A Cloud-Agnostic Framework for Geo-Distributed Data-Intensive Applications

Fan Jiang
Department of Computer Science
University of North Carolina at Chapel Hill
dcvan@cs.unc.edu

Claris Castillo, Stan Ahalt
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
{claris, ahalt}@renci.org

***Abstract*—As the demand for Cloud computing trends up, valuable datasets are stored in the Cloud across various geographical regions and Cloud platforms and providers. The distribution of data across cloud providers imposes three major challenges for data-driven analysis and applications: the heterogeneity of cloud resources across clouds; low network throughput over the wide-area network; and high monetary cost resulting from moving data in/out cloud regions. In this work we propose a cloud-agnostic framework named *PIVOT* that builds on open-source technologies and abstraction principles to create the illusion of one single computer for applications and users. We have deployed a prototype across AWS and GCP and investigated its effectiveness against synthetic workloads. Using a combination of advanced middleware techniques and data-locality and cost aware scheduling strategies we show that *PIVOT* is able to achieve up to 4x improvement in network throughput and reduce $> 60\%$ monetary cost.**

I. Introduction

As the demand for cloud computing trends up, a number of cloud-based platforms have been developed in the industry and academia to address the increasing needs of data analysis and processing in the scientific and commercial sectors.

As a result, valuable datasets are stored in the Cloud across various geographical regions and cloud platforms. For instance, more recently the National Institute of Health (NIH) [16] established the Commons Cloud Pilot project to serve as a platform for technology experts to build the next generation cloud-based data sharing platform for the biomedical community. To enable this effort three large datasets of high scientific value were hosted in both clouds Amazon Web Services (AWS) and Google Cloud Platform (GCP), and made available to the participants of the consortium with the long term goal of making it available to the broader community. Similarly, the National Oceanic and Atmospheric Administration (NOAA) [17] provide access to real-time and archival weather radar data in AWS [18] for weather data analysis and prediction. Scientists however face steep challenges to be able to use this valuable datasets in ways that will enable new research and science collaborations.

The distribution of data across cloud providers imposes three major challenges for data-driven analysis and applications: **(C1)** The heterogeneity in resources, networking, application programming interfaces (APIs) and runtime among cloud platforms and providers prevents applications from scaling out across clouds, leveraging their unique capabilities and offerings (Google Genomics [10], SAGE [19]) and executing close to data. In other words, a user either runs an application on one cloud or the other but not across; **(C2)** Without the ability to scale across clouds in a seamless fashion, the applications trigger cross-region and cross-cloud data movements, which can hinder application performance due to low network throughput over the wide-area network (WAN). That is, to analyze combined data hosted in two clouds data must be moved out of one cloud into the other; **(C3)** Since commercial clouds monetize egress network traffic, *i.e.,* traffic result from moving data outside a cloud region within or across clouds, cross-cloud data analysis can incur prohibitive monetary cost for large datasets.

Therefore, minimizing the *financial* burden that hosting data and running computation in the cloud imposes on the user is critical to promote the adoption of cloud computing. As the name implies, egress network traffic cost is incurred when data needs to be transferred into a different cloud, a different region within the same cloud or on premise. Later we quantify the implications of making compute placement decisions that are oblivious to this cost. At the heart of our research work is the development of new scheduling strategies and techniques that factor egress network traffic cost in scenarios where data is distributed across multiple clouds and/or cloud regions.

Performance management is one other aspect that deter scientists from migrating their data-intensive applications into cloud environments. These applications are mainly data analytic workflows consisting of processes with temporal and software dependencies on each other, which execute on geo-distributed datasets and may produce new datasets at scale. Workflow engines such as Toil [39] and Arvados [3] and others [8] [24] are commonly used in the scientific community for data analysis. These solutions albeit providing predictable performance in controlled campus infrastructure have not been designed to perform well on geo-distributed environments; their scheduling capabilities are oblivious to both the network infrastructure that connect compute and storage resources as well as data locality.

Therefore, despite the existing efforts to host valuable data in the cloud, the lack of tools and mechanisms to support data analysis with reasonable performance levels along with the financial barrier resulting from moving data in and out the cloud hinders the ability of the scientific community to take full advantage of these efforts.

To address the aforementioned challenges we propose *PIVOT*, a cloud agnostic framework that creates an abstraction layer over compute and storage resources distributed across cloud providers to create the illusion of one very large computer thus hiding the complexity and heterogeneity of individual providers services and offerings. These resources are presented to users through an unified API via which data processing applications such as workflows can be executed and scaled across resources independently of where data is located. To achieve this, *PIVOT* decouples the abstraction and management of data and compute and builds on advanced middleware mechanisms that orchestrate how these resources are utilized *jointly* to provide applications with acceptable performance, while taking into account the financial cost on behalf of the users. A fined-grained resource model for cloud topology in combination with cost-aware scheduling algorithms allows *PIVOT* to place computation close to the data in order to minimize data movement and egress network traffic cost. We deployed an open-source based beta implementation of *PIVOT* across AWS and GCP and demonstrated its effectiveness through experiments with synthetic workloads. Our results show that by using its novel architecture and middleware mechanisms *PIVOT* can minimize egress network traffic cost ($> 60\%$) and improve network throughput up to a factor of 4x as compared to using traditional cost and data-locality oblivious scheduling strategies, respectively.

The key contributions of our work are manifold: (1) We have pioneered a *cloud-agnostic* framework and architecture that relies on two decoupled resource models for compute and data to create the illusion of one single large computer to users thus hiding the heterogeneity and complexity of cloud platforms. (2) We have developed middleware mechanisms that orchestrate these resources in a unified manner thus enabling the deployment of applications across multiple clouds without imposing additional burden on the user. (3) We have developed scheduling algorithms aware of cost and data locality that discern between cloud regions and providers to place applications close to the data thus minimizing data movement, improving performance and reducing financial cost. (4) An empirical evaluation of data-intensive applications in cross-cloud environments with respect to cost and throughput. To the best of our knowledge this is a first of a kind analysis on cross-cloud environments. (5) We have developed an open-source implementation of this framework that is available to the community to deploy their own software stack.

The rest of the paper is organized as follows. In Section II we provide an empirical analysis into the financial burden that results from placing computations and applications in cross-cloud environments. In Section III we outline the core architectural components and capabilities of *PIVOT*. An application model that leverages the systems capabilities follows in Section IV. Section V includes a deep architectural and functional view of *PIVOT* we outline the *PIVOT* core capabilities. In Section VI we provide an overview of the core open-source technologies that underpin our first implementation of *PIVOT* to provide early intuition on the fundamental capabilities that are needed to address the aforementioned challenges. We provide a deep evaluation of *PIVOT* in a real environment in Section VII. In Section VIII we provide an overview of research and technical contributions that aim at addressing the geo-distributed and cloud computing problem also considered in this work. We outline our next steps in extending *PIVOT* to support more complex applications in Section IX and conclude in Section X.

II. Motivation

To quantify the performance and monetary cost for running data-intensive applications across clouds, we show in Fig. 1 the proportions of different types of network traffic incurred on an Apache Mesos [2] cluster deployed across AWS and GCP. The network traffic is sampled over a total of $1,1500$ executions of data-intensive analytic jobs, which consume tens of gigabytes of data from geo-distributed data repositories in both clouds for processing. With the default scheduler, Mesos is oblivious to the location of data and therefore opportunistically assigns the jobs to resources regardless of the region and platform where they are provisioned. As a consequence, long-haul data transfers trigger by workflows leads to a huge amount of egress network traffic across regions and clouds. Quantitatively, roughly 90% of the network traffic travels across regions and clouds, incurring 98.5% of the total monetary cost for running the cluster. In contrast, only 1.5% of cost is incurred by compute resources. On the other hand, data transfers also result in low network throughput and create performance bottleneck as shortly disclosed in Section VII (Fig. 5). The challenges motivate us to develop *PIVOT*, to allow applications to scale out across geo-distributed regions in various clouds while retain data locality for efficiency in both performance and monetary cost.

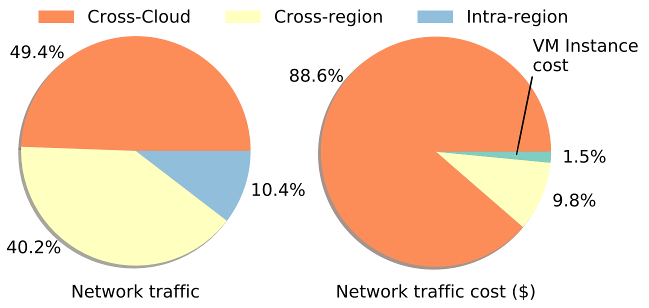


Fig. 1: Proportions of network traffic and monetary cost for running applications on a cross-cloud deployment of Apache Mesos cluster

III. *PIVOT* Architectural Overview

To provide the initial intuition about the system proposed, its functionality and construction, following we describe the core capabilities that underpin *PIVOT*.

In the previous section we identified three key challenges that users/applications face to harness cloud capabilities across the multiple cloud platforms, namely **(C1)-(C3)**. Following, we describe the three core capabilities that build on each other to address these challenges. When incarnated into *PIVOT*, these capabilities provide a novel and effective solution to

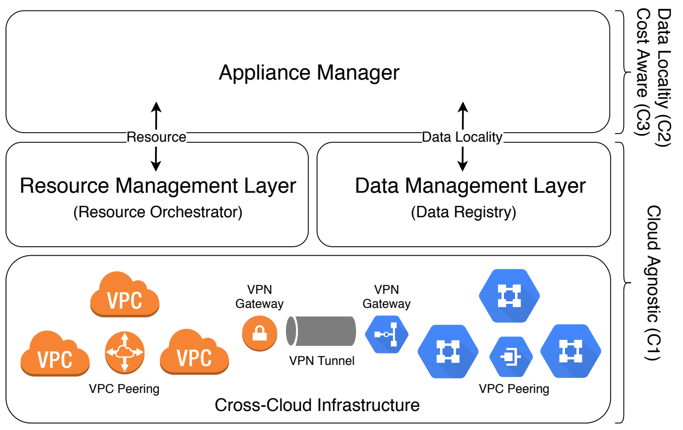


Fig. 2: PIVOT architecture

executing applications on a cross-cloud environment. Fig. 2 depicts a high level view of the *PIVOT* architecture. These capabilities can be organized into three core capabilities:

Unification via resource abstraction
For applications such as workflows to *seamlessly* utilize disparate cloud resources across multiple providers (**C1**), resources must be presented through a *unified* interface that coalesces these resources into a *cloud-agnostic* environment.

To achieve this *PIVOT abstracts* both data and compute infrastructure through widely used open-source technologies; and, relies on novel middleware techniques to orchestrate how these resources are managed to support the execution of applications across clouds.

We borrow widely adopted technologies developed for much simpler environments to implement our approach. As such, these technologies alone seldom address the three challenges we identified present in cross-cloud environments. To address (**C2-C3**), we have integrated these technologies with novel add-on middleware capabilities described as follows.

Cross-cloud data locality awareness
As described in Section I multi-cloud environments present unique obstacles that hinder the use of performance optimization strategies commonly used in large-scale distributed systems [26] [25]. For instance, with data analysis applications combining data hosted across multiple clouds, moving data is inevitable. A system capable of supporting such applications should take into consideration properties about the data including its location. Nevertheless, cross-cloud infrastructure is not only heterogeneous but also *opaque* therefore limiting cloud-based systems ability to exploit the physical location of data when making placement decisions.

To take into account the network performance degradation resulting from moving data across cloud platforms, *PIVOT* relies on a resource model that captures data locality attributes and scheduling capabilities that use these attributes to minimize data movement when possible. Thus, *PIVOT* is able to distinguish datasets by the regions and cloud platforms where they are hosted and place applications accordingly, hence addressing **C2**.

Cost-aware scheduling
In this work we are concerned with developing cloud-based systems that can support big data analysis in a cost-effective manner (**C3**). To start, in *PIVOT* we first consider *egress network traffic cost*. Recall that in Section II we quantified the implications of making decisions that are oblivious to this cost.

PIVOT leverages the resource model offered by the cloud-agnostic framework and the data-locality artifacts described earlier in combination with a generic, pluggable scheduling framework that is customized to take into account the egress network traffic cost when making placement decisions. As we will demonstrate via real experiments, our approach is effective in reducing the monetary cost of running applications in cross-cloud environments. Furthermore, we show that a fine-grained resource model that distinguishes between cloud regions and platforms is needed to effectively reduce cost.

IV. *PIVOT* – Application Model

In order to leverage the abstractions, resource models and the overall architecture in *PIVOT* we introduce an application abstraction, namely *appliance*. An appliance is a software application with enough operating system to run optimally in standard compute environments. In *PIVOT* an appliance consists of a collection of inter-operating containerized *applications* and corresponding configurations.

```
1  id: sample-appliance
2  containers:
3    - id: app1
4      type: job
5      image: pivot/workflow
6      cmd: /bin/run_workflow
7      resources:
8        cpus: 8
9        mem: 10240
10       disk: 10240
11       gpu: 0
12     dependencies:
13       - app0
14     data:
15       - dataset.0
16       - dataset.1
17   - id: app0
18     type: service
19         ...
20 scheduler:
21   name: locality-aware
22   config:
23     scalable: true
```

Listing 1: Sample appliance specification

PIVOT supports two types of applications: (1) A *service*, which is a long-running process with a specific functionality, *e.g.,* web server; and, (2) A *job*, which is a process or set of processes typically associated with a bounded temporal window, *e.g.,* `map` and `reduce` jobs. For each application, a set attributes are specified as shown in Listing 1 including resource demand (Line 7−11) and data locality (Line 14−16), which is used by *PIVOT* to schedule applications. We will introduce scheduling in *PIVOT* shortly in Section V-5.

We note that by combining these two types of applications, *PIVOT* can support a broad range of computing platforms consisting of jobs and services with complex dependencies. A computing platform may consist of a single containerized tool to complex and sophisticated software stacks. Notably, instead of handling services and jobs separately, *PIVOT* allows them to co-exist and interoperate with each other within a single appliance as exemplified in Fig. 3. In Section IX we provide examples of other software stacks that can be deployed on *PIVOT* thus demonstrating its extensibility and generality.

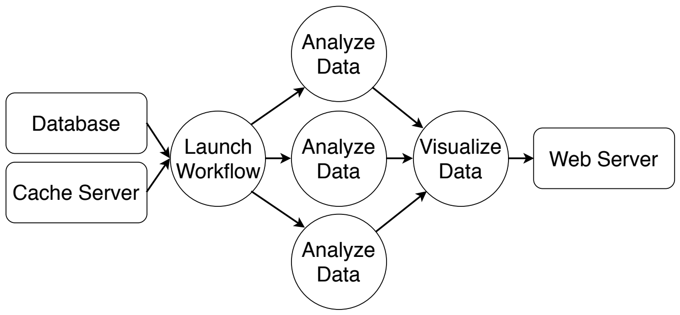


Fig. 3: An example of appliance: the workflow launcher waits on the database and cache server up and running before starting the workflow; the workflow spawns a number of parallel data analysis jobs that generates output data; the data visualization job visualizes the output data in observation that all the job analysis jobs are done; finally, the web server is started to render the visual after the data visualization job succeeds.

Finally, *PIVOT* allows users to use customized appliance-level scheduler for each appliance (Line 20−23 in Listing 1) to achieve application-driven scheduling goals. The schedulers are installed as *plug-ins* as will be introduced shortly in Section V-4. Besides, *PIVOT* also allows users to provide custom scheduling configuration in the appliance specification to facilitate the custom appliance-level scheduling at a finer granularity. For instance, as illustrated in Listing 1, the `locality-aware` scheduler takes a binary value `scalable` as the configuration, to determine whether to scale out containers to other cloud regions/platforms in observation of resource shortage at the optimal one.

V. System model

PIVOT consists of two major components as illustrated in Fig. 2: the cloud-agnostic framework and the appliance manager.

1) Cloud-agnostic Framework: The cloud-agnostic framework is underpinned by abstraction mechanisms that hide and unify the heterogeneity underneath multi-cloud environments. It can be further organized into: (1) data management layer and (2) resource management layer. As we describe later in this section, *PIVOT* takes an engineering approach to networking.

The cross-cloud infrastructure is built on top of computing, storage and network resources provisioned in multiple geo-distributed regions on multiple cloud platforms. Fig. 4 depicts the geographic footprint of a common setup that we will use to demonstrate the effectiveness of *PIVOT*.

2) Resource Management Layer: The resource management layer abstracts the geo-distributed, heterogeneous and dynamic resources provisioned in the cross-cloud infrastructure into a single cloud-agnostic resource pool, where resources are subscribed on-demand at fined granularity, *e.g.,* 1 CPU and 2GB memory. In this layer, container technology offers the ability to instantiate consistent and isolated runtime environments for applications regardless of the underlying resource infrastructure underneath. Hence, containerized applications can easily migrate among heterogeneous resources in the cross-cloud infrastructure and provide consistent functionality, decoupled from specific cloud vendors. The resource manager coordinates the allocation of resources among applications with diverse resource demands, and also provides control knobs for application-driven management. Via these control knobs, the resource manager delegates high-level resource scheduling decisions to the appliance manager (described later in this section). A few generic resource managers such as Kubernetes [13] and Apache Mesos [2] provide this functionality out-of-the-box and therefore can be utilized to implement this component. Our current implementation of the resource manager uses Apache Mesos, which is introduced in more detail in Section VI-A.

3) Data Management Layer: To unify access to data generated and accessed by applications across disparate heterogeneous cloud resources, we introduce a data abstraction layer that maintains a logical view of the data storage infrastructure. Through this logical view applications can access and process data independently of where the data is physically located. At the heart of this component is a *data registry* which allows applications to register metadata associated with data objects accessible in *PIVOT*. Metadata contains information about data objects such as size, physical location, owner, among others, hence has the potential of enhancing data awareness in the system. Additionally, the data registry provides a set of APIs for creating, updating and retrieving metadata entries through controlled access mechanisms. In *PIVOT* we use iRODS [12] to implement this component. Refer to Section VI-B for an overview of this technology. The functionality can also be implemented using distributed data storage such as Ceph [5] and GlusterFS [9] with an additional data registration module enabled.

4) Appliance Manager: The appliance manager serves as an interface for users to create and interact with appliances (Section IV) in *PIVOT*. Intuitively, this component creates the illusion of a single monolithic server with the resource and data abstraction provided by the cloud-agnostic framework. Internally, it makes application placement and resource allocation decisions for every appliance. The appliance manager consists of a two-level configurable scheduler: a system-level scheduler which aims at global optimum; and an appliance-level scheduler which optimizes for application-driven goals. In our current implementation, *PIVOT* delegate the system-level scheduling tasks to the default scheduler of the *resource orchestrator*, *i.e.,* Apache Mesos, which adopts an opportunistic strategy for application placement. For the appliance-level scheduler, we have developed a pluggable scheduling framework in which schedulers are developed and installed as *plug-ins*. To enable the pluggable scheduling framework, *PIVOT* decouples the scheduler from other modules in the system and provides a set of APIs for the scheduler to schedule containers in an appliance. With the scheduling APIs, the scheduler is able to control the (re)placement of each container throughout its life cycle and orchestrates among containers with respect to the dependencies. In order to make scheduling decisions that take into account data locality, the appliance-level scheduler obtains information about the availability of resources and location of data from the *resource orchestrator* and *data registry*, respectively.

To run an appliance on *PIVOT*, a user submits an appliance request (Listing 1 as an example) to the appliance manager. The appliance manager launches an instance of the appliance-level scheduler specified in the request. The dependencies included in the request are represented as a directed acyclic graph (DAG) which is later utilized to drive the scheduling plan.

With help of the resource and the data management layer, the appliance-level scheduler identifies the resources that meet the resource demand requirements and are available to host the application while respecting its data-locality requirements. The appliance-level scheduler then generates a scheduling plan with the resource allocations for the scheduled applications, and sends it to the system-level scheduler to verify compliance with global scheduling policies. If compliant, the scheduling plan is converted into a number of resource requests submitted to the resource manager for execution. The resource manager launches the applications on the specific resources as containers following the resource allocation determined by the application-level scheduler.

5) Locality-aware scheduling: The goal of locality-aware scheduling is to place applications close to their input data to minimize cross-region and cross-cloud data transfers, which result in low network throughput and high egress network traffic cost. Optimally, every application should run on the same region hosting its input datasets. As shown in Listing 1, users provide unique identifiers, *e.g.,* UUID, to identify datasets that are needed by specific applications. The data management layer relies on the data registry to validate and map these identifiers to physical locations in the cross-cloud infrastructure, *i.e.,* cloud regions and platforms. Two situations arise when scheduling applications: 1) the datasets required by an application is distributed across multiple cloud regions and platforms; 2) there is resource shortage in the region hosting the input datasets.

To address the first challenge, the scheduler calculates the monetary cost for every possible placement and selects the one incurring the least cost using the cost function below:

$$C(p) = \sum_{r \in R} M_{r,p} \cdot d_p$$

In this function, p and R denote a potential placement (*i.e.,* cloud region) of application and the set of cloud regions hosting the data, respectively. M is a cost matrix that includes unit monetary cost for network traffic between every pair of cloud regions. This matrix is fixed and obtained from the cloud providers. For example, Table I includes the prices charged by AWS and GCP for traffic transferred between cloud regions and platforms. d_t denotes the total amount of egress network traffic which is inferred from the data size. The function calculates a total cost for egress network traffic if the application is placed at p.

To address the second problem, *PIVOT* allows users to opt in/out of scaling appliances across cloud regions (`scalable` option in Listing 1). When a user opts-out, applications in the appliance may have to wait for resources to become available before being provisioned. Hence, this strategy trades egress network traffic cost for waiting time (and overall wall time) of the application. Alternatively, when a user opts-in, the scheduler invokes the cost function to find the placement that incurs the second least egress network traffic cost, and the applications may be provisioned immediately, albeit in a different region.

One variant of the vanilla locality-aware scheduling strategy relaxes the constraint of co-locating applications and their input data. Under this strategy, the scheduler places applications on regions that are nearby the region hosting the data and hence trades a small monetary cost in egress network traffic for increased resource availability. To determine candidate regions, this strategy uses the same function described earlier in this section and consider regions in an increasing order of cost.

	AWS	GCP
Intra-region	0	0
Cross-region	0.018	0.010
Cross-cloud	0.090	0.120

TABLE I: Average monetary cost ($/GB) for different types of network traffic among North America regions in AWS and GCP

Network infrastructure in *PIVOT*. To construct the cross-Cloud network infrastructure, we face multiple challenges. To start, there are a multitude of heterogeneous, isolated networks to be bridged and administered in order to enable network connectivity among the geo-distributed resources. Most cloud platforms allocate dynamic public IP addresses to resources, namely *floating IP allocation*, further complicating network management. To address the challenge, we establish a unified virtual network over the geo-distributed resources across platforms. The virtual network consists of subnetworks in every cloud region. To bridge regional subnetworks within the same cloud, we leverage the *network peering* (or equivalent) provided by most cloud providers, which creates optimized network routes among the subnetworks. To bridge subnetworks across Clouds, we adopt virtual private network (VPN) tunneling to enable the communication over the Internet due to its simplicity and automatability in deployment and maintenance. We notice the alternative to bridge cross-Cloud subnetworks is to reserve dedicated network circuits, which improves network efficiency but incurs IT efforts, and consider it as an ideal option for long-term, large-scale *PIVOT* deployment.

VI. Underpinning Technologies

In this section, we provide the reasoning before our choices of technology to build the software stack of *PIVOT*. We note that the architecture of *PIVOT* (Section V) by design is generic and extensible and therefore these technologies can be substituted by others with similar capabilities.

A. Mesos

We use Apache Mesos for resource orchestration across regions and cloud platforms due to its capability to abstract heterogeneous, distributed resources. It relies on the container technology, specifically Docker [7], to instantiate consistent, isolated runtime for various applications across heterogeneous resources, eliminating their dependencies on resources and therefore allowing them to seamlessly migrate heterogeneous environments. Mesos supports fine-grain sharing of resources and multi-tenancy. It allows resources on a single host to be shared among multiple different applications in quantifiable portions, *e.g.,* 1 CPU, 2GB memory and 2GB disk space. Mesos adopts a master/slave model, in which the masters coordinates the resource allocations among the applications, and the slaves fulfill the allocations and execute the actual processes of the applications. Additionally, Mesos also enables a constraint-based mechanism for controlling application placement, which can be leveraged for developing advanced scheduling algorithm on top. Because of these properties, we select Mesos as the resource orchestration subsystem for *PIVOT* due to its capability in resource abstraction and fine granularity in resource subscription. We also find Kubernetes provides a similar set of functionalities, which make it an ideal replacement for Mesos in *PIVOT*. However, deploying Mesos or Kubernetes out-of-the-box in a geo-distributed, cross-cloud environment is still experimental and yet to be proven effective [6] [14].

B. iRODS

We introduce the integrated rule-oriented data system (iRODS) [12] into *PIVOT* to function as an abstract data layer over distributed data stored in heterogeneous storage systems. The iRODS federates distributed and heterogeneous data into a single logical file system and provides a modular interface to integrate new client-side applications and server-side data and compute resources. It also acts as a third-party mediator providing authentication, authorization and auditing, optimized data movement protocols and rich support for metadata at multiple levels of data collections. The iRODS runs a catalog service that provides APIs for registering and querying metadata of disparate distributed datasets in a unified logical namespace, effectively serving an abstract layer for data management in *PIVOT*. The iRODS is an open-source software that serves a broad community including NASA [15], Bayer [4], Wellcome Trust Sanger Institute [20], the NOAA among others. We also note that distributed storage systems such as Ceph and GlusterFS can provide the similar capabilities with additional support in metadata operations.

VII. Evaluation

To evaluate the effectiveness of our approach we have deployed a prototype implementation of *PIVOT* on AWS and GCP. We use synthetic workloads that are representative of generic data-intensive applications which would benefit from data-locality aware scheduling in cross-cloud environments. We note that non-data-intensive applications are not negatively impacted by data-locality aware scheduling.

A. Experiment Setup

1) System prototype: The prototype is built on top of Apache Mesos and iRODS for resource and data management, respectively. The appliance manager is developed using Python 3. The prototype consists of 20 virtual machine (VM) instances provisioned in ten geographical cloud regions in North America across AWS and GCP (Fig. 4) with `m5.xlarge` (4 cores, 16GB memory) and `n1-standard-4` (4 cores, 15GB memory) instances, respectively. The VM instances support up to 10Gbps network throughput. Each VM is configured with the Mesos slave and iRODS resource daemons to abstract compute and data. Additional VMs are set up for the Mesos master, the iRODS catalog server and the appliance manager.

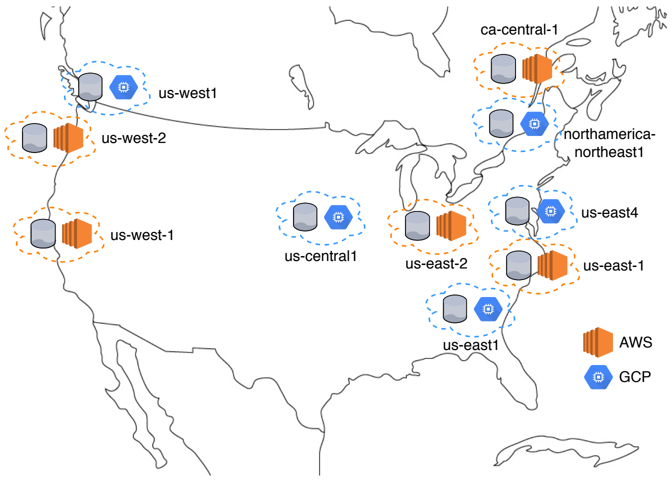


Fig. 4: *PIVOT* deployment across ten North America regions in AWS and GCP.

2) Scheduling strategies: In our evaluation we consider three scheduling strategies as follows:

- `Locality-oblivious` places applications onto any node with sufficient resources available in an opportunistic manner, regardless of data locality of datasets used by the applications.
- `Locality-aware` places applications *only* on resources that are hosted on the same region than the input data. Under this strategy, applications may be queued for an indefinite amount of time waiting for resources to become available. See Section V-5.
- `Locality-aware-scale` allows applications to be placed on non-optimal regions in a *best-effort* manner.

That is, this strategy considers placing applications on remote regions while aiming to minimize egress cost. See Section V-5.

3) Experimental workload: We have synthesized experimental workflows as appliances consisting of *data generation* and *data processing* jobs. The workflow starts with a number of data generation jobs running at each cloud region, each of which generates and registers a synthetic dataset into the data registry. The size of datasets follows a uniform distribution ranging between 500MB−2GB. Following, the workflow launches the data processing jobs in parallel, which ingest and emulate the processing of the synthetic datasets. The parallelism level of the jobs is configurable and allows us to evaluate the system under varying loads; the level of parallelism ranges between 1 and 40 jobs in these experiments.

4) Performance metrics: We focus on the following performance metrics for evaluating the performance of the scheduling strategies in *PIVOT*:

- **Network throughput** measures the amount of data transferred from/to a job in a unit of time (per second). Considering that the time for network I/O is commonly dominant in the runtime of data-intensive jobs, high network throughput can result in noticeable runtime reduction for *individual* jobs.
- **Monetary cost** captures the monetary cost incurred by utilizing various resources in the cloud, including compute, storage and network resources. In our experiments we measure the monetary cost by taking an average of the total cost over the amount of data being transferred (in gigabytes).
- **Walltime** measures the time between the starting and ending time of the appliance. In the case of workflows, it refers to the duration between the starting time and ending time of the first and last job, respectively.

B. Results

To evaluate the effectiveness of *PIVOT* in balancing monetary cost for egress network traffic and network performance in Fig. 5 we show the average cost for network traffic incurred by the experimental jobs with both locality-aware and locality-oblivious scheduling. As it is observed, the use of locality-aware scheduling strategy saves 67.6% of the cost for network traffic per GB as compared to the locality-oblivious scheduling strategy. To better understand these results, Fig. 6) depicts how different network traffic types compare for both strategies. We consider three types of network traffic: cross-cloud, cross-region and intra-region traffic corresponding to traffic across different cloud providers, different regions within the same cloud; and, within the same region, respectively. Our experiments show that when using the locality-aware scheduling strategy 58.9% of network traffic remains within the same cloud region, while only $< 40\%$ travels across regions and cloud providers –and hence incurs additional monetary cost. This is in contrast to when the locality-oblivious scheduling strategy is used; in which case, 90% of the overall network traffic flows between cloud regions and cloud providers. These results follow intuition since the locality-aware scheduler in *PIVOT* leverages data locality to optimally place jobs and minimize cost for network traffic. Since cloud providers typically do not charge for intra-region traffic (Table I), the locality-aware scheduler favors placing jobs within the same region hosting the input data provided that there are resources available to meet the requirements of the application. In contrast, under the locality-oblivious scheduling strategy jobs can be placed onto resources in cloud regions that are distinct from the region hosting the input data. As a result, this strategy causes unnecessary network traffic and results in prohibitive financial cost to the users.

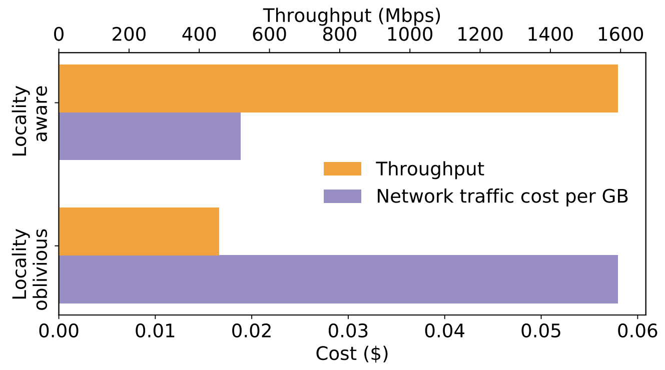


Fig. 5: Average throughput and monetary cost with locality-oblivious and -aware scheduling

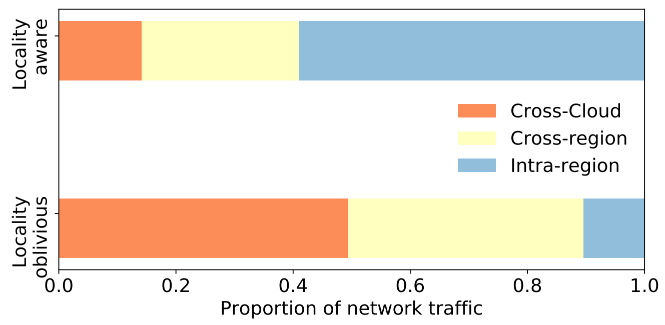


Fig. 6: Proportions of network traffic with locality-oblivious and -aware scheduling

Furthermore, Fig. 5 shows that the `locality-aware` strategy achieves 4x improvement on average throughput as compared to the locality-oblivious scheduling strategy. This is because cloud network infrastructure is optimized for throughput and latency while WANs connecting cloud regions and providers are typically outside the cloud provider's control. Therefore, by optimizing for cost, the locality-aware scheduler ensures that network flows remain short and within a region and exhibit high throughput. To gain further insight into this observation, Fig. 7 depicts the negative correlation between egress network traffic cost and throughput in our experimental setup.

To investigate the impact that scaling an application across cloud-regions would have on the application performance and

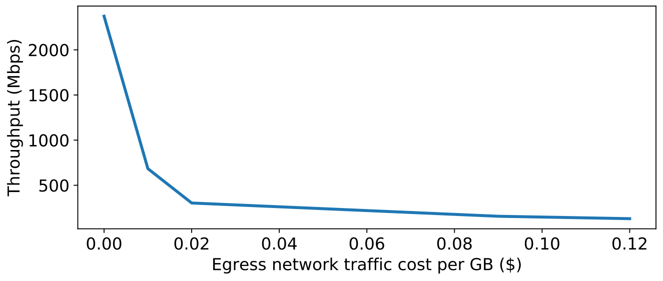


Fig. 7: Negative correlation between throughput and network traffic cost

the egress network traffic cost in lieu of *waiting* for resources to become available in the optimal region we perform a comparative experiment of the three scheduling strategies applied to a job with various levels of concurrency.

Fig. 8 shows the results of these experiment. We observe that provided with an infinite pool of resources to choose from the locality-oblivious strategy allows concurrent jobs to benefit from scaling out at the expense of high egress network traffic cost. As expected, the locality oblivious strategy exhibits the highest egress network traffic cost among the three strategies. The locality-aware strategy on the other hand offers limited choices for scheduling and therefore the job cannot rip the benefits of its parallelism. As the figure shows, in this case jobs queued up due to lack of resources available in the *optimal* cloud region and the job experiences the highest walltime as compared to the other two strategies. Finally, the locality-aware scale strategy strikes a balance between egress network traffic cost and the ability of the application to take advantage of concurrency by being presented with more resources to utilize for scaling the application while being sensitive to egress network traffic cost. As the figure shows, this strategy exhibits the lowest walltime and a low egress network traffic cost up to an inflection point when the strategy maxes out

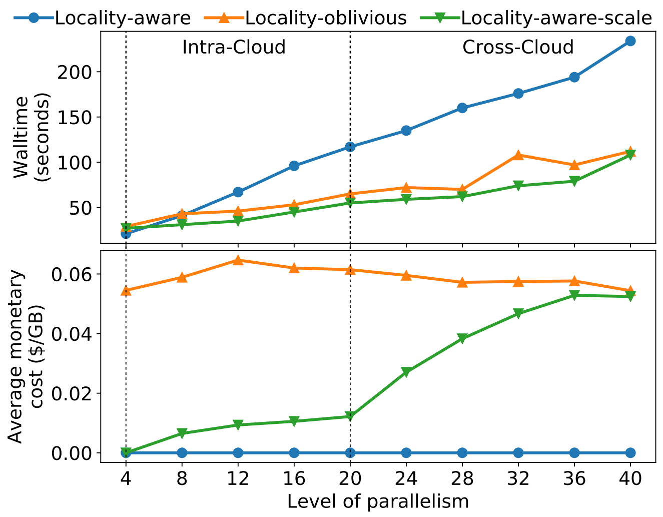


Fig. 8: Comparison of end-to-end runtime and network traffic cost with different scheduling strategies as job concurrency increases

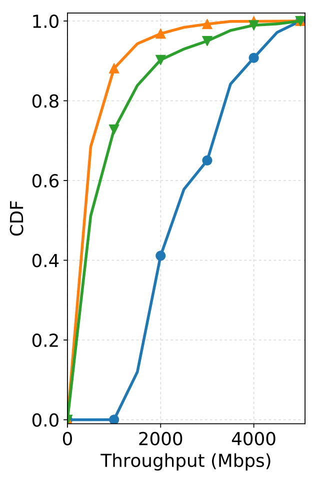
its ability to leverage resources across clouds incurring data movement and higher egress network traffic cost. Therefore, the locality-aware scale strategy is a cost-effective scheduling strategy .

To gain further insight into the effectiveness of the scheduling strategies in Fig. 9a and Fig. 9b we show the cumulative distribution function for throughput and number of running concurrent tasks per appliance, respectively. We note that in the case of the locality-aware scheduling strategy in more than 80% of the time the number of concurrent running jobs is $<20\%$ more as a result of queuing jobs in the local cloud provider. Both the locality-aware-scale and `locality-oblivious` strategy however show an even distribution of concurrent jobs over time as seen in Fig. 9b. As expected, throughput reduces due to the cross-cloud region network traffic resulting from running computation across the cloud regions.

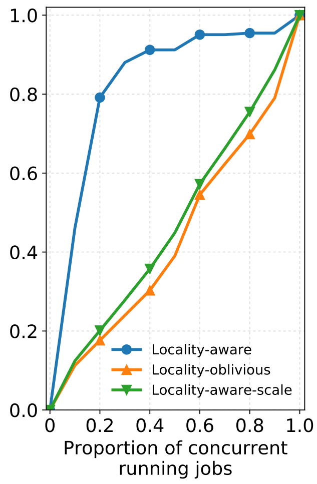
We note that the locality-aware scale strategy yields the highest throughput at a negligible egress network traffic cost. However, due to the fact that the `locality-aware` scheduling strategy is limited to place jobs on resources that are in the same region hosting the input data as depicted in Fig. 9b, $>70\%$ of jobs are queued-up during 80% of the runtime of the appliances. In contrast, the `locality-oblivious` scheduler is able to scale jobs out to utilize idle resources in other cloud regions and take advantage of parallelizing compute at the expense of a high egress network traffic cost as compared to the other the scheduling strategies due to is inability to take into account data locality information. We note that the `locality-aware-scale` scheduling strategy strikes a balance between data locality and job scalability by trading data-locality for walltime. As a result, it represents a cost-effective scheduling strategy as compared to `locality-aware` and `locality-oblivious` strategies in circumstances where cloud resources are limited – which is common in practice.

VIII. Related Work

Geo-distributed computation. A number of recent research works identify the great demand for running applications at scale in geo-distributed environments [23] [33] [27] and seek to address a variety of challenges in these environments [28] [35] [38] [32]. It has been recognized as a major challenge [35] [40] [41] that substantial data transfers incurred among applications running on geo-distributed resources create the performance bottleneck due to the inefficiency of data transfers over WANs. JetStream [35] proposes to perform data pre-processing on data sources and send only partially aggregated data for central processing, reducing the amount of network traffic incurred over WANs. However, this work is limited to applications dominated by data aggregation processes. Other research efforts [34] [29] [32] [22] [30] tackle the problem by exploiting data locality and co-locating data and applications to avoid remote data transfers over WANs. Iridium [34] introduces an online heuristic that approximates the optimal co-location of data analytic jobs and



(a)



(b)

Fig. 9: Throughput and proportion of concurrently running jobs with different scheduling strategies

datasets, effectively shortening the response time for data analytic queries upon geo-distributed datasets. DRASH [22] and Cachalot [31] dynamically replicate and cache commonly used datasets across geo-distributed resources to enhance data locality. AWAN [32] takes advantage of explicit job runtime and data placement provided by users to optimize data locality for jobs executed on geo-distributed resources, effectively minimizing average job completion time. However, these works mostly focus on the performance aspect of geo-distributed applications but seldom the cost aspect, which is equally crucial in cloud-based, geo-distributed environments especially with resources provisioned and budgeted by multiple cloud vendors.

Cloud agnosticism. Only few works in research literature explore the possibility to build cross-cloud, geo-distributed systems to enable cloud agnosticism. In [26] the authors recognize the necessity of developing multi-cloud systems and classify a stack of cloud solutions for easing the heterogeneity and lack of interoperability among cloud platforms. [36] [37] introduce multi-cloud storage systems that secure outsourced data in the Cloud with bandwidth and cost efficiency. Nevertheless, these solutions address the problem in a piecemeal fashion. This is in contrast to our holistic approach in *PIVOT* wherein an integrated and end-to-end solution has been developed to address the problem in a much more comprehensive and pragmatic manner. Container orchestration systems, such as Apache Mesos and Kubernetes, exhibit a great potential to serve as an resource abstraction layer over resources geo-distributed regions and cloud platforms to support a diversity of applications. However, they lack the capability for enabling data awareness for the system and applications running on top, which are proven essential in geo-distributed environment.

IX. Future Applications in *PIVOT*

Designed as a generic computing platform, *PIVOT* imposes minimal restrictions on the composition and scheduling of appliances. We promote customizable, pluggable scheduler in *PIVOT* to satisfy varying scheduling demands. Moreover, existing computing systems and framework (*e.g.,* Apache Spark) can be easily refactored into containers and run as an appliance on *PIVOT*. Therefore, a variety of legacy applications can be seamlessly ported onto *PIVOT* without any changes. Following that we present two real-world use cases of running existing data-intensive applications across clouds on *PIVOT*.

A. Use Case 1: Apache Spark appliance

A use case of *PIVOT* is to create a workspace for scientists to run large-scale genomic analytic applications using the geo-distributed resources. With most of the applications developed using the API of Apache Spark, it requires substantial code changes to convert them into appliances directly running on *PIVOT*. Instead, we run the Spark cluster as an appliance with a number of containerized services. In this case, *PIVOT* only allocates resources to the services per request but delegates the task scheduling to the framework, *i.e.,* the scheduler of Apache Spark. To instantiate the Spark appliance, *PIVOT* is able to co-locate the services (*e.g., workers*) with the data repository based on the logical identifier of data specified in the appliance request. Within the appliance, the existing applications can be executed by interfacing with the Spark appliance without any code changes; legacy scheduling algorithms can also be applied directly in the appliance. Moreover, the Spark appliance can still take advantage of the data-locality facilities in *PIVOT* to enhance the scheduling algorithm – the Spark scheduler can query the data registry for metadata of datasets and make scheduling decisions accordingly. This approach for migrating existing applications onto *PIVOT* can be extrapolated to computing frameworks such as Hadoop [1], Mesos and HTCondor [11] among many others.

B. Use Case 2: Cross-cloud execution of genomic workflow

We are also developing appliances for enabling cross-cloud execution of genomic workflows, *e.g.,* exomic alignment workflow, in *PIVOT*. The genomic workflows typically ingest terabytes of geo-distributed datasets hosted on various cloud platforms, which are intrinsically difficult to be moved around for centralized processing. Besides, many existing genomic workflows are encoded in Common Workflow Language (CWL) [21] and runnable on the Toil workflow engine. To leverage the locality-aware scheduling in *PIVOT*, we integrate *PIVOT* as the batch system at the backend of Toil for scheduling and executing the workflow jobs. With the genomic datasets registered in *PIVOT* and referenced in the workflow specifications, *PIVOT* is able to scale the workflows across geo-distributed regions across clouds and place the jobs close to the input datasets for improved performance in network I/O and reduced cost for egress network traffic.

X. Conclusion

We have presented *PIVOT*, a generic geo-distributed, cross-cloud computing platform for running data-intensive applications with awareness of data locality. We have designed a loosely coupled architecture across cloud platforms to enable cloud agnosticism. Moreover, we have developed a locality-aware scheduling algorithm to improve data locality of applications, therefore improving network performance of data transfers and reducing monetary cost for egress network traffic. Our evaluation demonstrates that *PIVOT* is able to achieve up to 4x improvement in network throughput and $> 60\%$ saving in the cost for egress network traffic as compared to the baseline.

ACKNOWLEDGMENT

This work is supported by the NSF SciDAS project (ACI #1659300) and the NIH Data Commons Cloud Pilot Phase 1 (OT3 ODO25464-01).

REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org/.

[2] Apache Mesos. http://mesos.apache.org/.

[3] Arvados. https://arvados.org/.

[4] Bayer. https://www.bayer.com/.

[5] Ceph. https://ceph.com/.

[6] DC/OS HA. https://docs.mesosphere.com/1.10/installing/oss/high-availability/multi-region/.

[7] Docker. https://www.docker.com/.

[8] Galaxy. https://usegalaxy.org/.

[9] GlusterFS. https://redhatstorage.redhat.com/products/glusterfs/.

[10] Google Genomics. https://cloud.google.com/genomics/.

[11] HTCondor. https://research.cs.wisc.edu/htcondor/index.html.

[12] iRODS. https://irods.org/.

[13] Kubernetes. https://kubernetes.io/.

[14] Kubernetes Federation. https://kubernetes.io/docs/concepts/cluster-administration/federation/.

[15] NASA. https://www.nasa.gov/.

[16] National Institutes of Health. https://www.nih.gov/.

[17] National oceanic and atmospheric administration. http://www.noaa.gov/.

[18] Registry of Open Data on AWS. https://registry.opendata.aws/noaa-nexrad/.

[19] Social Alteration Google Earth (SAGE). http://socialalterations.com/googleearth/.

[20] Wellcome Trust Sanger Institute. https://www.sanger.ac.uk/.

[21] Peter Amstutz, Robin Andeer, Brad Chapman, John Chilton, Michael R Crusoe, Roman Valls Guimera, Guillermo Carrasco Hernandez, Sinisa Ivkovic, Andrey Kartashov, John Kern, et al. Common workflow language, draft 3. 2016.

[22] M. W. Convolbo, J. Chou, S. Lu, and Y. C. Chung. DRASH: A Data Replication-Aware Scheduler in Geo-Distributed Data Centers. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 302–309, December 2016.

[23] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, and Peter Hochschild. Spanner: Googles globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[24] Ewa Deelman, Karan Vahi, Mats Rynge, Gideon Juve, Rajiv Mayani, and Rafael Ferreira da Silva. Pegasus in the cloud: Science automation through workflow technologies. *IEEE Internet Computing*, 20(1):70–76, 2016.

[25] W. Dou, X. Zhang, J. Liu, and J. Chen. HireSome-II: Towards Privacy-Aware Cross-Cloud Service Composition for Big Data Applications. *IEEE Transactions on Parallel and Distributed Systems*, 26(2):455–466, February 2015.

[26] Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin, and Arnor Solberg. Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems. pages 887–894, June 2013.

[27] Ashish Gupta, Fan Yang, Jason Govig, Adam Kirsch, Kelvin Chan, Kevin Lai, Shuo Wu, Sandeep Govind Dhoot, Abhilash Rajesh Kumar, and Ankur Agiwal. Mesa: Geo-replicated, near real-time, scalable data warehousing. *Proceedings of the VLDB Endowment*, 7(12):1259–1270, 2014.

[28] Mohammad Hajjat, David Maltz, Sanjay Rao, and Kunwadee Sripanidkulchai. Dealer: application-aware request splitting for interactive cloud applications. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 157–168. ACM, 2012.

[29] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. page 21.

[30] Chien-Chun Hung, Leana Golubchik, and Minlan Yu. Scheduling jobs across geo-distributed datacenters. pages 111–124. ACM Press, 2015.

[31] Fan Jiang, Claris Castillo, and Stan Ahalt. Cachalot: A network-aware, cooperative cache network for geo-distributed, data-intensive applications. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018.

[32] Albert Jonathan, Abhishek Chandra, and Jon Weissman. Awan: Locality-aware Resource Manager for Geo-distributed Data-intensive Applications. page 12.

[33] Tim Kraska, Gene Pang, Michael J. Franklin, Samuel Madden, and Alan Fekete. MDCC: Multi-data center consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 113–126. ACM, 2013.

[34] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. Low Latency Geo-distributed Data Analytics. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 421–434, New York, NY, USA, 2015. ACM.

[35] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S. Pai, and Michael J. Freedman. Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area. In *NSDI*, volume 14, pages 275–288, 2014.

[36] Y. Singh, F. Kandah, and Weiyi Zhang. A secured cost-effective multi-cloud storage in cloud computing. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 619–624, April 2011.

[37] Emil Stefanov and Elaine Shi. Multi-cloud Oblivious Storage. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 247–258, New York, NY, USA, 2013. ACM.

[38] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. CLARINET: WAN-Aware Optimization for Analytics Queries. In *OSDI*, volume 16, pages 435–450, 2016.

[39] John Vivian, Arjun Arkal Rao, Frank Austin Nothaft, Christopher Ketchum, Joel Armstrong, Adam Novak, Jacob Pfeil, Jake Narkizian, Alden D Deran, Audrey Musselman-Brown, Hannes Schmidt, Peter Amstutz, Brian Craft, Mary Goldman, Kate Rosenbloom, Melissa Cline, Brian O'Connor, Megan Hanna, Chet Birger, W James Kent, David A Patterson, Anthony D Joseph, Jingchun Zhu, Sasha Zaranek, Gad Getz, David Haussler, and Benedict Paten. Toil enables reproducible, open source, big biomedical data analyses. *Nature Biotechnology*, 35:314 EP –, 04 2017.

[40] Ashish Vulimiri, Carlo Curino, Philip Brighten Godfrey, Thomas Jungblut, Konstantinos Karanasos, Jitendra Padhye, and George Varghese. Wanalytics: Geo-distributed analytics for a data intensive world. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1087–1092. ACM, 2015.

[41] Ashish Vulimiri, Carlo Curino, Philip Brighten Godfrey, Thomas Jungblut, Jitu Padhye, and George Varghese. Global Analytics in the Face of Bandwidth and Regulatory Constraints. In *NSDI*, volume 7, pages 7–8, 2015.