2018 30th International Symposium on Computer Architecture and High Performance Computing

Runtime Management of Data Quality for Scientific Observatories Using Edge and In-Transit Resources

Ali Reza Zamani, Daniel Balouek-Thomert, J. J. Villalobos, Ivan Rodero, and Manish Parashar
Rutgers Discovery Informatics Institute (RDI^2), Rutgers University, Piscataway, NJ, 08854, USA
Email: {alireza.zamani, daniel.balouek, jj.villalobos, irodero, parashar}@rutgers.edu

***Abstract*—Modern Cyberinfrastructures (CIs) operate to bring content produced from remote data sources such as sensors and scientific instruments and deliver it to end users and workflow applications. Maintaining data quality/resolution and on-time data delivery while considering an increasing number of computing, storage and network resources requires a reactive system, able to adapt to changing demands. In this paper, we propose a modelization of such system by expressing the dynamic stage of resources in the context of edge and in-transit computing. By considering resource utilization, approximation techniques and users' constraints, our proposed engine is generating mappings of workflow stages on heterogeneous geo-distributed resources. We specifically propose a 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 the data streams. We implement our model into a subscription-based data streaming framework which enables integration of large facilities and advanced CIs. Experimental results show that dynamically adapting data resolution can overcome bandwidth limitation in wide area streaming analytics.**

I. Introduction

Large scale observatories are designed to provide the scientific community with open access to data generated from geographically distributed instruments and sensors. As the number of sensors and their accuracy (e.g. image resolution) increases over time, the volume, variety and velocity of generated data grows exponentially. In such ecosystem, processing large volume of data requires large amount of resources, which are typically not co-located with the data sources.

Processing is usually carried out in external and remote locations within well-provisioned data-centers in public/private clouds or academics institutions. Advanced Cyber-Infrastructures (ACIs) such as XSEDE (e.g., Jetstream) and cloud (e.g., AWS) resources play an important role to address limited and scarce local resources by providing on-demand resources for the users. In order to use remote ACIs, the data should be outsourced to remote resources for further processing.

In this context, users and applications require to receive processed/transformed data with several particular 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 the users in environments composed of heterogeneous resources such as network links, virtual machines and bare metal servers requires sophisticated service/resource coordination. Moreover, in environments where data is continuously generated and processed via complex workflows, providing the guaranteed QoS and data quality for a long period of time while avoiding QoS degradation requires a comprehensive monitoring.

In this paper, we propose a framework that integrates and utilizes heterogeneous distributed resources to process data while it moves towards the users, and manages data resolution in order to satisfy QoS requested by the users. We prove that edge and in-transit resources can be leveraged to process the data and adjust the data quality/resolution while it is moving between geo-distributed nodes. It considers users' constraints (deadline, budget and data resolution) and status of the resources in order to deploy workflows and coordinate data streams. A key feature of our approach is the ability to manage the data resolution being delivered to the users at runtime based on the comprehensive monitoring of the streams.

The main contribution of this work is to deploy the workflow stages over geo-distributed resources located between data source and destination, and leverage on-demand feedback loops to ensure end-to-end QoS for the users. Our work involves design and deployment of a subscription-based data streaming framework, consists of Kafka clusters [1], that incorporates edge and in-transit resources for workflow deployment and dynamically adjusts the quality of data at runtime. We also propose a computational model to deploy stream oriented workflows on heterogeneous resources. This framework targets large scale observatory applications, along with infrastructures with similar characteristics such as IoT applications and cyber-physical scientific experiments.

The remainder of this paper is organized as follows. Section II motivates our work by discussing current data delivery limitations in scientific observatories. The problem of allocating workload using a comprehensive mathematical model is formulated in Section III. The proposed framework and its implementation are explained in Section IV. Experimental results are presented in Section V followed by related work in Section VI. Finally, Section VII concludes the paper and outlines future work.

978-1-5386-7769-8/18/$31.00 ©2018 IEEE
DOI 10.1109/SBAC-PAD.2018.00053

II. Data Delivery Limitations in Scientific Observatories

Large-scale scientific facilities are essential part of the science and engineering enterprise. The generated data products from sensors and instruments within these facilities are made to be accessible for users around the world. For instance, the Ocean Observatories Initiative (OOI) currently serves data from 57 stable platforms and 31 mobile assets, carrying 1,227 instruments, provides over 100,000 scientific and engineering data products [2].

Each second, massive amount of data is generated from distributed devices that needs to be processed in a timely manner. As the size of the data grows, processing and storing these data becomes challenging, costly and time consuming. These challenges cause limitations which have negative impacts on scientific discoveries. Although there has been a tremendous effort by the community to use public/private cloud and ACI services [3, 4], there is still a huge gap between ACI and scientific facilities which cause the users to be part of deployment and delivery cycle.

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

As the resources near the sensors and devices are limited, data is usually processed at centralized data centers. However, with 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 the in-transit nodes (between edge and core resources) can increase the efficiency of workflow execution and data processing by filtering unwanted data and reduce network traffic.

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 quality of the processed data being delivered to the users. These decisions are static (to map workflow stages to the resources) and dynamic (based on 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 the constraints expressed by users. Hence, a system that monitors the progress of the workflow to meet users' demands is necessary. Finally, such framework can integrate ACIs into the processing cycles and fill the gap between ACIs and large scale observatories.

III. Problem Definition and Model

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

Processing each of the consumers' requests is considered a computational job in our system. Any given job J is represented by a sequence of stages $S : \{S_0, S_1, ..., S_Z, S_{Z+1}\}$, forming a workflow pipeline. Data production stage considered as S_0 and data consumption stage considered as S_{Z+1}. Figure 1 describes an overview of the pipeline workflow model. The processed data needs 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 user to spend on computing job J.

Fig. 1: Workflow pipeline

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

- $P(r_i)$: The average number of tasks that resource r_j 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.
- $Task\_num(J, S_i)$: The number of tasks that is associated with stage S_i .
- $CompCost(r_i)$: 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. Equation 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 Figure 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, data resolution effects the size of the data generated by 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 sum of the time spent transferring data for each stage (between current stage and next stage) excluding consumption stage. We consider that the time it takes for the producer to generate raw data and consumer to consume the processed data are negligible.

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

$$T(J, S_i) = \sum_{j=0}^{q+1} \sum_{k=0}^{q+1} L_{i,j} * L_{i+1,k} * D_i / Bandwidth(r_j, r_k) \tag{2}$$

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:

$$Datasize(J, r_j, r_k) = \sum_{i=0}^{z} L_{i,j} * L_{i+1,k} * D_i \tag{3}$$

Equation 3 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:

$$CompTime(J) \leq Deadline(J) \tag{4}$$

$$Cost(J) \leq Budget(J) \tag{5}$$

Aside from satisfying users' constrains, due to the fact that the network bandwidth plays an important role in streaming engines, the objective of this model is to minimize overall wide area network traffic between different components for each stream. Geographic distance between different components and data size have been considered for data movement minimization. Our overall objective is to minimize the function below:

$$\sum_{i=0}^{q+1} \sum_{j=0}^{q+1} Datasize(J, r_i, r_j) * Dist(r_i, r_j) \tag{6}$$

The proposed model is solved using a Linear Programming Optimizer, PuLP [5], to determine $L_{i,j}(J)$. It has been mathematically proven in [6] that the mapping of linear workflows on the heterogeneous resources is NP-complete and finding an optimal solution can take a prohibitive time, if the number of resources increases dramatically; however, an approximate solution can be reached by considering a subset of the nodes for the optimization process. Note that the model presented in this paper is designed to minimize the amount of wide area traffic. Other strategies such as random mapping, minimization of cost or execution time can be considered and deployed [7, 8].

IV. Stream-Oriented Data Processing Framework

In this section, a framework has been proposed which targets the execution of stream-oriented pipeline workflows/applications. These applications are modeled and executed as sequential functions and modules [6] which traditionally have been called linear workflows. The proposed framework is built on top of 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.

A. Overall Architecture

An overview of the architecture of the framework is illustrated in Figure 2.

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 light-weighted *agents*,

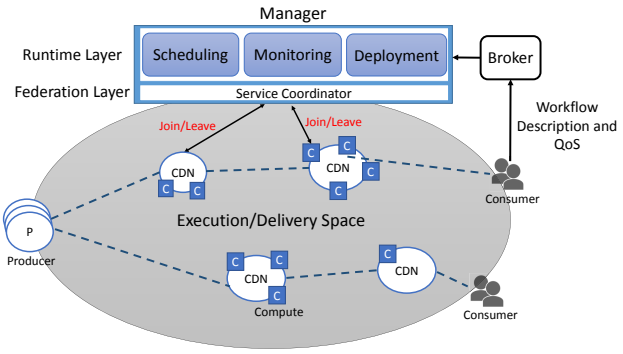


Fig. 2: Services Involved in Data Delivery and Processing

in charge of giving resources access to the federation layer, managing local resources and sending status reports. The main components of *Execution/Delivery Space* are:

CDN servers: CDN servers are responsible for moving the data toward the clients. The proposed streaming engine relies on *Apache Kakfa*, a distributed streaming platform that stores streams of records in categories called *topics* [1].

Producer and Consumer nodes: Producer nodes are responsible for getting data from large scale observatories and pushing it to one of the available CDN servers. Consumer nodes are the end-users requesting processed data with specific constraints.

Compute nodes: Compute nodes process data streams and apply workflow stages/functions on data while it moves toward 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 with the framework. It translates high-level instructions from the users to low-level instructions used by the *Runtime layer*. Constraints, priorities and workflow description are provided by the users through the *Broker*. The focus of this paper 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 scheduling stage, the deployment layer installs the routes between the nodes and starts the execution by setting the nodes ready for requested stream.
3) Monitoring: Our framework deploys control loops to check status and progress of the workflows/streams. The monitoring service defines the execution plan and rules for the nodes and asks them to notify the monitoring service if the progress of the workflow does not follow the execution plan.

B. Implementation

1) Subscription Based Data Movement: A publish/-subscribe messaging system is considered as the main technique for data movement between the components. When the manager receives a request from the 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 are going to be involved in each particular data stream. After the data path is set, the stream of data moves toward its destination. Figure 3 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 publish/subscribe method. Using this approach, data moves towards 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, it subscribes to the associated data topic and CDN server, applies the function on the stream and publishes the partially/completely processed data back to the CDN server (as depicted in Figure 3).

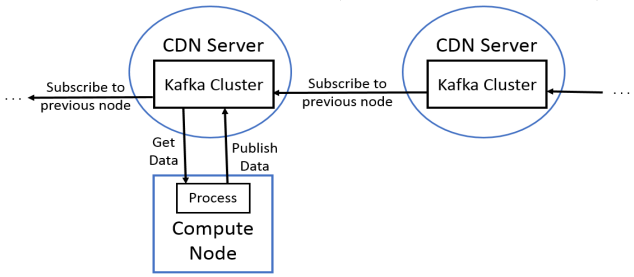


Fig. 3: Subscription-Based Data Movement Using Distributed Kafka Clusters

2) Resource Join Procedure: Each component (CDN, producer, consumer and compute) is able to 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 [9] to the provided IP addresses in order 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 could be built on top of perfSONAR [10], which is a multi-domain network monitoring and measurement framework.

3) Monitoring and Approximation: The monitoring is implemented through the series of *rules/thresholds, actions and reactions* which are established by manager. *Rules/thresholds, actions and reactions* create feedback loops between resource and monitoring systems which causes monitoring service to be able 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 execution

TABLE I: 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%

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 get/publish the data, provide the data for compute process and check these *Rules/thresholds* and take the *actions*, if needed.

We consider data approximation, i.e. adjusting the data resolution, in two different ways. (i) In 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 tell monitoring services if resolution of data is sufficient. Then, based on this information, at runtime, the framework is able to change the resolution of the generated data at each node by increasing or decreasing it, if needed. The increase/decrease in data resolution is amended for the next data chunk by instructing previous nodes to publish data with higher/lower resolution. The decrease in resolution is also applied for current data chunk by publishing lower resolution data to next node.

In this paper, 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, manager estimates the arrival time of data at each node. Based on the difference between actual arrival time and estimated arrival time of data (called $diff$), five different categories have been considered in this paper. These thresholds and reactions are listed in Table I. The runtime strategy for data resolution is to start the delivery of the streams with the minimum required resolution and adjust the resolution at runtime. It is worth noting that we used fixed timing and conditions. However, strategies where timing and conditions are different for various requests and they are changing at runtime can be considered and deployed. In this paper, in order to be sure that data is delivered without delay, resolution reduction rate has been considered higher than induction rate (as mentioned in Table I).

V. Evaluation

A. Workflow

In this work, our target is to use images captured by OOI [2] high quality underwater cameras as our image stream inputs. These images are processed in order to identify the different types of fish appearing in them. The image processing and object detection workflow used in this paper is derived from Dalal et al. [11]. Traditionally this method which is called *sliding window and image pyramids* is mainly used for object detection algorithms in image data sets. A stage of the workflow has been shown in Figure 4. Three consecutive sliding window stages have been considered for this paper (three stages workflow).

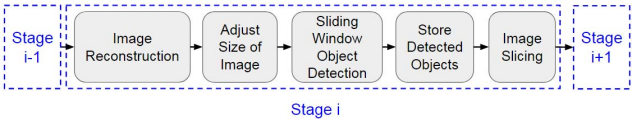


Fig. 4: Sliding Window and Image Pyramids

B. 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 requires 1 CPU and 1 GB RAM), respectively. Edge resources are the closest resources to the producer and the producer-edge bandwidth is higher than producer-in-transit and producer-core. Evaluation has been performed on Cloudlab [12], a distributed testbed for the computer science research community. It provides on-demand servers and virtual machines over distributed sites in the United States. Hierarchy Token Bucket (HTB) [13] has been used to deploy links with different bandwidths. Figure 5 presents a schematic view of our infrastructure.

Fig. 5: Infrastructure Consisting of Producer, Edge, In-transit and Core Resources

The cameras are generating 10MB size images every 10 seconds. The quality of service is deadline, budget and data resolution. If the system is able to process and deliver data within requested constraints, it accepts user requests and starts the delivery of the processed images.

In total, 194 users join the system, each user requests an independent image stream for a random period of time between 50 to 400 seconds. Users join the system following a Poisson distribution with a mean of 15 minutes and variance of 7 minutes. We considered a period of 30 minutes for each experiment. In all of the scenarios, for each request, we used the model described in Section III to map workflow stages to the available resources. Also, we considered a deadline of 25 seconds and assigned $1 budget for each image in the stream. The Amazon EC2 prices for bandwidth and the t2.micro template size for compute nodes [14] has been considered in this work. The main focus of this paper is deadline and resolution constraints. If the system accepts to deliver and satisfy minimum QoS requested by the users, the minimum bandwidth that can satisfy such request will be allocated for that user and others streams cannot use that amount of bandwidth

until the stream stops. This is enforced by controlling the amount of data that each node is allowed to produce per second for specific stream.

C. Results

In this section, several scenarios with various parameters for deadline and resolution have been considered to indicate the effectiveness of our framework in various conditions by comparing 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 the data to one central well-provisioned data center for processing, i.e., all the data goes to the core resources and the workflow stages implemented at central core data center using three workers for each stream. Figure 6 shows the utilization of the resources and number of streams being delivered to the user at any given time throughout the experiment for baseline scenario. Figure 6, demonstrates that although there are free resources available at the core, they remain unused due to the fact that bandwidth resources are limited and being used for previous requests and there is not enough bandwidth available for new requests. The utilization of the infrastructure at best is 55% and only 11 concurrent streams are guaranteed for delivery with requested QoS.

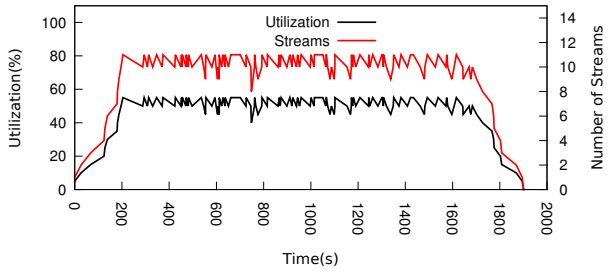


Fig. 6: Utilization And Number of Streams without Approximation, Edge and In-Transit Resources

Using the same setup, with availability of edge and in-transit resources, the number of streams being delivered over time has been measured. As shown in Figure 7, if data resolution requested for the streams goes down, the concurrent number of streams that can be processed and delivered increases. This 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.

In order to compare the baseline scenario with scenarios where edge, in-transit and approximation are available, we compare the acceptance ratio of different scenarios. Acceptance ratio is the percentage of the accepted requests to total number of requests. Figure 8 proves that using heterogeneous resources near data sources and using them to filter unwanted data increases the number of accepted requests and potentially users' satisfaction.

The resource utilization for the edge, in-transit and core resource for three different data resolutions have been

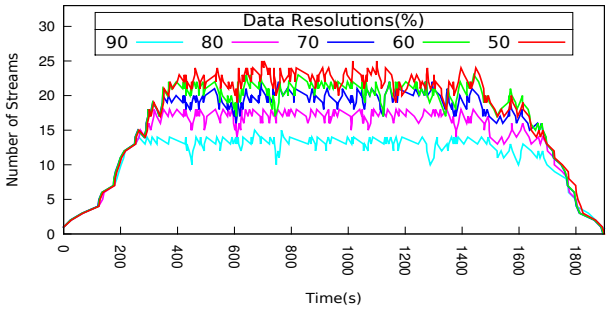


Fig. 7: Number of Streams Being Delivered to Users Over Time for Different Minimum Acceptable Data Resolutions.

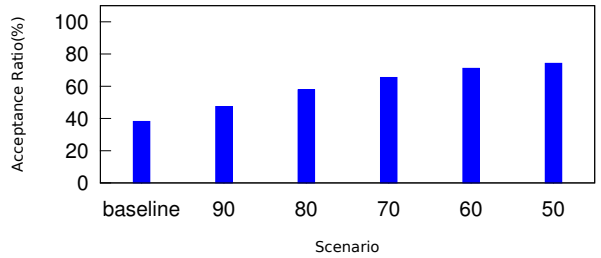
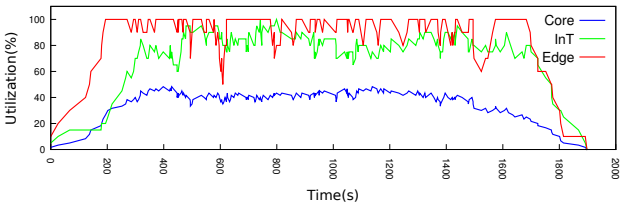


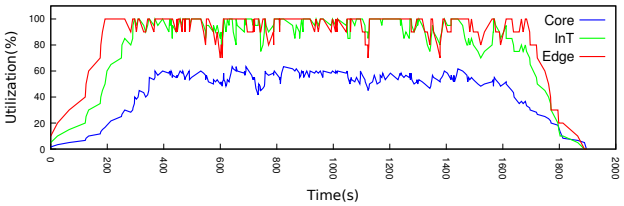
Fig. 8: Comparing Acceptance Ratio in Baseline and Edge/In-Transit Enabled Scenarios for Different Qualities.

shown in Figure 9. With 80% data resolution, utilization of core resource can reach 48% which is slightly less than our baseline due to edge and in-transit participation. However, for data resolution of 60% and 50%, 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 is filtered before it reaches bottleneck links and more data can be injected to the core resource which results in more utilization at the core.

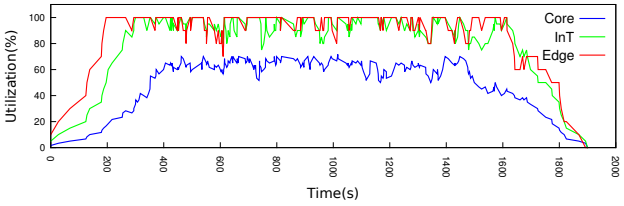
Next, we show how system dynamically adjusts the streams resolution when the execution/delivery environment is changed at runtime. We used 12 concurrent streams for three different resolutions (100, 80 and 60), 4 streams each. We use the same infrastructure depicted in Figure 5. After 220 seconds, the network bandwidth between In-transit and Core nodes cuts down to 200Mbit/s. Figures 10 shows the number of streams and the average resolution of the streams being delivered to the user for various request types. For high resolution requests (e.g. 100%) the streams are stopped and nothing will be delivered to corresponding users as 100% strict condition does not allow system to adjust the resolution and the system cannot do anything to recover delivery of the streams. It is also shown that the number of streams for 100% resolution decreases after the incident and reaches zero (note: resolution zero means no streams for such resolution requirement is being delivered). For 80% minimum resolution requirement, our system shows more resistance by first reducing the resolution and then stopping the



(a) 80% Data Resolution



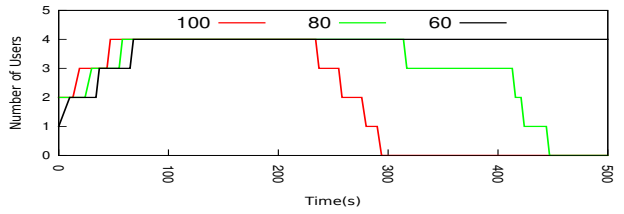
(b) 60% Data Resolution



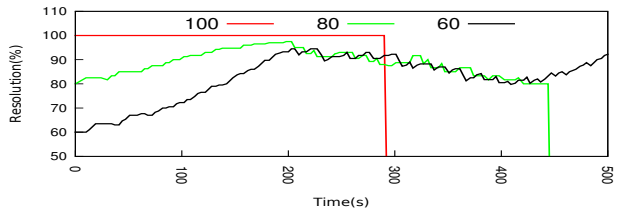
(c) 50% Data Resolution

Fig. 9: Utilization of Resources for Different Resolutions

streams. However, for 60% resolution, the system is able to continue flawlessly by reducing the quality of the streams for a while. It is interesting to note that the resolution of 60% requests increases once other streams (100% and 80%) being dropped and more network bandwidth gets available for remaining streams (60%). Hence, the system dynamically adjusts the resolution at runtime and overcomes the changes in execution space by reducing the resolution of the streams, if users are willing to sacrifice resolution.



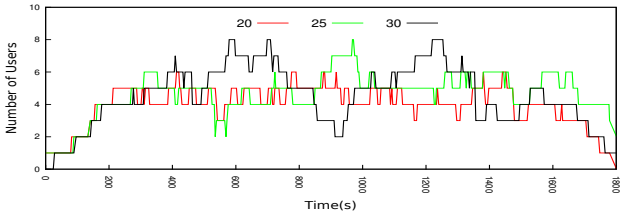
(a) Number of Streams



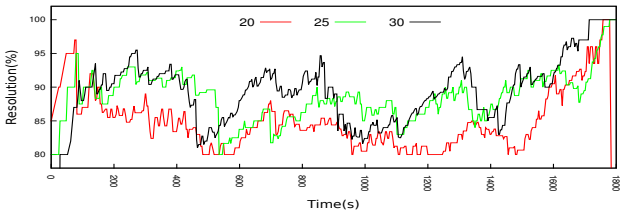
(b) Average Resolution

Fig. 10: Effect of Sudden Change in Bandwidth on Number of Streams and Average Resolution of the streams.

Finally, we studied the effect of different deadlines on the data resolution and number of accepted streams. We used similar setup as previous experiment and considered 80% as minimum acceptable resolution. The deadlines are 20, 25 and 30 seconds uniformly distributed across the requests. It is clear that 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 shows in Figure 11a) and data is being delivered with higher resolution for longer deadline requests (as shown in Figure 11b).



(a) Number of Streams



(b) Average Resolution

Fig. 11: Number of Users and Average Resolution at Runtime for Various Deadlines and Resolution of 80%

VI. Related Work

In order to process the stream of data, traditional approaches such as Apache Spark [15] and Apache Storm [16] are designed to process the data streams using the 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 [17]. Several researches have been proposed to resolve this issue by introducing the concept of edge computing and filtering the data at the edge of the network which have inspired us in this work. For instance, Heinz et al. [18] have proposed the idea of group aggregation to aggregate the data at the edge nodes and reduce the amount of network traffic. In another work, Santos et al. [19] have utilized edge resource to down sample the data at the edge resources. However, our proposed solution takes another step forward by taking advantage of the in-transit nodes [20], mapping the workflow stages on the available resource based on location and network conditions, and placing the workflow stages to minimize amount of the network traffic.

Data processing and delivery with end-to-end QoS constraints has been explored in different papers. Karim et al. [21] maps the user's QoS to the SaaS layer by developing hierarchial QoS model and assigning QoS weights.

In [22] authors 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. [20] 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 has been investigated in [23] that targets cost-time optimization techniques to schedule the workflows which is complementary to our work. Yu et al. [24] proposed a genetic algorithm to schedule the workflows under deadline and budget constraints. On other side, our proposed framework combines both of static and dynamic approaches to provide end-to-end QoS. Static approach is the workflow stage mapping to find best resources that meet deadline and budget, and dynamic approach by using feedback loops to check the execution of the streaming workflows at runtime. In another work, Heintz et al. [25] considered the trade-off between accuracy of the result and the time it takes to process the data, i.e., timeliness. In our framework, we consider this idea to 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 execution of each stage on edge and in-transit nodes.

VII. Conclusion and Future Work

Large scale observatories rely on the efficient processing and delivery of data generated from geographically distributed instruments and sensors. This paper introduces a subscription based data streaming framework and a runtime management system that provides QoS for the users and effectively utilizes heterogeneous geo-distributed resources to maintain application's quality of service. 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 and filters unwanted data going through the core resources. The evaluation showed that our system can increase main resources utilization by more than 10% using unwanted data filtering at edge/in-transit nodes for low resolution requests.

For future work, we will consider the concept of data merging across multiple requests and workflows in order to reduce amount of data going toward multiple users and eliminate redundant data transfer across multiple requests.

Acknowledgments : This research is supported in part by NSF via grants numbers OAC 1339036, OAC 1441376, OAC 1464317, OCE 1745246 and OAC 1640834.

References

[1] Jay Kreps et al. Kafka: A distributed messaging system for log processing. In *Proc. of the NetDB*, pages 1–7, 2011.

[2] Leslie M. Smith et al. The ocean observatories initiative. *Oceanography*, 31, March 2018. URL https://doi.org/10.5670/oceanog.2018.105.

[3] Manish Parashar, Moustafa AbdelBaky, Ivan Rodero, and Aditya Devarakonda. Cloud paradigms and practices for computational and data-enabled science and engineering. *Computing in Science & Engineering*, 15(4):10–18, 2013.

[4] Nitish Chopra and Sarbjeet Singh. Deadline and cost based workflow scheduling in hybrid cloud. In *Int. Conf. Advances in Computing, Communications and Informatics*, pages 840–846. IEEE, 2013.

[5] Stuart Mitchell. An introduction to pulp for python programmers. *Python Papers Monograph*, 1:14, 2009.

[6] Qishi Wu and Yi Gu. Performance analysis and optimization of linear workflows in heterogeneous network environments. In *Grid Computing*, pages 89–120. Springer, 2011.

[7] Ali Reza Zamani et al. Deadline constrained video analysis via in-transit computational environments. *IEEE Trans. Services Computing*, 2017.

[8] Ali Reza Zamani et al. A computational model to support in-network data analysis in federated ecosystems. *Future Generation Computer Systems*, 80:342–354, 2018.

[9] Ajay Tirumala, Tom Dunigan, and Les Cottrell. Measuring end-to-end bandwidth with iperf using web100. In *Presented at*, number SLAC-PUB-9733, 2003.

[10] Andreas Hanemann et al. Perfsonar: A service oriented architecture for multi-domain network monitoring. In *Int. Conf. Service-Oriented Computing*, pages 241–254. Springer, 2005.

[11] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conf. Computer Vision and Pattern Recognition*, volume 1, pages 886–893. IEEE, 2005.

[12] CLoudLab. https://www.cloudlab.us.

[13] HTB.https://linux.die.net/man/8/tc-htb.

[14] Amazon EC2 Prices.https://aws.amazon.com/ec2/pricing/.

[15] Matei Zaharia et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[16] Ankit Toshniwal et al. Storm@ twitter. In *Proc. of the 2014 ACM SIGMOD Int. Conf. on Management of data*, pages 147–156. ACM, 2014.

[17] Albert Jonathan et al. Nebula: Distributed edge cloud for data intensive computing. *IEEE Transactions on Parallel and Distributed Systems*, 28(11):3229–3242, 2017.

[18] Benjamin Heintz, Abhishek Chandra, and Ramesh K Sitaraman. Optimizing grouped aggregation in geo-distributed streaming analytics. In *Proc. of the 24th Int. Symposium on High-Performance Parallel and Distributed Computing*, pages 133–144. ACM, 2015.

[19] Ivo Santos, Marcel Tilly, Badrish Chandramouli, and Jonathan Goldstein. Dial: distributed streaming analytics anywhere, anytime. *Proc. of the VLDB Endowment*, 6(12):1386–1389, 2013.

[20] Virai Bhat, Manish Parashar, and Scott Klasky. Experiments with in-transit processing for data intensive grid workflows. In *Proc. of the 8th IEEE/ACM Int. Conf. on Grid Computing*, pages 193–200. IEEE Computer Society, 2007.

[21] Raed Karim, Chen Ding, and Ali Miri. An end-to-end qos mapping approach for cloud service selection. In *2013 IEEE Ninth World Congress on Services (SERVICES)*, pages 341–348. IEEE, 2013.

[22] Florian Rosenberg et al. An end-to-end approach for qos-aware service composition. In *Enterprise Distributed Object Computing Conf., 2009. EDOC'09. IEEE Int.*, pages 151–160. IEEE, 2009.

[23] Amandeep Verma and Sakshi 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)*, volume 4, pages 1–4. iRAFIT (7), 2012.

[24] Jia Yu and Rajkumar Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14(3-4):217–230, 2006.

[25] Benjamin Heintz, Abhishek Chandra, and Ramesh K Sitaraman. Trading timeliness and accuracy in geo-distributed streaming analytics. In *SoCC*, pages 361–373, 2016.