data. *Information Sciences*. 2014;275:314-347.
2. Sagiroglu S, Sinanc D. Big data: a review. Paper presented at: 2013 International Conference on Collaboration Technologies and Systems (CTS); 2013; San Diego, CA.
3. Gaur N, Bhogal KS, Johnson CD, Kaplinger TE, Berg DC, inventors: International Business Machines Corp, assignee. System and method of optimization of in-memory data grid placement. US Patent 9,405,589. August 2, 2016.
4. Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation USENIX Association; 2012; San Jose, CA.
5. Greenberg A, Hamilton J, Maltz DA, Patel P. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Comput Commun Rev*. 2008;39(1):68-73.

6. Meng X, Pappas V, Zhang L. Improving the scalability of data center networks with traffic-aware virtual machine placement. In: Proceedings of the IEEE INFOCOM; 2010; San Diego, CA.
7. Williams JW, Aggour KS, Interrante J, McHugh J, Pool E. Bridging high velocity and high volume industrial big data through distributed in-memory storage & analytics. Paper presented at: 2014 IEEE International Conference on Big Data (Big Data); 2014; Washington, DC.
8. Wang L, Tao J, Ranjan R, et al. G-Hadoop: MapReduce across distributed data centers for data-intensive computing. *Future Gener Comput Syst*. 2013;29(3):739-750.
9. Anderson DP, Fedak G. The computational and storage potential of volunteer computing. Paper presented at: Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID); 2006; Singapore.
10. Alonso-Monsalve S, García-Carballeira F, Calderón A. A new volunteer computing model for data-intensive applications. *Concurrency Computat Pract Exper*. 2017;29(24):e4198.
11. Giovannozzi M, Skands P, Zacharov I, et al. LHC@ home: a volunteer computing system for massive numerical simulations of beam dynamics and high energy physics events. Paper presented at: 3rd International Particle Accelerator Conference (IPAC); 2012; New Orleans, LA.
12. Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the Condor experience. *Concurrency Computat Pract Exper*. 2005;17(2-4):323-356.
13. Milanovic N, Malek M. Current solutions for web service composition. *IEEE Internet Comput*. 2004;8(6):51-59.
14. Yue K, Wang XL, Zhou AY. Underlying techniques for web services: a survey. *J Softw*. 2004;3:13.
15. Villamizar M, Garces O, Ochoa L, et al. Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. Paper presented at: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid); 2016; Cartagena, Colombia.
16. Truong HL, Dustdar S. A survey on context-aware web service systems. *Int J Web Inf Syst*. 2009;5(1):5-31.
17. Hadley MJ, Sun Microsystems Inc. Web Application Description Language (WADL). 2006.
18. Christensen E, Curbera F, Meredith G, Weerawarana S. Web Services Description Language (WSDL) 1.1. 2001.
19. Pautasso C, Zimmermann O, Leymann F. Restful web services vs. big'web services: making the right architectural decision. In: Proceedings of the 17th International Conference on World Wide Web; 2008; Beijing, China.
20. Box D, Ehnebuske D, Kakivaya G, et al. Simple Object Access Protocol (SOAP) 1.1. 2000.
21. Treiber M, Dustdar S. Active web service registries. *IEEE Internet Computing*. 2007;11(5):66-71.
22. Zhang Q, Zhang S, Ding Z, Zong Y, Gu N, Liu J. Service registration and discovery in a domain-oriented UDDI registry. Paper presented at: The Fifth International Conference on Computer and Information Technology (CIT); 2005; Shanghai, China.
23. Dogac A, Tambag Y, Pembecioglu P, et al. An ebXML infrastructure implementation through UDDI registries and RosettaNet PIPs. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data; 2002; Madison, WI.
24. The Apache Software Foundation. Apache jUDDI. 2018. https://juddi.apache.org/
25. McKeown N. Software-defined networking. *INFOCOM keynote talk*. 2009;17(2):30-32.
26. Phemius K, Bouet M, Leguay J. Disco: distributed multi-domain SDN controllers. Paper presented at: 2014 IEEE Network Operations and Management Symposium (NOMS); 2014; Krakow, Poland.
27. Cyrus. May 9th: Ibm announces version 2.2.6 of IBM spectrum copy data management. IBM. 2017. https://spectrumcdmsite.wordpress.com/2017/05/12/may-9th-ibm-announces-version-2-2-6-of-ibm-spectrum-copy-data-management/
28. Nexion. Nexion networks. 2018. https://www.nexionnetworks.com/cloud-solutions/
29. Catalogic. What's new in Catalogic® ECX™ 2.6. 2017. https://catalogicsoftware.com/assets/uploads/resources/Catalogic-Whats-New-in-ECX-26.pdf
30. Paganelli F, Ulema M, Martini B. Context-aware service composition and delivery in NGSONs over SDN. *IEEE Commun Mag*. 2014;52(8):97-105.
31. Jain S, Kumar A, Mandal S, et al. B4: experience with a globally-deployed software defined WAN. *ACM SIGCOMM Comput Commun Rev*. 2013;43(4):3-14.
32. Davie BS, Rekhter Y. *MPLS: technology and applications*. Burlington, MA: Morgan Kaufmann Publishers Inc; 2000.
33. Kathiravelu P, Chiesa M, Marcos P, Canini M, Veiga L. Moving bits with a fleet of shared virtual routers. Paper presented at: IFIP Networking; 2018; Zürich, Switzerland.
34. Kumar VP, Lakshman TV, Stiliadis D. Beyond best effort: router architectures for the differentiated services of tomorrow's internet. *IEEE Commun Mag*. 1998;36(5):152-164.
35. Ager B, Chatzis N, Feldmann A, Sarrar N, Uhlig S, Willinger W. Anatomy of a large European IXP. *ACM SIGCOMM Comput Commun Rev*. 2012;42(4):163-174.
36. Afolabi I, Taleb T, Samdanis K, Ksentini A, Flinck H. Network slicing and softwarization: a survey on principles, enabling technologies and solutions. *IEEE Commun Surv Tutor*. 2018;20(3):2429-2453.
37. Manzalini A, Saracco R. Software networks at the edge: a shift of paradigm. Paper presented at: 2013 IEEE SDN for Future Networks and Services (SDN4FNS); 2013; Trento, Italy.
38. Galis A, Clayman S, Mamatas L, et al. Softwarization of future networks and services-programmable enabled networks as next generation software defined networks. Paper presented at: 2013 IEEE SDN for Future Networks and Services (SDN4FNS); 2013; Trento, Italy.
39. Curry E. Message-oriented middleware. In: *Middleware for Communications*. Chichester, UK: John WIley & Sons; 2004:1-28.
40. Kathiravelu P, Van Roy P, Veiga L. Software-defined data services: interoperable and network-aware big data executions. Paper presented at: 2018 Fifth International Conference on Software Defined Systems (SDS); 2018; Barcelona, Spain.
41. Xiong R, Du Y, Jin J, Luo J. HaDaap: a hotness-aware data placement strategy for improving storage efficiency in heterogeneous Hadoop clusters. *Concurrency Computat Pract Exper*. 2018;30(20):e4830.
42. Vinoski S. Advanced message queuing protocol. *IEEE Internet Comput*. 2006;10(6):87-89 .
43. Hunkeler U, Truong HL, Stanford-Clark A. MQTT-S—a publish/subscribe protocol for wireless sensor networks. Paper presented at: 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE); 2008; Bangalore, India.

44. Johns M. *Getting Started With Hazelcast*. Birmingham, UK: Packt Publishing Ltd; 2013.
45. Marchioni F, Surtani M. *Infinispan Data Grid Platform*. Birmingham, UK: Packt Publishing Ltd; 2012.
46. Kathiravelu P, Veiga L. SDN middlebox architecture for resilient transfers. Paper presented at: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM); 2017; Lisbon, Portugal.
47. Bajpai V, Eravuchira SJ, Schönwälder J. Lessons learned from using the ripe atlas platform for measurement research. *ACM SIGCOMM Comput Commun Rev*. 2015;45(3):35-42.
48. Antonioletti M, Atkinson M, Baxter R, et al. The design and implementation of Grid database services in OGSA-DAI. *Concurrency Computat Pract Exper*. 2005;17(2-4):357-376.
49. Newcomer E, Lomow G. *Understanding SOA With Web Services*. Boston, MA: Addison-Wesley; 2005.
50. Rupprecht L. *Network-Aware Big Data Processing* [PhD thesis]. London, UK: Imperial College London; 2017.
51. Paiva J, Ruivo P, Romano P, Rodrigues L. Autoplacer: scalable self-tuning data placement in distributed key-value stores. *ACM Trans Auton Adapt Syst*. 2015;9(4). Article No 19.
52. McClatchey R, Anjum A, Stockinger H, Ali A, Willers I, Thomas M. Data intensive and network aware (DIANA) grid scheduling. *J Grid Comput*. 2007;5(1):43-64.
53. Marozzo F, Rodrigo Duro F, Garcia Blas J, Carretero J, Talia D, Trunfio P. A data-aware scheduling strategy for workflow execution in clouds. *Concurrency Computat Pract Exper*. 2017;29(24):e4229.
54. Wang S, Zhou A, Yang F, Chang RN. Towards network-aware service composition in the cloud. *IEEE Trans Cloud Comput*. 2016;1:1-14.
55. Klein A, Ishikawa F, Honiden S. Towards network-aware service composition in the cloud. In: Proceedings of the 21st International Conference on World Wide Web; 2012; Lyon, France.
56. Dabek F, Cox R, Kaashoek F, Morris R. Vivaldi: a decentralized network coordinate system. *ACM SIGCOMM Comput Commun Rev*. 2004;34(4):15-26.
57. Zhang Q, Zhu Q, Zhani MF, Boutaba R, Hellerstein JL. Dynamic service placement in geographically distributed clouds. *IEEE J Sel Areas Commun*. 2013;31(12):762-772.
58. Oppenheimer D, Chun B, Patterson D, Snoeren AC, Vahdat A. Service placement in a shared wide-area platform. In: Proceedings of the 2006 USENIX Annual Technical Conference, General Track; 2006; Boston, MA.
59. Kathiravelu P, Grbac TG, Veiga L. Building blocks of mayan: componentizing the escience workflows through software-defined service composition. Paper presented at: 2016 IEEE International Conference on Web Services (ICWS); 2016; San Francisco, CA.
60. Jararweh Y, Al-Ayyoub M, Benkhelifa E, Vouk M, Rindos A. SDIoT: a software defined based Internet of Things framework. *J Ambient Intell Humaniz Comput*. 2015;6(4):453-461.
61. Darabseh A', Al-Ayyoub M, Jararweh Y, Benkhelifa E, Vouk M, Rindos A. SDSecurity: A software defined security experimental framework. Paper presented at: 2015 IEEE International Conference on Communication Workshop (ICCW); 2015.
62. Darabseh A, Al-Ayyoub M, Jararweh Y, Benkhelifa E, Vouk M, Rindos A. Sdstorage: a software defined storage experimental framework. Paper presented at: 2015 IEEE International Conference on Cloud Engineering; 2015; Tempe, AZ.
63. SAP. Sap data services. 2018. https://www.sap.com/products/data-services.html
64. Oracle. Oracle rest data services. 2018. http://www.oracle.com/technetwork/developer-tools/rest-data-services/overview/index.html
65. Portworx. Portworx. 2018. https://portworx.com/
66. PureStorage. The data platform for the cloud era. 2018. https://www.purestorage.com/uk/products.html
67. IBM. IBM spectrum copy data management. 2018. https://www.ibm.com/us-en/marketplace/spectrum-copy-data-management
68. Catalogic. Catalogic ECX. 2018. https://catalogicsoftware.com/products/ecx/
69. Red hat storage: Why software-defined storage matters. 2016. https://www.redhat.com/en/about/videos/why-software-defined-storage-matters
70. Groff J. VMware certified software-defined data services offering enables enhanced performance gains for tier-1 virtualized applications. 2016. https://www.primaryio.com/vmware-certified-software-defined-data-services-offering-enables-enhanced-performance-gains-for-tier-1-virtualized-applications/
71. PrimaryIO. PrimaryIO: application performance accelerator (APA) 2.5. 2018. http://www.primaryio.com/
72. Sheikh H. HPE hyper converged: Hewlett Packard Enterprise (HPE). 2016. https://tdhpe.techdata.eu/Documents/SWEDEN/Server%20on%20Tour/Hyper-Converged%20for%20TechData%20Sweden_HasanSheikh.pdf
73. Mellor C. Hammer hopes to nail software-defined future for commvault. 2016. https://www.theregister.co.uk/2016/10/26/commvault_set_fair_for_sustained_turnaround/
74. Commvault. Commvault introduces new innovations for the commvault data platform in software defined data services, orchestration and user interface. 2016. https://www.commvault.com/news/2016/october/commvault-introduces-new-innovations-for-the-commvault-data-platform-in-software-defined-data-services-orchestration-and-user-interface
75. Merkel D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*. 2014;2014(239). Article No 2.

How to cite this article: Kathiravelu P, Van Roy P, Veiga L. Interoperable and network-aware service workflows for big data executions at internet scale. *Concurrency Computat Pract Exper*. 2020;32:e5212. https://doi.org/10.1002/cpe.5212