Future Generation Computer Systems 110 (2020) 107–118

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

An edge-aware autonomic runtime for data streaming and in-transit processing

Ali Reza Zamani [*], Daniel Balouek-Thomert, J.J. Villalobos, Ivan Rodero, Manish Parashar

Rutgers Discovery Informatics Institute, Rutgers University, USA

A R T I C L E I N F O

Article history:
Received 16 March 2019
Received in revised form 1 March 2020
Accepted 14 March 2020
Available online 3 April 2020

Keywords:
Stream-oriented workflows
Edge processing
In-transit processing
Wide-area analytics

A B S T R A C T

One of the major endeavors of modern cyberinfrastructure (CI) is to carry content produced on remote data sources, such as sensors and scientific instruments, and to deliver it to end users and workflow applications. Maintaining data quality, data resolution, and on-time data delivery and considering the increasing number of computing, storage, and network resources are challenging tasks that require a receptive system able to adapt to ever-changing demands. In this paper, we propose a mathematical model of such system by expressing the dynamic stages of different resources in the context of edge and in-transit computing. By considering resource utilization, approximation techniques, and user constraints, our proposed model generates mappings of different workflow stages on heterogeneous geographically distributed resources. Specifically, we propose an autonomic runtime management layer that adapts the data resolution being delivered to the users by implementing feedback loops over the resources involved in the delivery and processing of data streams. The implementation of our model is based on a subscription-based data streaming framework that enables the integration of large facilities and advanced CI. Moreover, the idea of stream or request aggregation is incorporated into our framework, which eliminates redundant data streams to save bandwidth. Experimental results show that dynamically adapting data resolution and stream aggregation can overcome bandwidth limitations in wide-area streaming analytics by leveraging the resources at the edge and in-transit.

1. Introduction

Large-scale observatories, such as the Laser Interferometer Gravitational-Wave Observatory (LIGO) [1], Large Synoptic Survey Telescope (LSST) [2], Large Hadron Collider (LHC) [3], Square Kilometer Array (SKA) [4], NSF Ocean Observatories Initiative (OOI) [5], and National Ecological Observatory Network (NEON) [6], are designed to provide the scientific community with open access to data and data products generated from geographically distributed instruments and sensors. As the number of sensors and their accuracy (e.g., image resolution and sampling rate) increases over time, the volume, variety, and velocity of generated data exponentially grow. In such ecosystems, processing large volume of data requires a large amount of resources, which are typically not co-located with the data sources [7].

Data processing is usually carried out in outsourced and remote locations within well-provisioned data centers in public or private clouds or academic institutions. Advanced cyberinfrastructure (ACI), such as XSEDE (e.g., Jetstream [8]) and commercial cloud resources (e.g., Amazon Web Services and Google Cloud), play an important role in addressing access limitations to compute and storage resources by providing elastically provisioned on-demand resources to users. To use remote ACI, the data should be outsourced to remote resources for further processing [9], which is not trivial as data rates grow.

In this context, end users and applications require to receive processed or transformed data and data products meeting specific constraints, such as deadline, budget, and quality. We refer to these constraints as quality of service (QoS). However, guaranteeing on-time data delivery within specific constraints imposed by users in environments composed of heterogeneous resources, such as network links, virtual machines, and bare-metal servers, requires sophisticated service/resource orchestration. Furthermore, the implementation of complex workflows based on streamed data and online data product generation while maintaining QoS requirements require comprehensive monitoring.

Widely adopted technologies for data stream processing, such as Apache Storm [10], are typically deployed within a cluster or datacenter; however, as data processing is conducted closer to the edge, centralized on-time stream processing is not possible due to the size of the data and network bandwidth limitations [11]. Recent research efforts have addressed this issue by introducing the concept of edge computing and filtering the data at the

[*] Corresponding author.
E-mail address: alireza.zamani@rutgers.edu (A.R. Zamani).

https://doi.org/10.1016/j.future.2020.03.037
0167-739X/© 2020 Elsevier B.V. All rights reserved.

edge of the network [11–14]. As opposed to existing works, our proposed solution considers stream-oriented workflows on edge and in-transit nodes and combines data streams to reduce data movement based on resource location, resolution, and user constraints.

In this paper, we propose a framework that integrates and utilizes heterogeneous and distributed resources to process data while they move toward the users and manages data resolution to satisfy QoS requested by the users. We demonstrate that edge and in-transit resources can be leveraged to process the data and adjust the data quality or resolution while they move between geo-distributed nodes. The framework considers user constraints (deadline, budget, and data resolution) and status of the resources in deploying workflows and coordinating data streams. A key feature of our approach is the ability to manage the data resolution being delivered to users at runtime based on the comprehensive monitoring of the streams. Our proposed framework targets large-scale observatory applications, along with infrastructure with similar characteristics, such as the Internet of Things (IoT) applications and cyber–physical scientific experiments. The main contributions of this study are summarized as follows:

- Proposes the deployment and scheduling of the workflow stages over geo-distributed resources located between the data source and its destination
- Leverages on-demand feedback loops to ensure end-to-end QoS for the users and dynamically adjusts the quality of data at runtime
- Provides a design and deployment of a subscription-based data streaming framework, consisting of Apache Kafka clusters [15], which incorporate edge and in-transit resources to workflow deployments
- Formulates a model to deploy stream-oriented workflows on heterogeneous resources
- Proposes a data stream aggregation or clustering mechanism to reduce the network traffic and remove redundant data transfer and processing

The remainder of this paper is organized as follows. Current data delivery limitations in scientific observatories are discussed in Section 2. The problem of allocating workload using a comprehensive mathematical model is formulated in Section 3. The proposed framework and its implementation are explained in Sections 4 and 5, respectively. Experimental results are presented in Section 6, followed by related work in Section 7. Finally, the conclusions and summary of the results are provided in Section 8.

2. Data delivery limitations in scientific observatories

Large-scale scientific facilities are an essential part of the science and engineering enterprise. Generated data products from sensors and instruments within these facilities are made to be accessible for users around the world. For instance, the OOI currently serves data from 57 stable platforms and 31 mobile assets, carrying 1,227 instruments and providing over 100,000 scientific and engineering data products [16,17].

Every second, a massive amount of data are generated from distributed devices that need to be processed in a timely manner. As the size of the data grows, processing and storing these data become challenging, costly, and time consuming. These challenges cause limitations that have negative impacts on scientific discoveries. Although tremendous efforts have been made by the community to use public or private cloud and ACI services [18–20], a huge gap between ACI and scientific facilities still exists, which makes users a part of the deployment and delivery cycles.

In such environments, more effective data delivery mechanisms that can better integrate large facilities with cyberinfrastructure (CI) services, dynamically and automatically provide execution environments, and leverage multiple resources from different entities to provide QoS for the users are needed. Furthermore, the quality of data flowing toward users needs to be reduced or adjusted (if applicable) at runtime to satisfy more requests from users and overcome network bandwidth limitations [21,22].

As the resources near the sensors and devices are limited, data are usually processed at centralized data centers. However, with the current trend in big data applications and network limitations, this model is no longer sustainable. Hence, a new model that can integrate the edge resources (closer from the data source) and in-transit nodes (between edge and core resources) can increase the efficiency of workflow execution and data processing by filtering unwanted data and reducing network traffic. As such, edge and in-transit nodes have better network connection compared with data centers located far away from data sources.

Our proposed framework aims at executing stream-oriented workflows considering user requirements and constraints using geo-distributed resources. The framework makes decisions related to data movement between different components and the quality of processed data being delivered to users. These decisions are static (to map workflow stages to the resources) and dynamic (based on the current state of the resources, timing, and data resolution). Current observatories and IoT applications require such framework to effectively deliver and process data by considering the heterogeneous nature and properties (computing power and cost) of the resources and constraints expressed by users. Hence, a system that monitors the progress of the workflow to meet user demands is necessary. Finally, such framework can integrate ACIs into the processing cycles and fill the gap between ACIs and large-scale observatories.

3. Problem definition and model

In this section, a mathematical model that enables the mapping of the workflow stages to available heterogeneous geographically distributed resources considering deadline and budget constraints is proposed. The objective is to minimize the overall wide area network (WAN) traffic caused by each stream. The inputs of the model are the workflow description and constraints (deadline and budget) that are imposed by users and status of resources. The output is the mapping between workflow stages and resources. Each stage, except the source and sink, acquires data from the previous stage, performs several operations on the data, and provides the data to the next stage.

Processing each consumer request is a computational job in the system. Any given job J is represented by a sequence of stages $S : \{S_0, S_1, \ldots, S_Z, S_{Z+1}\}$, forming a workflow pipeline. The data production stage is considered S_0, and the data consumption stage is considered S_{Z+1}. Fig. 1 describes an overview of the pipeline workflow model. The processed data need to be sent to the consumer site for storage, visualization, and potentially additional offline processing with historical data.

The constraints of a job J includes a deadline ($Deadline(J)$) by which results have to be placed at the destination, typically determined by users, and a budget ($Budget(J)$) describing the maximum amount available to the users to spend on the computing job J.

A set of q geographically distributed computing resources (nodes) $R : \{r_1, \ldots, rq\}$ are in charge of data processing and applying part of the workflow. r_0 and r_{q+1} represent the producer and consumer hops, respectively. Consequently, the available set of hops is defined as $H : \{r_0, r_1, \ldots, r_q, r_{q+1}\}$. The following variables are used to characterize the problem:

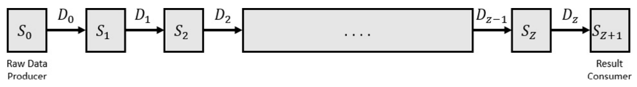


Fig. 1. Workflow pipeline.

- $Task\_num(J, S_i)$: The number of tasks that is associated with stage S_i. Note that each stage is composed of several tasks that should be executed sequentially in order complete stage S_i.
- $P(r_i)$: The average number of tasks that resource r_i executes per unit of time.
- $E(J, r_i)$: The time job J spent computing at resource r_i.
- $ES(J, S_i, r_j)$: The time job J spent computing stage S_i at resource r_j.
- $CompCost(r_i)$: The cost per unit of time for using resource r_i for computation.
- $T(J, r_i, r_k)$: The time spent transferring data between resources r_i and r_k for job J.
- $CostNet(r_i, r_k)$: The cost of using the network channel per unit of data size, between resources r_i and r_k.
- $Bandwidth(r_i, r_j)$: The available network bandwidth, between resources r_i and r_j.
- $Dist(r_i, r_j)$: The geographic distance between r_i and r_j.

Additional variables $L_{i,j}(J)$ are used to determine the mapping of stage S_i to node r_j.

$$L_{i,j}(J) : \begin{cases} 1 \rightarrow \text{ if stage } S_i \text{ is mapped to resource } r_j \\ 0 \rightarrow \text{ if stage } S_i \text{ is not mapped to resource } r_j \end{cases}$$

The production and consumption stages are mapped to the producer and the consumer sites, respectively.

Each stage is mapped to exactly one hop. Eq. (1) shows that stage i of the workflow should be executed on exactly one node and the stages are not preemptive.

$$\sum_{j=0}^{q+1} L_{i,j}(J) = 1 \tag{1}$$

As depicted in Fig. 1, the size of data generated at stage i is assumed to be D_i. Other than D_0, which is the data generated from devices, the data resolution affects the size of the data generated in each stage. For the mapping function, we considered the minimum acceptable resolution to determine the size of data at each stage. The overall time needed to process a job J is defined as:

$$CompTime(J) = \sum_{j=0}^{q+1} E(J, r_j) + Transfer(J)$$

the $Transfer(J)$ and $E(J, r_j)$ are measured as follows:

$$Transfer(J) = \sum_{i=0}^{z} T(J, S_i)$$

$$E(J, r_j) = \sum_{i=0}^{Z+1} ES(J, S_i, r_j) * L_{i,j}(J)$$

$$ES(J, S_i, r_j) = Task\_num(J, S_i)/P(r_j)$$

Basically, the total transfer time of job J is equal to the sum of the time spent transferring data for each stage (between the current stage and next stage) excluding the consumption stage. We consider that the time it takes for the producer to generate raw data and for the consumer to consume the processed data are negligible.

The data transfer time between stage i and stage $i + 1$ can be measured as follows:

$$T(J, S_i) = \sum_{j=0}^{q+1}\sum_{k=0}^{q+1} L_{i,j} * L_{i+1,k} * D_i/Bandwidth(r_j, r_k)$$

The cost of computing job J, $Cost(J)$, is defined as:

$$Cost(J) = CostExec + CostNet$$

where the computational cost $(CostExec)$ is defined as:

$$CostExec = \sum_{i=0}^{q+1}[CompCost(r_i) * E(J, r_i)]$$

The cost of transferring data associated with a job $(CostNet)$ is defined as:

$$CostNet = \sum_{j=0}^{q+1}\sum_{k>j}^{q+1}[Datasize(J, r_j, r_k) * CostNet(r_j, r_k)]$$

Where the data size between two resources r_j and r_k is measured as follows:

$$Datasize(J, r_j, r_k) = \sum_{i=0}^{z} L_{i,j} * L_{i+1,k} * D_i \tag{2}$$

Eq. (2) is derived by considering execution of stage S_i on r_j and stage S_{i+1} on r_k.

These general formulations are subject to ensuring the QoS requirements of each processed job:

$$CompTime(J) \leq Deadline(J) \tag{3}$$

$$Cost(J) \leq Budget(J) \tag{4}$$

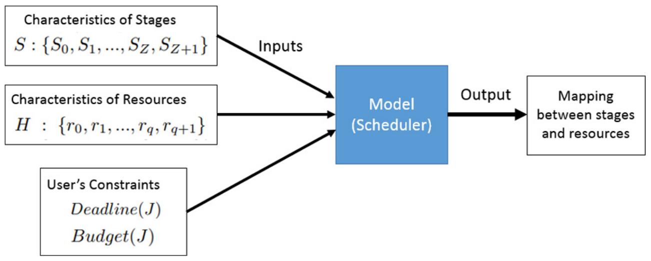
Aside from satisfying user constraints, as the network bandwidth plays an important role in streaming engines, the objective of this model is to minimize the overall WAN traffic between different components for each stream. Aside from minimizing the network bandwidth usage, the secondary effect of this objective is to minimize the amount of energy that is being consumed due to data transfer. The geographical distance between different components and data size are considered for the data movement minimization. Our overall objective is to minimize the function below:

$$\sum_{i=0}^{q+1}\sum_{j=0}^{q+1} Datasize(J, r_i, r_j) * Dist(r_i, r_j) \tag{5}$$

Fig. 2 summarizes the inputs and outputs of our model. The proposed model is solved using the linear programming optimizer PuLP [23] to determine $L_{i,j}(J)$. [24] has mathematically proven that the mapping of linear workflows on heterogeneous resources is NP-complete and finding an optimal solution can take a prohibitive time if the number of resources dramatically increases. However, an approximate solution can be reached by considering a subset of the nodes for the optimization process. Of note, the model presented in this paper is designed to minimize the amount of WAN traffic. Other strategies, such as random mapping, minimization of cost, or execution time, can be considered and deployed [25,26].

4. Stream-oriented data processing framework

In this section, a framework that targets the execution of stream-oriented pipeline workflows or applications is proposed. These applications are modeled and executed as sequential functions and modules [24], which traditionally have been called linear workflows. The proposed framework is built on top of



Characteristics of Stages
$S : \{S_0, S_1, ..., S_Z, S_{Z+1}\}$
Characteristics of Resources
$H : \{r_0, r_1, ..., r_q, r_{q+1}\}$
User's Constraints
$Deadline(J)$
$Budget(J)$
Inputs
Model (Scheduler)
Output
Mapping between stages and resources

Fig. 2. Overview of the inputs and outputs of our mathematical model.

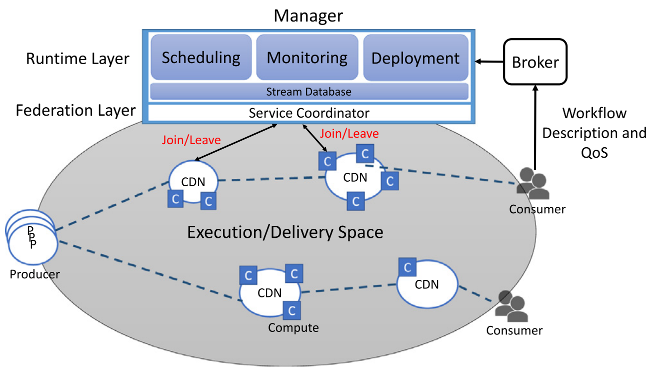


Fig. 3. Services involved in data delivery and processing.

the geo-distributed ACIs and deploys the workflow stages on available resources at different locations based on the origin and destination of data. Moreover, it monitors the execution and progress of the workflows at runtime.

4.1. Overall architecture

An overview of the architecture of the framework is illustrated in Fig. 3.

Execution/delivery space consists of geo-distributed resources and data sources, such as sensors and instruments that are part of the observatories. Resources join the *execution/delivery space* by executing lightweight *agents* in charge of giving resources access to the federation layer, managing local resources, and sending status reports. The main components of the *execution/delivery space* are as follows:

CDN servers: CDN servers are responsible for forwarding data toward the clients. The proposed streaming engine relies on *Apache Kafka*, a distributed streaming platform that stores streams of records in categories called *topics* [15].

Producer and consumer nodes: Producer nodes are responsible for acquiring data from large-scale observatories and storing them in one of the available CDN servers. Consumer nodes are end-user requested processed data with specific constraints.

Compute nodes: Compute nodes process data streams and apply workflow stages or functions on data while they move toward a destination.

The *federation layer* enables the coordination of resources and allows them to join and leave the federation as needed. The *broker* provides an interface for the programmers and end users to interact within the framework. It translates high-level instructions from users to low-level instructions used by the *runtime layer*. Constraints, priorities, and workflow description are provided by users through the *broker*. The focus of this study is on the *runtime layer*, which provides the following capabilities:

1. Scheduling: This function maps the stages of the workflows to the heterogeneous resources considering QoS, location, and available bandwidths.
2. Deployment: After the scheduling stage, the deployment layer installs the routes between the nodes and starts the execution by setting the nodes ready for the requested stream.
3. Monitoring: Our framework deploys control loops to monitor the status and progress of the workflows or streams. The monitoring service defines the execution plan and rules for the nodes and requests them to notify the monitoring service if the progress of the workflow does not follow the execution plan.
4. Stream database: This module stores an up-to-date information regarding the status of the streams currently being delivered to users. The main benefit of this module is to help schedulers in reducing the amount of streams or data moving toward the users and clustering the requests, which is further explained below.

4.2. Combining user requests

In scientific and IoT ecosystems, a significant amount of temporal locality exists among the client requests [27]. Moreover, in large-scale observatories, users or applications are interested in particular data sources or request to apply the same workflow on the same data [27,28]. For example, users may want to receive the processed data if a specific event occurs in one of the data sources or if applications running on distributed nodes request information generated from the same data sources in (near) real time. In [27], Shannigrahi et al. explored current request patterns in large climate data distribution. They have shown that the requests are indeed aggregatable and data aggregation can reduce the load on the servers.

In this context, we propose to leverage shared resources wherever and whenever possible to optimize resource usages and process more requests by sharing resources across requests. If the data source and workflow that are requested by users are similar, then leveraging this approach causes multiple streams to be clustered together, which reduces redundant executions and data transfers. The main benefits of this approach are the reduction of the bandwidth and computation of resource usage per request by sharing resources for multiple requests. Combining or clustering of the streams is performed based on the user geographic location, requested data, and workflow, the current stream being delivered, and user preferences.

5. Implementation

5.1. Subscription-based data movement

A publish/subscribe messaging system is the main technique for data movement between components. When the manager receives a request from a consumer, it maps the workflow stages to the nodes and finds an appropriate path. Then, the deployment layer installs the necessary subscriptions on the nodes that will be involved in each particular data stream. After the data path is set, the stream of data moves toward its destination. Fig. 4 shows how subscription-based data movement is performed within the execution space. Specifically, the deployment layer provides each node with topics and IP addresses for the publish/subscribe method. By using this approach, data move toward the clients without any requirement on handling the data transfer procedure for every data chunk. If part of the execution is assigned to any of the compute nodes, then it subscribes to the associated data topic and CDN server, applies the function on the stream, and publishes the partially or completely processed data back to the CDN server (as depicted in Fig. 4).

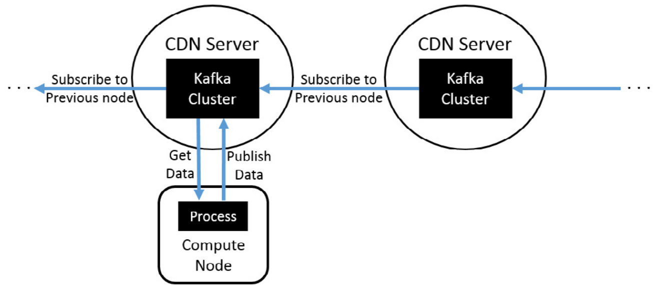
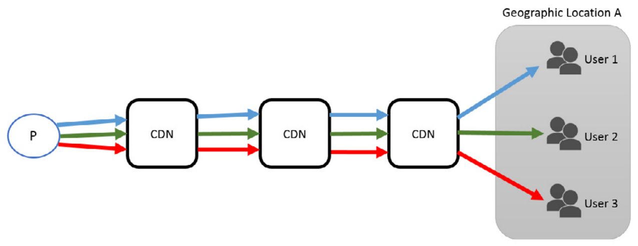
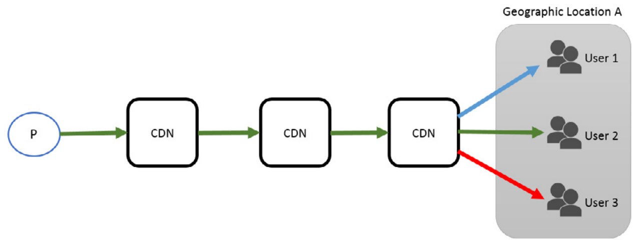


Fig. 4. Subscription-based data movement using distributed Apache Kafka clusters.



(a) Regular stream processing.



(b) Stream processing while combining the requests is enabled.

Fig. 5. Comparing regular stream processing with combining requests approach. Worker nodes have not shown here for simplicity.

5.2. Resource join procedure

Each component (CDN, producer, consumer, and compute) can access the execution space by knowing the IP address of one of the bootstrap nodes and sending join/leave requests to that IP. Upon receiving a join request, the service coordinator provides the IP address of CDN servers present in the system. Each component runs iperf [29] to the provided IP addresses to measure the available bandwidth between itself and remote IP addresses. This bandwidth information is reported to the manager that stores them in a database and virtually creates a network of nodes. This information is periodically measured and reported to the manager. Based on the network connectivities, the manager assigns a CDN server to providers, consumers, and compute nodes. A production solution can be built on top of perfSONAR [30], which is a multi-domain network monitoring and measurement framework.

5.3. Monitoring and approximation

The monitoring is implemented through a series of *rules/ thresholds, actions, and reactions*, which are established by the manager. *Rules/thresholds, actions, and reactions* create feedback loops between the resource and monitoring systems, which allows the monitoring service to control infrastructure and workflow progress. *Rules/thresholds* are constraints that are installed on the compute nodes. They are locally checked by each node before and after the execution of each stage. Each of the *rules/ thresholds* is associated with an *action* that is also installed on the compute nodes and indicates an action that each node should take if one of the thresholds is met or rules are violated. For each stream, a thread is created at each of the corresponding compute nodes that is responsible to acquire or publish the data, provide the data for compute process, and check the *rules/thresholds* and take *actions* if needed.

We consider data approximation, i.e., adjusting the data resolution, in two different ways. (i) In the scheduling procedure, the minimum resolution that is needed for each request is considered for scheduling. (ii) Approximation is tied to monitoring services, such that the rules and threshold can identify monitoring services if the resolution of data is sufficient. Then, based on this information, at runtime, the framework can change the resolution of the generated data at each node by increasing or decreasing it if needed. The increase or decrease in the data resolution is amended for the next data chunk by instructing previous nodes to publish data with higher or lower resolution. The decrease in resolution is also applied for the current data chunk by publishing lower-resolution data to the next node.

Here, the *action* at each node is to inform monitoring service. The *reaction* is the decision of the monitoring service, which is data resolution adjustment for upstream nodes. Considering the deadline and estimated data transfer and execution time, the manager estimates the arrival time of data at each node. Based on the difference between the actual arrival time and estimated arrival time of data (called *diff*), five different categories are considered in this study. The thresholds and reactions are listed in Table 1.

The runtime strategy for the data resolution is to start the delivery of the streams with the minimum required resolution and adjust the resolution at runtime. Of note, we used fixed timing and conditions. However, strategies where timing and conditions are different for various requests and changing at runtime can be considered and deployed. In this study, the resolution reduction rate is set higher than the induction rate (as mentioned in Table 1) to ensure that data are delivered without delay.

Table 1
Conditions and monitoring reactions.

Condition	Timing (s)	Monitoring reaction
Very early	$2 < diff$	Increase resolution by 5%
Early	$1 < diff \leq 2$	Increase resolution by 2%
On-time	$0 \leq diff \leq 1$	N/A
Late	$-1 \leq diff < 0$	Decrease resolution by 5%
Critically late	$diff < -1$	Decrease resolution by 10%

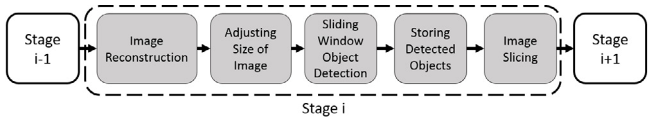


Fig. 6. Sliding window and image pyramids.

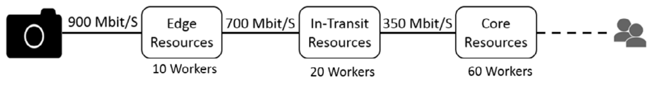


Fig. 7. Infrastructure consisting of producer, edge, in-transit, and core resources.

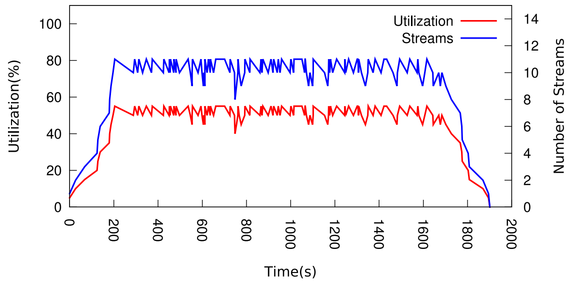


Fig. 8. Utilization and the number of streams without approximation, edge, and in-transit resources.

5.4. *Combining the requests at runtime*

Once a request is submitted to the system by users or applications, such request is evaluated and the workflow on the geo-distributed resources is scheduled. However, the scheduler first contacts the stream database and inquires for any possible way that this request can be satisfied using the existing data streams that are currently being served. If the requested resolution, data source, and workflow can be satisfied by other requests or streams, then these information will be sent to the deployment module. Then, the deployment module adds more steps that should be taken to combine these requests. Next, the processed data will be delivered to the users or applications and the request will be satisfied. If satisfying the new request could not be accomplished by combining the requests, then the scheduler would try to schedule the workflow from scratch using the model explained in Section 3.

In addition, as our proposed framework is based on a publish/subscribe model, clustering the streams is fast and feasible without the need to adjust existing streams. It is fast and feasible because once the new request can be satisfied using existing data streams, then we just need to add few additional subscriptions between the new data path and previous existing path. For example, in Fig. 5, after the clustering of new requests with an existing one is determined by the runtime layer and manager, the client or user program is instructed to subscribe to the corresponding topic and data that are already existent in the CDN node to start receiving processed data.

The main advantage of our proposed approach is shown in Fig. 5. In regular streaming approaches, separate data streams with the same content are sent to different users. However, in this work, we send one stream for the same data source and workflow to a specific geographic region and add subscriptions at proper locations to save computing resources and, more importantly, network bandwidth resources. Clearly, when the request clustering is available, redundant data transfers are eliminated.

6. Experimental evaluation

6.1. *Workflow*

In this work, our target is to use images captured by OOI's [16] high-quality underwater cameras as our image stream inputs. These images are processed to identify the different types of fish appearing in them. The image processing and object detection workflow used in this study are derived from Dalal et al. [31]. This method, which is traditionally called *sliding window and image pyramids*, is mainly used for object detection algorithms in image data sets. A stage of the workflow is shown in Fig. 6. Accordingly, three consecutive sliding window stages are considered for this study (three-stage workflow).

6.2. *Experimental setup and scenario*

The experimental setup is comprised of three types of resources: Edge, in-transit, and core resources with 10, 20, and 60 compute nodes (each node has 1 CPU at 2.4 GHz and 1 GB RAM), respectively. Edge resources are the closest resources to the producer, and the producer-edge bandwidth is higher than the producer-in-transit and producer-core bandwidths. Evaluation is performed on CloudLab [32], a distributed testbed for the computer science research community. It provides on-demand servers and virtual machines over distributed sites in the United States. Hierarchical token bucket (HTB) [33] has been used to deploy links with different bandwidths. Fig. 7 presents a schematic view of our infrastructure. Note that the experimental setup follows what has been described in Section 5. However, to simplify the figure eliminated the details in Fig. 7.

The cameras generate 10 MB size images every 10 s. The QoS constraints are the deadline, budget, and data resolution. If the system can process and deliver data within the requested constraints, then it accepts user requests and starts the delivery of the processed images.

In total, 194 users have joined the system, with each user requesting an independent image stream for a random period of time between 50 and 400 s. Users joined the system following a Poisson distribution with a mean of 15 min and variance of 7 min. We considered a period of 30 min for each experiment. In all of the scenarios, for each request, we used the model described in Section 3 to map workflow stages to the available resources. We also considered a deadline of 25 s and assigned $1 budget for each image in the stream. The Amazon EC2 prices for the bandwidth and the t2.micro template size for compute nodes [34] are considered in this work. The main focus of this study is the deadline and resolution constraints. If the system accepts to deliver and satisfy the minimum QoS requested by the users, then the minimum bandwidth that can satisfy such request will be allocated for that user and other streams cannot use that amount of bandwidth until the stream stops. This method is enforced by controlling the amount of data that each node is allowed to produce per second for a specific stream.

6.3. *Results*

In this section, several scenarios with various parameters for deadline and resolution are considered to indicate the effectiveness of our framework in various conditions by comparing the number of streams being delivered, resource utilization, and handling change in execution conditions. Our baseline is a current state-of-the-art solution, which streams all data to one central well-provisioned data center for processing. That is, all data go to the core resources and the workflow stages implemented at the central core data center using three workers for each stream. In the baseline scenario, each stream of data will only be processed using the 60 compute nodes at the core of the network. In other word, data does not get prepossessed using resources close to the data sources. Fig. 8 shows the utilization of the resources and number of streams being delivered to the users at any given time throughout the experiment for baseline scenario. Fig. 8 demonstrates that although free resources are available at the core, they remain unused as the bandwidth resources are limited and used for previous requests. In other words, not enough bandwidth is available for the new requests. The utilization of the infrastructure at best is 55%, and only 11 concurrent streams are guaranteed for the delivery with the requested QoS.

With the availability of edge and in-transit resources, the number of streams delivered over time is measured using the same setup. As shown in Fig. 9, if the data resolution requested

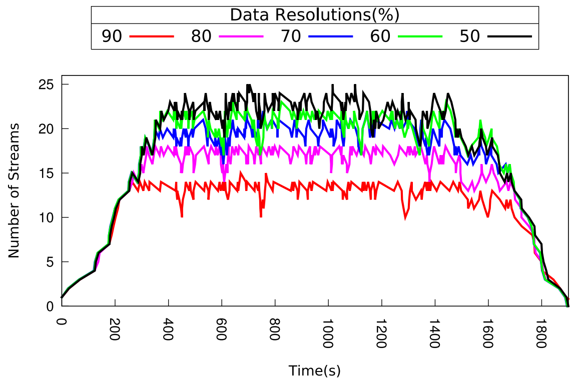


Fig. 9. Number of streams delivered to users over time for different minimum acceptable data resolutions.

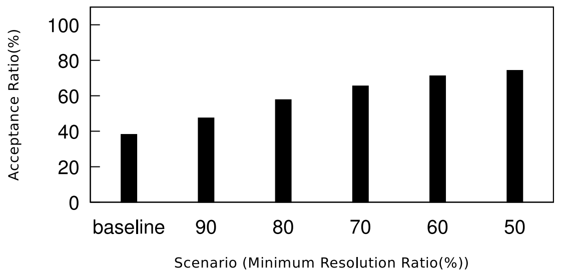


Fig. 10. Comparing acceptance ratio in baseline and edge/in-transit enabled scenarios for different qualities.

for the streams decreases, then the concurrent number of streams that can be processed and delivered increases. This case shows the effectiveness of our model in taking advantage of edge and in-transit resources to reduce the data size going toward core and end users.

To compare the baseline scenario with scenarios where edge, in-transit, and approximation resources are available, we compare the acceptance ratio of different scenarios. Acceptance ratio is the percentage of the accepted requests to the total number of requests. Fig. 10 proves that using heterogeneous resources near data sources and using them to filter unwanted data increase the number of accepted requests and potentially user satisfaction.

The resource utilization for the edge, in-transit, and core resources for three different data resolutions is shown in Fig. 11. With 80% data resolution, the utilization of core resource can reach 48%, which is slightly less than our baseline due to edge and in-transit participation. However, for 60% and 50% data resolutions, the maximum utilization of 63% and 71% have been reached, respectively, which are higher than the baseline scenario (55%). By comparing these figures, we conclude that by leveraging edge and in-transit resources, data are filtered before they reach bottleneck links and more data can be injected to the core resource, which results in more utilization at the core.

Next, we show how the system dynamically adjusts the stream resolution when the execution or delivery environment is changed at runtime. We used 12 concurrent streams for three different resolutions (100%, 80%, and 60%), with four streams each. We use the same infrastructure depicted in Fig. 7. After 220 s, the network bandwidth between in-transit and core nodes cuts down to 200 Mbit/s. Fig. 12 shows the number of streams and the average resolution of the streams delivered to the users for various request types. For high-resolution requests (e.g., 100%), the streams are stopped and nothing will be delivered to the corresponding users as the 100% strict condition does not allow the system to adjust the resolution and perform anything to recover the delivery of the streams. Moreover, the number of streams for 100% resolution decreases after the incident and reaches zero (note: a zero resolution means no streams for such resolution requirement are being delivered). For 80% minimum resolution requirement, our system shows more resistance by first reducing the resolution and then stopping the streams. However, for 60% resolution, the system can continue flawlessly by reducing the quality of the streams for a while. Of note, the resolution of 60% requests increases once other streams (100% and 80%) are dropped and more network bandwidths are available for the remaining streams (60%). The reason that the system drops the high resolution requests is because it reduces the data resolution until it reaches the minimum acceptable resolution requested by the user. Once the system reaches the point where it cannot satisfy the QoS by reducing the resolution, it drops such requests. Hence, the system dynamically adjusts the resolution at runtime and overcomes the changes in the execution space by reducing the resolution of the streams if users are willing to sacrifice the resolution. After high-resolution streams are dropped, more bandwidth resources become available for low-resolution requests. Hence the actual delivery resolution for 60% streams increases toward the end of the experiment.

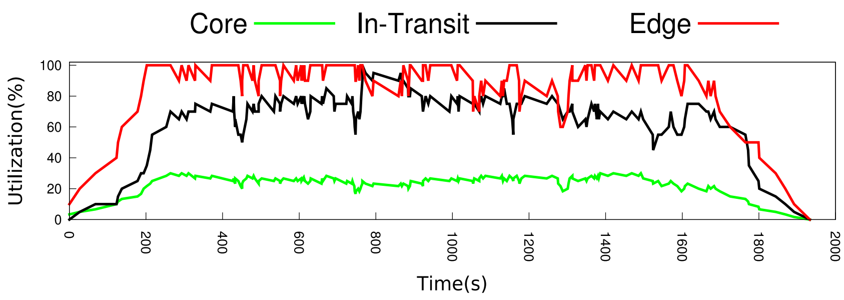
Finally, we examined the effect of different deadlines on the data resolution and number of accepted streams. We used a similar setup as the previous experiment and considered the 80% minimum acceptable resolution. The deadlines are 20, 25, and 30 s uniformly distributed across the requests. Clearly, the requests with a longer deadline require less bandwidth as they have more time available for data transfer. Hence, in general, more streams with a higher deadline are accepted (as shown in Fig. 13(a)) and data are delivered with a higher resolution for longer deadline requests (as shown in Fig. 13(b)).

6.4. Request aggregation

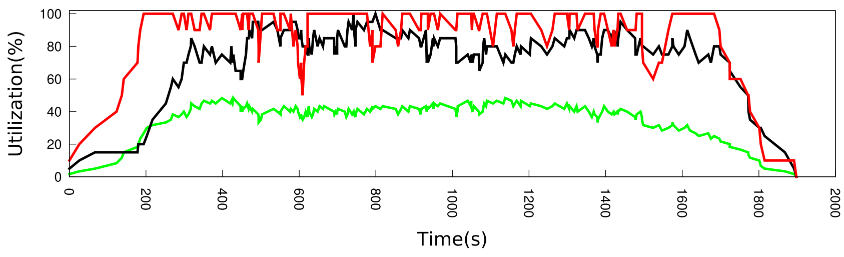
In this section, we run another set of experiments in order to show the effectiveness our proposed solution in request aggregation/combination. In this section, data delivery and processing without request aggregation is considered as our baseline scenario. We consider two metrics: (i) acceptance ratio and (ii) the number of requests being served throughout the experiment. Then, we measure these two metrics for the baseline scenario and request aggregation-enabled approach and demonstrate the advantage of our approach. Note that the baseline scheduling technique that has been considered in this experiment is similar to the one used in previous experiments.

The experimental setup of this experiment is similar to that of previous experiments shown in Fig. 7. The duration of the experiment is 30 min, in which 194 users joined the system with the Poisson distribution with a mean of 15 min and variance of 7 min. Users stayed for a random period of time and requested the camera images to be processed using the sliding window image pyramid (shown in Fig. 6) and delivered to them with 100% image resolution. The deadline is set to be 25 s. The only different in this experiment is the number of available data sources (cameras). We increased the number of available data sources or cameras. We also considered 10, 20, 30, 50, and 100 data sources for different scenarios. Users randomly picked one of the data sources and requested data to be processed and delivered to them. All of the users were located in close proximity with one another. We also assume that the users were located in one geographic region.

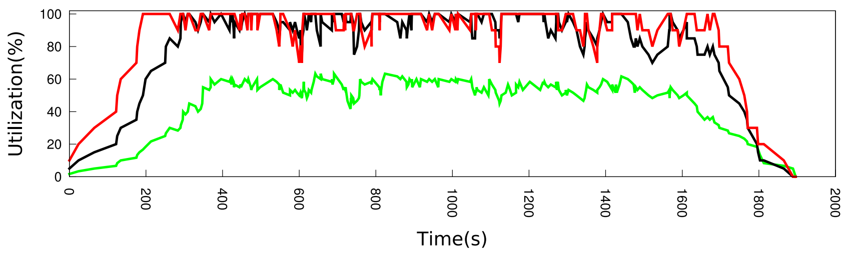
To prove the effectiveness of our request aggregation approach, we measured the acceptance ratio for the mentioned scenarios. In general, when the number of available data sources decreases, the possibility to choose a common data source for



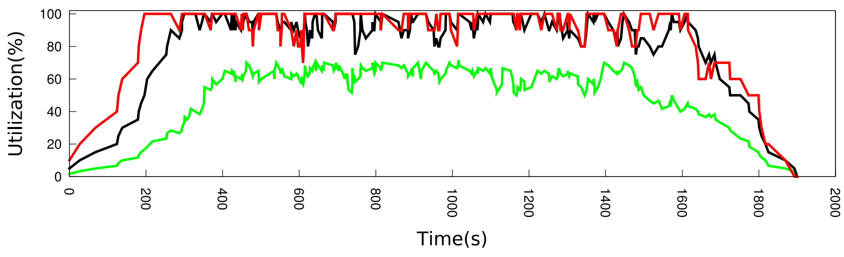
(a) 90% Data Resolution



(b) 80% Data Resolution



(c) 60% Data Resolution



(d) 50% Data Resolution

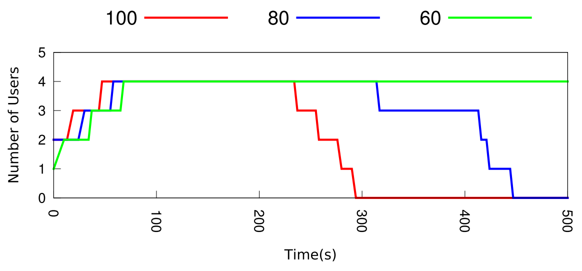
Fig. 11. Utilization of resources for different resolutions.

different users increases. Hence, our framework can cluster and accept more requests if the number of data sources is bounded. Increase in the available data sources also decreases the acceptance ratio as users can choose from a wide range of data sources and less chance for request clustering or aggregation is available. Fig. 14 shows that with 10 data sources, the framework can accept 100% of the requests. However, this number has been reduced to 64% with 20 data sources. The acceptance ratio for 100 data sources is 41%, which is slightly higher than the baseline scenario of 35%.

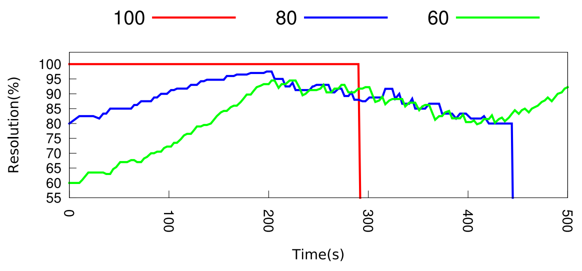
Fig. 15 demonstrates the number of requests served throughout the experiment. As depicted in Fig. 15, with a fewer number of data sources, more requests can be satisfied with our request clustering approach. As we combined the request, we reduced the bandwidth and resource usage per stream. With the same infrastructure, by sharing the resources across the requests, more users can receive their processed data within the requested QoS.

7. Related work

To process the stream of data, traditional approaches, such as stream processing engines [10,35] or workflow systems [36–38], have been designed to process data streams using resources within one cluster or data center. However, as the data sources are located far from these resources and at multiple locations, on-time stream processing is not possible due to the size of the data and network bandwidth limitations [11]. Several types of methods have been proposed to resolve this issue, for example, by introducing the concept of edge computing and filtering the data at the edge of the network, which has inspired us

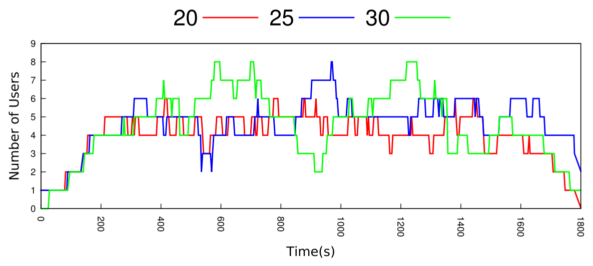


(a) Number of Streams

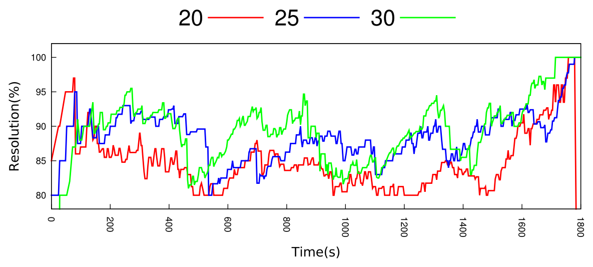


(b) Average Resolution

Fig. 12. Effect of sudden change in bandwidth on the number of streams and average resolution of the streams.



(a) Number of Streams



(b) Average Resolution

Fig. 13. Number of users and average resolution at runtime for various deadlines and resolution of 80%.

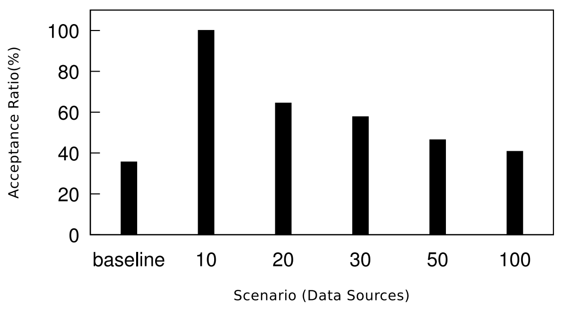


Fig. 14. Comparing acceptance ratio in baseline and data-merging availability for various data products.

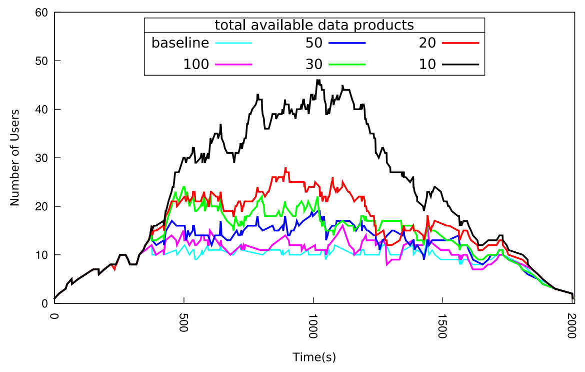


Fig. 15. Number of users served throughout the experiment.

in this work. For instance, Santos et al. [12] utilized edge resource to downsample data using edge resources. Deng et al. [39] demonstrated the trade-off between power consumption and delay in fog–cloud environment. Their solution divides the fog–cloud workload deployment into three sub-problems (fog deployment, cloud deployment, and communication delay) and solves them to achieve the best energy-delay trade-off for workload deployment. The same aspect was examined by Kaur et al. [40], who discussed the integration of IoT and edge computing and proposed the task selection and scheduling at the edge of the network using container as a service (CoaaS). Cheng et al. [41] targeted cloud-edge-based on-demand stream processing system, which automatically configures and manages stream processing tasks. Their model, which is called GeeLytics, enables on-demand edge analytics. However, our proposed solution takes another step forward by taking advantage of the in-transit nodes [42], mapping workflow stages on available resources based on location and network conditions, and placing the workflow stages to minimize network traffic. Other systems based on actor-oriented design (e.g., Kepler [43]; Pegasus [37]) acquire the workflow description from users and automatically deploy tasks on available resources using, for example, Condor-G [38]. These systems do not consider edge/in-transit processing, approximation, and flexible QoS to deploy the workflows. To the best of our knowledge, workflow management systems are specialized to deploy complex workflows on the distributed resources, whereas our framework is a stream-oriented system that tries to provide QoS using heterogeneous resources located between users and data sources. Moreover, the focus of our proposed solution is on stream-oriented workflows where data are continuously generated from the devices and need to be processed and delivered in a timely manner.

Wide-area data analytics that leverages resource connected using WAN links has been explored in the literature. Vulimiri et al. [44] explained the concept of wide-area big data (WABD) and highlighted the issues of substantial cross-data center network costs. To alleviate the issue in WABD, a system called WANalytics, which is a Hadoop-based system, has been proposed. This system automatically replicates data and initiates computation toward the edge. Pixida [45] is a wide-area data analytics scheduler that aims to minimize the inter-datacenter traffic. When data processing request is submitted to this system, it gets the task-level graph of the job from users and provides data partition from distributed storage systems. Geode [46] is an extension of WANalytics, which targets the distributed databases across data centers. Geode tries to address wide-area data analytics of data structures as SQL analysis while minimizing the bandwidth usage. Iridium [47] explores the minimization of latency in wide-area data analytics. To achieve low-latency query response time, Iridium uses a greedy heuristic optimization technique to find the best task and data placement on the geographically distributed data centers. Iridium identifies the bottleneck link between the sites and moves data on that link before it is needed. Our proposed solution takes one step forward and targets stream-oriented workflows and applies workflow stages on edge and in-transit node to reduce WAN traffic and increase utilization of the core resources.

Data processing and delivery with end-to-end QoS constraints have been explored in different studies. Karim et al. [48] mapped user QoS to the SaaS layer by developing a hierarchical QoS model and assigning QoS weights. Rosenbert et al. [49] took another approach to control the execution environment at runtime by finding service composition that meets QoS and recomposing the services at runtime if necessary. Bhat et al. [42] investigated in-transit data manipulation and proposed reactive strategies to achieve higher QoS even in the congested network conditions. Processing the data within the deadline and budget constraints have been investigated in [50] that targets cost-time optimization techniques to schedule the workflows, which is complementary to our work. Yu et al. [51] proposed a genetic algorithm to schedule the workflows under deadline and budget constraints. On the contrary, our proposed framework combines static and dynamic approaches to provide end-to-end QoS. The static approaches are the workflow stage mapping that finds the best resources that meet the deadline and budget constraints and a dynamic approach that uses feedback loops to examine the execution of the streaming workflows at runtime. In another work, Heintz et al. [52] considered the trade-off between the accuracy of the results and the time it takes to process data, i.e., timeliness. Maswood et al. [53] developed a mixed integer linear programming method to allocate resources under QoS constraints and minimizing location-dependent costs. By using a numerical evaluation, the authors showed that their approach reduces provisioning cost and energy consumption. In our framework, we provide end-to-end QoS by reducing the resolution of the data. However, the decision about this trade-off is made using on-demand control loops created after the execution of each stage on edge and in-transit nodes.

Data aggregation aims to combine responses from multiple sources into a single message. The idea of data aggregation has been mostly explored in papers based on wireless sensor network applications. As energy consumption plays a crucial role in wireless sensor nodes, data aggregation can potentially reduce the amount of energy consumption at the wireless node drastically [54]. Authors in [55] adaptively adjusted the aggregation period in different levels of their aggregation tree so that the expected query accuracy is met and end-to-end query delay is optimized. In [56], fuzzy logic in-network data aggregation was considered to increase the energy efficiency of data transmission. Aside from wireless sensor network applications, data aggregation is also used in wide-area data analytics. For instance, Heintz et al. [57] proposed the idea of group aggregation to aggregate the data at the edge nodes and reduce the amount of network traffic. Rabkin et al. [58] proposed a system called Jetstream. Their proposed system addresses a wide-area stream of queries with latency-bound requirement. This system tries to overcome the network bandwidth limitations in streaming engines. Query aggregation, which combines multiple queries, and lossy adaptive data degradation, which have been proposed by the authors, can potentially reduce the data size based on available bandwidth and latency. In this paper, we also explored the aggregation of the requests at runtime to reduce the number of streams moving between the nodes. However, our proposed solution considers stream-oriented workflows and deploys the workflows on the edge and in-transit nodes. It also combines the data streams to reduce redundant data transfers between the node based on the source and destination locations, resolution, and user constraints.

8. Conclusion

Large-scale observatories depend on the efficient processing and delivery of data generated from geographically distributed instruments and sensors. This paper introduces a subscription-based data streaming framework of the leverage edge and in-transit resources and the advanced CI to process data generated from large-scale observatories and deliver the processed data to users at various geographic locations. In addition, a run-time management system is added to propose a framework that provides QoS for users and effectively utilizes heterogeneous geo-distributed resources to maintain the application of QoS.

The proposed framework fills the gap between large-scale observatories and ACIs by automatically executing user-defined pipeline workflows on available geo-distributed resources. It also adjusts the data quality by taking advantage of the resources at the edge and in-transit resources and filters unwanted data going through the core resources. The evaluation showed that our system can increase main resource utilization by more than 10% using unwanted data filtering at edge or in-transit nodes for low-resolution requests.

Furthermore, a request aggregation technique is added to the proposed framework, which creates clusters of requests based on the workflow and data content requested by the users. This technique reduces the WAN traffic and eliminates redundant data transfer and executions. With the amount of resources, more requests have been satisfied and the number of accepted requests have drastically increased.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Ali Reza Zamani: Conceptualization, Methodology, Software, Validation, Writing - original draft, Investigation. **Daniel Balouek-Thomert:** Writing - review & editing. **J.J. Villalobos:** Resources. **Ivan Rodero:** Supervision, Writing - review & editing. **Manish Parashar:** Supervision, Writing - review & editing.

Acknowledgments

This research is supported in part by National Science Foundation (NSF), USA via grants numbers OAC 1339036, OAC 1441376, OCE 1745246, OAC 1826997, OAC 1835692, and OAC 1640834, and was conducted as part of the Rutgers Discovery Informatics Institute (RDI^2).

References

[1] Laser Interferometer Gravitational-wave Observatory (LIGO), (Last accessed on 2019), https://www.ligo.caltech.edu.
[2] W. Gressler, D.J.E. Hileman, D.R. Neill, LSST Telescope and site status, Ground-based and Airborne Telescopes V, 91451A, http://dx.doi.org/10.1117/12.2056711.
[3] L. Evans, P. Bryant, LHC machine, J. Instrum. 3 (2008) http://dx.doi.org/10.1088/1748-0221/3/08/S08001.
[4] B. Wang, X. Zhu, C. Gao, Y. Bai, et al., Square kilometre array telescope - precision reference frequency synchronisation via 1f-2f dissemination, Sci. Rep. 5 (2015) http://dx.doi.org/10.1038/srep13851.
[5] L.M. Smith, et al., The ocean observatories initiative, Oceanography 31 (1) (2018) http://dx.doi.org/10.5670/oceanog.2018.105.
[6] The national ecological observatory network (NEON), 2019, http://www.neonscience.org, (Last accessed on 2019).
[7] M. Parashar, et al., Report from the nsf large facilities cyberinfrastructure workshop, 2017, http://dx.doi.org/10.7278/S5SN074P.

[8] R. Tudoran, O. Nano, I. Santos, A. Costan, H. Soncu, L. Bougé, G. Antoniu, Jetstream: Enabling high performance event streaming across cloud data-centers, in :Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, 2014, pp. 23–34.
[9] J.L. Schnase, T.J. Lee, C.A. Mattmann, C.S. Lynnes, et al., Big data challenges in climate science: improving the next-generation cyberinfra-structure, IEEE Geosci. Remote Sens. Mag. 4 (3) (2016) http://dx.doi.org/10.1109/MGRS.2015.2514192.
[10] A. Toshniwal, et al., Storm@ twitter, in: Proc. of the 2014 ACM SIGMOD Int. Conf. on Management of Data, ACM, 2014, pp. 147–156.
[11] A. Jonathan, et al., Nebula: Distributed edge cloud for data intensive computing, IEEE Trans. Parallel Distrib. Syst. 28 (11) (2017) 3229–3242.
[12] I. Santos, M. Tilly, B. Chandramouli, J. Goldstein, Dial: distributed streaming analytics anywhere, anytime, Proc. VLDB Endow. 6 (12) (2013) 1386–1389.
[13] A.M. Rahmani, et al., Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach, Future Gener. Comput. Syst. 78 (2018) 641–658.
[14] M. Aazam, E.-N. Huh, Fog computing: The cloud-iot/ioe middleware paradigm, IEEE Potentials 35 (3) (2016) 40–44.
[15] J. Kreps, et al., Kafka: A distributed messaging system for log processing, in: Proc. of the NetDB, 2011, pp. 1–7.
[16] I. Rodero, M. Parashar, Data cyber-infrastructure for end-to-end science: experiences from the nsf ocean observatories initiative, Comput. Sci. Eng. (2019).
[17] I. Rodero, M. Parashar, Architecting the cyberinfrastructure for national science foundation ocean observatories initiative (OOI), in: 7th International Workshop on Marine Technology: MARTECH 2016, 2016, pp. 99–101.
[18] M. Parashar, M. AbdelBaky, I. Rodero, A. Devarakonda, Cloud paradigms and practices for computational and data-enabled science and engineering, Comput. Sci. Eng. 15 (4) (2013) 10–18.
[19] N. Chopra, S. Singh, Deadline and cost based workflow scheduling in hybrid cloud, in: Int. Conf. Advances in Computing, Communications and Informatics, IEEE, 2013, pp. 840–846..
[20] Y. Qin, et al., Towards a smart, internet-scale cache service for data intensive scientific applications, in: Proceedings of the 10th Workshop on Scientific Cloud Computing, ScienceCloud@HPDC 2019, Phoenix, AZ, USA, June 25, 2019, 2019, pp. 11–18, http://dx.doi.org/10.1145/3322795.3331464.
[21] A.R. Zamani, et al., Submarine: a subscription-based data streaming framework for integrating large facilities and advanced cyberinfrastructure, Concurr. Comput.: Pract. Exp. (2019).
[22] A.R. Zamani, et al., Supporting data-driven workflows enabled by large scale observatories, in: 13th IEEE International Conference on E-Science, E-Science 2017, Auckland, New Zealand, October 24–27, 2017, 2017, pp. 592–595, http://dx.doi.org/10.1109/eScience.2017.95.
[23] S. Mitchell, An introduction to pulp for python programmers, Python Pap. Monogr. 1 (2009) 14.
[24] Q. Wu, Y. Gu, Performance analysis and optimization of linear workflows in heterogeneous network environments, in: Grid Computing, Springer, 2011, pp. 89–120..
[25] A.R. Zamani, et al., Deadline constrained video analysis via in-transit computational environments, IEEE Trans. Serv. Comput. (2017).
[26] A.R. Zamani, et al., A computational model to support in-network data analysis in federated ecosystems, Future Gener. Comput. Syst. 80 (2018) 342–354.
[27] S. Shannigrahi, C. Fan, C. Papadopoulos, Request aggregation, caching, and forwarding strategies for improving large climate data distribution with ndn: a case study, in: Proceedings of the 4th ACM Conference on Information-Centric Networking, ACM, 2017, pp. 54–65.
[28] P. Costa, A. Donnelly, A. Rowstron, G. O'Shea, Camdoop: Exploiting in-network aggregation for big data applications, in: Presented as Part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation, {NSDI} 12, 2012, pp. 29–42.
[29] A. Tirumala, T. Dunigan, L. Cottrell, Measuring end-to-end bandwidth with iperf using web100, in: Presented at, no. SLAC-PUB-9733, 2003.
[30] A. Hanemann, et al., Perfsonar: A service oriented architecture for multi-domain network monitoring, in: Int. Conf. Service-Oriented Computing, Springer, 2005, pp. 241–254.
[31] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in: IEEE Conf. Computer Vision and Pattern Recognition, vol. 1, 2005, pp. 886–893.
[32] Cloudlab, 2019, https://www.cloudlab.us, (Last access 2019).
[33] HTB, 2019, https://linux.die.net/man/8/tc-htb, (Last access 2019).
[34] Amazon EC2 prices, 2019, https://aws.amazoncom/ec2/pricing/, (Last access 2019).
[35] M. Zaharia, et al., Apache spark: a unified engine for big data processing, Commun. ACM 59 (11) (2016) 56–65.
[36] E. Deelman, et al., The future of scientific workflows, Int. J. High Perform. Comput. Appl. (2017).
[37] E. Deelman, G. Singh, M.-H. Su, J. Blythe, et al., Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Sci. Program. 13 (3) (2005) 219–237.
[38] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-g: A computation management agent for multi-institutional grids, Cluster Comput. 5 (3) (2002) 237–246.
[39] R. Deng, R. Lu, C. Lai, T.H. Luan, Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing, in: Communications (ICC), 2015 IEEE International Conference on, IEEE, 2015, pp. 3909–3914.
[40] K. Kaur, T. Dhand, N. Kumar, S. Zeadally, Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers, IEEE Wirel. Commun. 24 (3) (2017) 48–56.
[41] B. Cheng, A. Papageorgiou, M. Bauer, Geelytics: Enabling on-demand edge analytics over scoped data sources, in: IEEE Int. Congress on Big Data, BigData Congress, IEEE, 2016, pp. 101–108.
[42] V. Bhat, M. Parashar, S. Klasky, Experiments with in-transit processing for data intensive grid workflows, in: Proc. of the 8th IEEE/ACM Int. Conf. on Grid Computing, IEEE Computer Society, 2007, pp. 193–200.
[43] I. Altintas, et al., Kepler: an extensible system for design and execution of scientific workflows, in: Proc. 16th Int. Conf. Scientific and Statistical Database Management, IEEE, 2004, pp. 423–424.
[44] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, G. Varghese, Wanalytics: analytics for a geo-distributed data-intensive world, in: CIDR, 2015.
[45] K. Kloudas, M. Mamede, N. Preguiça, R. Rodrigues, Pixida: optimizing data parallel jobs in wide-area data analytics, Proc. VLDB Endow. 9 (2) (2015) 72–83.
[46] A. Vulimiri, C. Curino, P.B. Godfrey, T. Jungblut, J. Padhye, G. Varghese, Global analytics in the face of bandwidth and regulatory constraints, in: NSDI, 2015, pp. 323–336.
[47] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, I. Stoica, Low latency geo-distributed data analytics, ACM SIGCOMM Comput. Commun. Rev. 45 (4) (2015) 421–434.
[48] R. Karim, C. Ding, A. Miri, An end-to-end qos mapping approach for cloud service selection, in: 2013 IEEE Ninth World Congress on Services, SERVICES, IEEE, 2013, pp. 341–348.
[49] F. Rosenberg, et al., An end-to-end approach for qos-aware service composition, in: Enerprise Distributed Object Computing Conf. 2009. EDOC'09. IEEE Int, IEEE, 2009, pp. 151–160.
[50] A. Verma, S. Kaushal, Deadline and budget distribution based cost-time optimization workflow scheduling algorithm for cloud, in: IJCA Proc. on Int. Conf. on Recent Advances and Future Trends in Information Technology, iRAFIT 2012, vol. 4, iRAFIT (7), 2012, pp. 1–4.
[51] J. Yu, R. Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, Sci. Program. 14 (3–4) (2006) 217–230.
[52] B. Heintz, A. Chandra, R.K. Sitaraman, Trading timeliness and accuracy in geo-distributed streaming analytics, in: SoCC, 2016, pp. 361–373..
[53] M.M.S. Maswood, R. Nasim, A.J. Kassler, D. Medhi, Cost-efficient resource scheduling under qos constraints for geo-distributed data centers, in: NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2018, pp. 1–9.
[54] F. Hu, X. Cao, C. May, Optimized scheduling for data aggregation in wireless sensor networks, in: International Conference on Information Technology: Coding and Computing, ITCC'05-Volume II, vol. 2, IEEE, 2005, pp. 557–561.
[55] F. Hu, C. May, X. Cao, Data aggregation in distributed sensor networks: towards an adaptive timing control, in: Third International Conference on Information Technology: New Generations, ITNG'06, IEEE, 2006, pp. 256–261.
[56] J. Zhang, P. Hu, F. Xie, J. Long, A. He, An energy efficient and reliable in-network data aggregation scheme for wsn, IEEE Access 6 (2018) 71857–71870.
[57] B. Heintz, A. Chandra, R.K. Sitaraman, Optimizing grouped aggregation in geo-distributed streaming analytics, in: Proc. of the 24th Int. Symposium on High-Performance Parallel and Distributed Computing, ACM, 2015, pp. 133–144.
[58] A. Rabkin, M. Arye, S. Sen, V.S. Pai, M.J. Freedman, Aggregation and degradation in jetstream: Streaming analytics in the wide area, in: NSDI, vol. 14, 2014, pp. 275–288.

Ali Reza Zamani received his Ph.D. and M.Sc. degrees in Computer Science from Rutgers University. He received his B.Sc. from Sharif University of Technology, Iran. His research interests are in the areas of Cloud Computing, Internet of Things (IoT) and Stream Processing.

Daniel Balouek-Thomert is a Postdoctoral Research Associate at the Rutgers Discovery Informatics Institute (RDI2). His research interests revolve around distributed systems with a focus on resource management and energy efficiency for cloud/edge computing and IoT systems. He received his Ph.D. degree from Ecole Normale Supérieure de Lyon (France).

J.J. Villalobos is part of the leadership team at the Rutgers Discovery Informatics Institute, where he is responsible for the security stability and operational excellence of the research and production advanced cyberinfrastructure, which includes Caliburn, the Rutgers supercomputer, one of the fastest academic supercomputers in the United States. J. J. Villalobos conducts original research as PI for federally-funded projects, and he is currently contributing as senior personnel in several NSF-funded projects. He has published several articles related to distributed systems, big data, large facilities and cybersecurity, and regularly conducts peer reviews for international conferences, journals and workshops. He is Senior Member of the Institute of Electrical and Electronics Engineers (IEEE), and he has over fifteen years of industry and research experience across several computer science and information technology disciplines including, but not limited to, systems architecture, high-performance computing, complex public-facing internet infrastructures and high-traffic sites reliability operations.

Ivan Rodero is Associate Research Professor and Associate Director at the Rutgers Discovery Informatics Institute (RDI^2). His research interests fall in the areas of parallel and distributed computing and include high performance computing, energy efficiency, cloud computing and big data systems. His current research also addresses new cyberinfrastructure models aiming at enabling the scalability and sustainability of next generation cyberinfrastructure. He has received various awards for his research and publications, including the 014 IEEE TCSC Young Achievers in Scalable Computing Award. He received his MS and Ph.D. degrees from Technical University of Catalonia Barcelona Tech. He is a senior member of IEEE, a senior member of ACM and a member of the American Association for the Advancement of Science (AAAS). Contact him at irodero@rutgers.edu.

Manish Parashar is Distinguished Professor of Computer Science at Rutgers University. He is also the founding Director of the Rutgers Discovery Informatics Institute (RDI2). His research interests are in the broad areas of Parallel and Distributed Computing and Computational and Data-Enabled Science and Engineering. Manish serves on the editorial boards and organizing committees of a large number of journals and international conferences and workshops, and has deployed several software systems that are widely used. He has also received a number of awards and is Fellow of AAAS, Fellow of IEEE/IEEE Computer Society and ACM Distinguished Scientist.