D3W: Towards self-management of distributed data-driven workflows with QoS guarantees

Mengsong Zou[1], Javier Diaz-Montes[1], Kiran Nagaraja[2], Nimish Radia[2], and Manish Parashar[1]
[1] *Rutgers Discovery Informatics Institute, Rutgers University, USA*
[2] *Ericsson, USA*

***Abstract*—Data-driven application workflows that leverage compute capabilities and hosted services near the network edge can support latency-sensitive and critical applications in emerging areas such as Internet of Things (IoT) and smart infrastructure. However, distributed instantiation and execution of these workflows using resources across service providers and datacenters can be challenging.**

In this paper, we present the formulation of a decentralized workflow management approach for the autonomous instantiation and execution of dynamic data-driven workflows based on the opportunistic discovery and composition of services on-demand. Given a workflow template specification, this approach allows us to decouple workflow stages, allowing the execution of different stages to be performed by individual services, which are discovered and instantiated dynamically, and can be independently scaled as needed. These services may be geographically distributed and may be offered by different service providers using various QoS levels and cost models. The design, implementation and experimental evaluation of a decentralized workflow management framework using a live media stream application in a multi-cloud infrastructure is presented. Evaluations using a sample topology shows up to 2.5 times increase in QoS-meeting throughput when using our dynamic multi-cloud approach instead of using a fixed centralized cloud of identical capacity.

***Keywords*-Multi-Cloud, Multimedia Streaming, Distributed Workflows**

I. Introduction

Advances and pervasive computing and communication technologies are enabling the deployment of compute capabilities and hosted services near the network edge to support latency-sensitive and critical applications in emerging areas such as Internet of Things (IoT) and smart infrastructure [2, 13]. The resulting pervasive service environment is enabling the formulation of a new class of application workflows that are based on the dynamic data-driven composition of these services to achieve desired trade-offs in response time, quality of service, cost and execution efficiency. Example application workflows include end-to-end media flows (e.g., for consumer entertainment, remote education, or video conferencing), surveillance, smart infrastructure, etc.

Distributed data-driven application workflows (D3W) typically have the following characteristics – they are formulated as a sequence of stages, are data intensive and have a geographically distributed data footprint, have non-trivial compute, storage, and network requirements, and may have stringent performance and reliability constraints. As a results, distributed instantiation and execution of these workflows using resources across service providers and datacenters can be challenging, due to, for example the heterogeneity of the resources and providers, as well as the variability in resource pricing, loads, network performance, etc. As a result, such application workflows are traditionally composed and managed centrally using resources hosted at a backend datacenter [9, 15, 16], which can limit the ability to scale and adapt to changes in the infrastructure.

In this paper, we present the formulation of a decentralized workflow management approach for the autonomous instantiation and execution of dynamic data-driven workflows based on the opportunistic discovery and composition of services on-demand. Given a workflow template specification, this approach decouples workflow stages, allowing the execution of different stages to be performed by individual services, which can be dynamically instantiated and scaled. These services may be geographically distributed and may be offered by different service providers using various QoS levels and cost models. The presented approach enables self-management of workflows by allowing services to discover and aggregate capabilities and capacities located across the infrastructure (from the edge to the core), and to make decisions that guarantee end-to-end QoS.

For example, consider an online video streaming application that distributes a live-captured video to one or more endpoints. The application workflow templates specifies the processing steps (filters, transcoding, ad-insertion, etc.), QoS requirements (e.g., maximum processing delay), and the destination for the end result (e.g., a cable head end, or an OTT streaming server), and is presented to a decentralized workflow scheduler. The decentralized scheduler discovers available services across the entire distributed infrastructure for each of the workflow steps, and instantiates and schedules services for the steps along the path from source to destination based on specified requirements and constraints. The selection of services could potentially minimize transfer overheads between processing steps, allocate best-fit resources to meet QoS, and/or improve overall

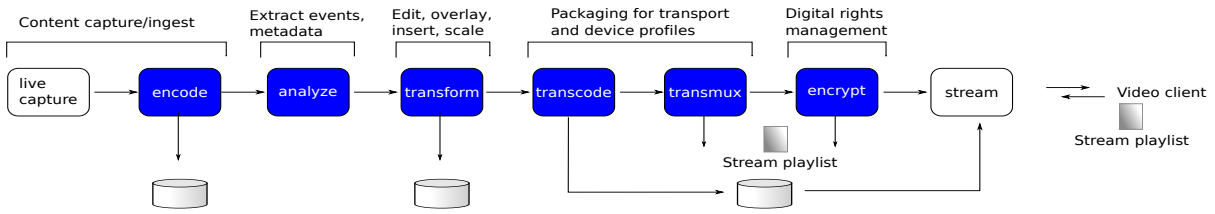


Figure 1: A sample workflow template for live streaming. The workflow composition approach presented in this paper addresses the dynamic instantiation of services for the colored boxes on distributed cloud resources.

resource utilization by balancing the workload across the distributed infrastructure.

The main contributions of this work are:

- Formalization of the problem of executing application workflows using services across a multi-cloud federation with QoS guarantees.
- Design and implementation of a decentralized workflow management framework that can discover and aggregate capabilities from multiple service providers with various QoS and cost models, towards the execution of application workflows.
- Experimental validation of the proposed approach using a live video streaming workflow and a multi-cloud federation.

II. Driving Use Cases

We focus in this paper on live video and multimedia workflows, where the challenges of managing the compute, storage and network demands are immediately apparent. Below we discuss two such representative scenarios that can benefit from flexible execution of workflow components on cloud resources available along the workflow path.

Live Consumer Media Delivery: These end-to-end media workflows deliver live video streams to consumers either through cable networks or OTT. The typical workflow is highly distributed with points of capture in a studio or in-field, a data-center or on-premise hosted processing platform, one or more distribution channels (e.g., IPTV, cellular), and finally delivered to consumer devices. The video data undergoes a number of transformations along the way prior to delivery, making up the stages of the workflow. Traditionally, the majority of these transformations are executed in a central platform prior to delivery, requiring ingestion of high quality content to this one location. We think that stages of this workflow can be executed in a more distributed manner (e.g., close to the origin) to improve efficiency and avoid central bottlenecks. Recent proposals have also pointed to the benefits and feasibility of delaying transformations much closer to the delivery end-point [3, 4].

IoT Video Streaming: We consider video surveillance and monitoring systems, where data from remote video cameras is transferred to a central location for analysis, archival, and further streaming. Depending on the capability of the end devices and the nodes enroute to the central location, some of the processing steps can alternatively be done at the origin or along the way. The decision also depends on the cost of transferring raw data to the central location, which can be particularly expensive for video data. For instance, in a video surveillance use case, it may not always be necessary to stream to central location if the monitored subject hasn't changed or is uninteresting. An analysis step performed close to the origin point could lower video fidelity, pause the streaming, or store the video locally [13].

A. Case Study: Live Video Streaming Workflow

The two video streaming use cases above though quite different in their quality, performance and reliability requirements, are not that dissimilar in their workflows. Their workflow stages can be jointly captured as shown in Figure 1. In both scenarios, we assume that the video stream, following various transformations, would be streamed to end consumers using a HTTP-based adaptive bitrate streaming delivery protocol such as MPEG-DASH. These protocols create and deliver chunks of the video stream to clients that request them off of a server provided playlist.

The captured live stream is first passed to an *encoder* stage that outputs a compressed base stream more amenable for transport and handling by later stages. As described in the IoT use case, the resulting stream might be passed through an *analyze stage* to extract meta-data necessary for transformations or to invoke predetermined actions (e.g., to raise an alert). Next, the base stream may optionally be edited using a *transform* stage, e.g., branding, ad marking/insertion, close-captioning.A *transcoder* stage converts the base stream into one or more streams each corresponding to a separate video profile – for example, a resolution and bitrate combination – to cover targeted devices (e.g., TV, desktop, phone) and delivery channels (IPTV, WiFi, or cellular). These variant streams are pushed to storage for access by subsequent stages of the workflow or for VOD delivery. In preparation for streaming delivery, a *transmux* stage converts the format into fixed-time segments or fragments – typically between 2 and 10 seconds long – for HTTP streaming. Finally, DRM (digital rights management) support may be added to

the stream using an *encrypt* stage.

Simplifying Assumptions: While evaluating live streaming case study, we equate an incoming stream to a sequence of video buffers – e.g., Group of Pictures (GOP) split at the I-frame boundaries – which are then pushed through the workflow stages as a sequence of individual jobs. The processing delay, a key QoS requirement for the workflow, is then measured by tracking the progress of these jobs across all stages. Finally, the figure depicts storage handling for intermediate results. We assume fast local storage in this work, leaving the considerations of using a unified distributed storage layer accessible across workflow stages as future work.

III. Workflow Composition Approach

In this work, we propose an approach to support the execution of data-driven workflows with QoS guarantees by enabling the aggregation of capabilities and capacities offered by federated services. Next, we formalize the problem and detail our approach.

A. Application Model

All D3W scenarios mentioned before have similar characteristics. They have an initial phase of content capture or ingest, where data is generated at a location L_{src} and is going to be delivered as a stream to location L_{dest}. We assume that a stream is split into multiple chunks (e.g., files), that are sequentially processed by the system and delivered to clients. To process these chunks, we create a set of jobs $J = \{j_1, j_2, ..., j_k\}$. Each job j_i involves executing a workflow, defined as a direct acyclic graph (DAG). The vertices of the graph represent the set of stages $S = \{s_1, s_2, ..., s_n\}$, where each stage performs certain operations or functions over the data, as shown in Figure 1. The number and type of services processing the different stages of a workflow can vary depending on service availability and capabilities. The edges of the graph, represent data dependencies between stages. This paper uses a live media streaming workflow as use case, where the workflow is a pipeline of stages that sequentially processes the set of jobs J and delivers processed media content to the clients. Every job j_i is characterized by a profile, which includes resolution, container type, bit-rate, codec, etc. Moreover, a job defines a set of m global QoS constraints $QC_i = \{qc_{i1}, qc_{i2}, ..., qc_{im}\}$ for the whole workflow (e.g., budget, deadline).

B. System Model

Next we formalize our system model and workflow composition approach. Figure 2 depicts our model.

Infrastructure Layer. The infrastructure layer consists of a set of service providers $SP = \{sp_1, ..., sp_n\}$. For simplicity let us assume that each service provider represents an individual data center. Any given service provider sp_i offers a set of service types $SV_i = \{sv_{i1}, sv_{i2}, ..., sv_{it}\}$. In our model, a service type is an abstract reference that can be used to create instantiations of a specific type of service on-demand. Service instances are responsible for performing the actual data processing using the function specified by its type. A service type sv_{ij}, provided by service provider sp_i, is characterized by the tuple $\langle ST, l, P, C, D\rangle$, where:

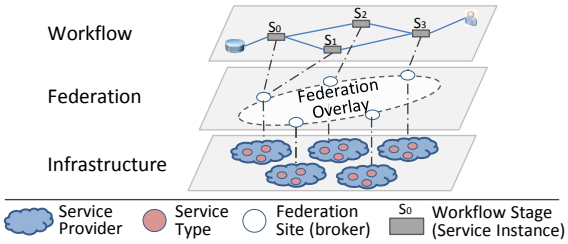


Figure 2: System Architecture.

- ST_{ij} is a set $\{s_1, s_2, ..., s_z\}; z > 0$ that defines the type of stages that this service type can perform. We assume uniform stage naming across service providers and, therefore, we can identify services using the stage names within ST_{ij} (handling semantics is out of the scope of this paper). For example, it could exist a service type that can only perform one type of stage, such as *Transcoder* or *Segmenter*, but it could also exist a service type that can perform both *Transcoder* and *Segmenter* by itself.
- l_{ij} is the geographic location of the service provider sp_i offering such service type.
- P_{ij} is a set of computational performances that instantiations of this service type can achieve for various input parameters and computational resources. It is defined as $CSPEC_{sv_{ij}} = \{(cspec_{sv_{ij}})_1, ..., (cspec_{sv_{ij}})_l\}$. This information can be obtained using historical executions [7, 11].
- C_{ij} is a set of costs per unit of time matching the computational performances in P. This set is defined as $Csv_{ij} = \{(ct_{sv_{ij}})_1, (ct_{sv_{ij}})_2, ..., (ct_{sv_{ij}})_l\}$, where $s = t$; $\forall\ (cspec_{sv_{ij}})_s \in CSPEC_{sv_{ij}}$ and $(ct_{sv_{ij}})_t \in Csv_{ij}$. These values can be as simple as a fixed price per unit of time for each type of service, or they could be obtained with a more involved cost function (e.g., considering demand and location).

The infrastructure is also composed by network resources connecting different service providers. We consider that the overheads within a service provider intranet are negligible compared with the wide area network ones. Hence, all service instances within a service provider share such service provider's properties. Specifically, a bidirectional network link connecting services providers sp_i and $sp_{i'}$ is characterized by: $bw_{ii'}$ which represents its total available bandwidth; $\lambda_{ii'}$ which represents its latency; $cl_{ii'}$ which represents the cost of transferring one gigabyte of data between

the two service providers; and $ntype_{ii'}$ which represent the type of network (reserved or shared). This information can be used to create a list of guarantees $\{(qg_{i-i'})_1, (qg_{i-i'})_2, ..., (qg_{i-i'})_n\}$. These guarantees include QoS parameters, such as estimated transfer time, and latency. Additionally, we include the costs of transferring all required data as part of its guarantees. This eases the service selection formulation in Section III-C.

Federation Layer. This layer uses the federation model proposed by the CometCloud framework [6]. This federation model creates an information lookup on top of a peer-to-peer overlay and a tuple-space abstraction to enable coordination and publish/subscribe messaging across peers (sites). The resulting federation allows different participating sites to join or leave at any point, talk to each other, discover services, and announce changes. We build on top of this federation model to expose service providers and enable coordination. In our approach, each federated site acts as a broker of a service provider, offering its services to the rest of the federation. Brokers also allow users to discover and request services in the federation regardless of their location.

Brokers subscribe to messages matching their service availability. When a message requesting certain types of services is published, all brokers with matching interests place one or multiple offers with different QoS guarantees and costs. Offers are generated using application information and QoS constraints combined with infrastructure historical information (e.g., computational performance, $CSPEC_{sv_{ij}}$, and cost, $C_{sv_{ij}}$). For example, let's say that a user places a request that can be processed by a service type sv_{ij} of the service provider sp_i.The broker representing service provider sp_i is going to generate a set of offers $Offers_{sv_{ij}} = \{(offer_{sv_{ij}})_1, ..., (offer_{sv_{ij}})_p\}$, where each offer $(offer_{sv_{ij}})_k$ contains a list of guarantees $\{(qg_{ijk})_1, (qg_{ijk})_2, ..., (qg_{ijk})_n\}$. These guarantees include different QoS parameters, such as estimated completion time, quality of the solution, and error margins. Additionally, we include the costs of the offer as part of its guarantees. This eases the service selection formulation described in Section III-C.

Workflow Layer. This layer is responsible for coordinating user's requests by autonomously creating workflows and ensuring end-to-end QoS. This layer is built-in within each broker in the federation, which allows brokers not only to accept requests from users, but also to coordinate their execution. Thus, when a user contact a broker, this acts as workflow coordinator and starts the process of identifying data sources (or entry point of the stream) as well as the different transformations or stages to be performed. Since we are considering an heterogeneous and dynamic infrastructure, an inquiry message is published through the federation layer to identify available services, across all service providers, that can be composed to satisfy the workflow template specification generated upon user's request. We also consider keeping information about past workflows to reduce the time required to identify workflow candidates.

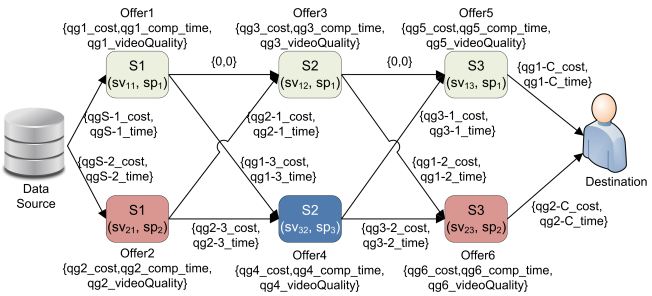


Figure 3: Example of service offering graph for a sequential workflow of three stages.

All offers received a the federation layer are collected to create a service offering graph to map all available service types with their various QoS guarantees. The service offering graph is a directed graph $OG(V, E)$ with a set of nodes V (representing offers, the source L_{src} and the destination L_{dest}), and a set of edges E (representing the network connection). The graph edges are created matching the stage ordering of the workflow. Figure 3 depicts a sample graph for a workflow with three sequential stages (s_1–>s_2–>s_3) and offers made by three different service providers. This service offering graph may vary depending on service availability, QoS requirements, user preferences, etc. Moreover, it can be regenerated over time if there are changes in the infrastructure. This service offering graph provides local QoS guarantees for individual services, hence we need to find a set of services that can meet the global QoS for the end-to-end workflow required by the user.

C. Service Selection Strategy

In this Section we describe a strategy that allows us to select a set of services that ensure end-to-end QoS. The offering service graph may have multiple combination of services that can meet the global QoS. Hence, the problem consists on finding the set of services that best fit user's needs. This requires being able to compare competing offers among themselves. Since each offer may have several local QoS guarantees, one way of comparing them is by normalizing values and using a weighted utility function. Thus, we normalize each QoS guarantee, qg, for every $(offer_{sv_{ij}})_k$ and $transfer_{i,i'}$ as follows:

$$(qg_{ijk})_{a\_norm} = \frac{(qg_{ijk})_a - min_{i,k}((qg_{ijk})_a)}{max_{i,k}((qg_{ijk})_a) - min_{i,k}((qg_{ijk})_a)} \quad (1)$$

$$(qg_{i,i'})_{b\_norm} = \frac{(qg_{i-i'})_b - min_{i,k}((qg_{i-i'})_b)}{max_{i,k}((qg_{i-i'})_b) - min_{i,k}((qg_{i-i'})_b)} \quad (2)$$

where i represents the service provider or site sp_i, j represents the service type sv_{ij}, k represents the offer $(offer_{sv_{ij}})_k$, a represents a specific service type QoS

attribute $(qg_{ijk})_a$, and b represents a specific network link QoS attribute $(qg_{i-i'})_b$. Minimum and maximum values are calculated among all i and k values.

We can use these normalized values to create a utility function for service type offers ($V_{util}[(offer_{sv_{ij}})_k]$) and another one for network links ($E_{util}[transfer_{i,i'}]$). These utility values now allow us to easily compare different offers and select the most appropriated ones.

$$V_{util}[(offer_{sv_{ij}})_k] = \sum_a ((qg_{ijk})_{a\_norm} * w_a) \quad (3)$$

$$E_{util}[transfer_{i,i'}] = \sum_b ((qg_{i-i'})_{b\_norm} * w_b) \quad (4)$$

where w_a and w_b are weights that allow us to adjust the result of the utility to different objectives. For example, if the QoS guarantees are video quality, completion time, and cost, we could prioritize finding the cheapest solutions.

Using these utility functions, we can define our service selection problem as a variant of the shortest path problem. Given an offering graph $OG(V, E)$ with source node L_{src}, target node L_{dest}, cost of each edge $(i, j) \in E$ is $E_{util}[e_{ij}]$, and cost of each node $i \in V$ is $V_{util}[v_i]$.We define a path as a sequence of nodes $v_1, ..., v_n$, where no node appears in the path more than once. Our problem is formulated as follows:

$$Minimize \sum_{ij \in E} (V_{util}[v_i] + E_{util}[e_{ij}])x_{ij} \quad (5)$$

subject to not violating any global QoS requirements and:

$$x_{ij} \in \{0, 1\}, \ \ \forall (i, j) \in E \quad (6)$$

$$\sum_{ij} x_{ij} \leq 1, \ \ \forall i \in V \quad (7)$$

$$\sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, & \text{if } i = L_{src} \\ -1, & \text{if } i = L_{dest} \\ 0, & \text{otherwise} \end{cases} \quad \forall i \in V \quad (8)$$

where x_{ij} is a binary variable that takes value 1 if the edge $(i, j) \in E$ belongs to the path, and consequently service offers on i and $j \in V$ are chosen to be part of the workflow – this is defined by constraints (6) and (7). Constraints (8) are flow conservation constraints to ensure that for all nodes, except source and destination, the number of incoming and outcoming edges that are part of the path is the same. The way to ensure global QoS requirements depends on the type of QoS requirement. For example, for a global deadline, the constraint would be $\sum_{ij \in E}(Time(v_i) + Time(e_{ij}))x_{ij} \leq globalDeadline$; while the constraint to ensure a minimum quality of video would be $VideoQuality(v_i) \geq globalVideoQuality, \forall v_i \in V$ that is part of the solution.

IV. Methodology

We validate our workflow composition approach by executing live media streaming workflows in a multi-cloud. We consider a media streaming workflow that includes four sequential stages, namely Encode, Transcode, Segment, and Encrypt, which have been characterized by executing the real application in a controlled environment using Docker containers. Video streams have a resolution of 1280x720P. Any given video stream generates a new chunk of 10 seconds video every 10 seconds. Our global QoS objective is to deliver a video chunk in less than 10 seconds to ensure uninterrupted video delivery to the client. For the considered resolution, a video chunk of 10 seconds amounts to 6 MB of data. This global QoS objective is ensured by composing and aggregating services, where each one of these services offer their own local QoS guarantees (i.e. performance).

In order to ensure reproducibility of the experiments, we emulate an actual geographically multi-cloud infrastructure in one of our local clusters. The characteristics of the actual multi-cloud infrastructure are described in Table I and Table II, cost is based on AWS pricing. Two clouds are part of Amazon Web Services, namely AWS_West and AWS_East, located in Oregon and North Virginia, respectively; one cloud is part of the Chameleon project [18] namely Chameleon, located in Texas; and one campus cluster namely Rutgers, located at Rutgers University, New Jersey. The performance of the resources is represented by the speedup, which has been experimentally calculated as a function of the performance of the machine used to characterize the workload. The network characteristics have also been experimentally obtained. We consider these sites offer services on-demand by deploying light-weighted processes to serve the computational requests (e.g., Docker containers) and therefore the overhead of allocating workload is minimal.

The emulation platform is a cluster composed of 128

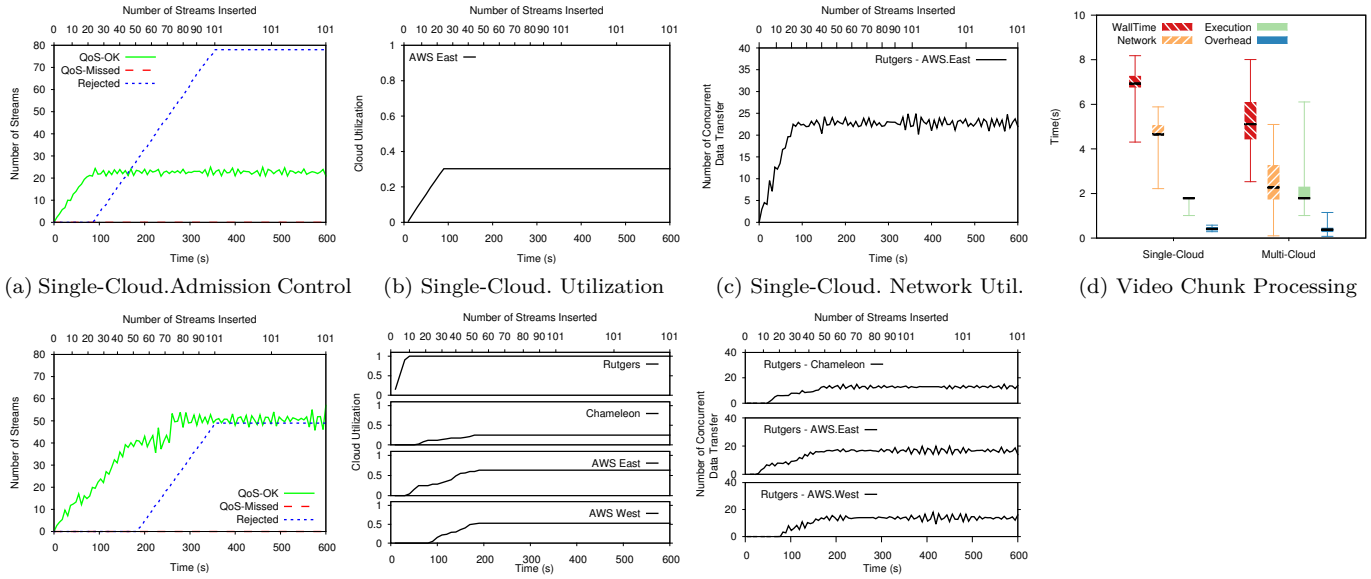
Table I: Resources available at each site and their characteristics.

VM type†	Cores	Mem.	Max.VMs‡	Perf.	Cost($/h)
AWS_East_m4.Xlarge	4	16 GB	4	0.55	0.239
AWS_East_m4.2Xlarge	8	32 GB	4	1.09	0.479
AWS_East_m4.4Xlarge	16	64 GB	4	2.18	0.958
AWS_West_m4.Xlarge	4	16 GB	4	0.55	0.239
AWS_West_m4.2Xlarge	8	32 GB	4	1.09	0.479
AWS_West_m4.4Xlarge	16	64 GB	4	2.18	0.958
Chameleon_Medium	2	4 GB	2	0.32	0.052
Chameleon_Large	4	8 GB	4	0.64	0.104
Chameleon_Xlarge	8	16 GB	4	1.27	0.208
Rutgers	16	24 GB	2	1.71	0.792

Note: † – Name of the site followed by the type of VM.
‡ – Maximum number of available VMs per type

Table II: Network speed in MB/s.

Network	Rutgers	AWS_East	AWS_West	Chameleon
Rutgers	-	15	10	20
AWS_East	15	-	15	10
AWS_West	10	15	-	10
Chameleon	20	10	10	-



(a) Single-Cloud.Admission Control (b) Single-Cloud. Utilization (c) Single-Cloud. Network Util. (d) Video Chunk Processing

(e) Multi-Cloud. Admission Control (f) Multi-Cloud. Utilization (g) Multi-Cloud. Network Util.

Figure 4: Summary of Experimental Results – Minimize Computational Time. QoS-OK and QoS-Missed represent accepted streams.

nodes, where each node has 2 CPU with 12 cores each at 2.5 GHz and 256 GB of memory, 1 PB storage, and 10 Gigabit Ethernet connection. We use different nodes of this cluster to emulate different sites as well as their computational resources. Network bandwidth is limited using Linux traffic controller (TC) tools, which allows us to control the bandwidth while transferring data across different machines, to match the bandwidth of the real infrastructure. The execution the workflow stages are emulated using a proxy code that simply occupies the amount of CPU and memory observed during the characterization of the actual workload.

The selection strategy described in Section III-C has been implemented as a variant of the best-first branch-and-bound strategy. This approach quickly provides a suboptimal solution that tends to have minimal overall utility value, as required by Equation 5. In our experiments we take the first solution found.

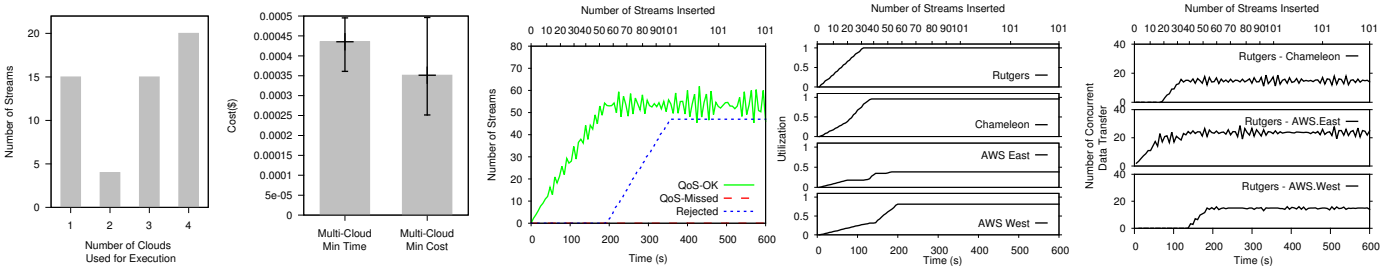
V. Evaluation

In these experiments, we consider all data centers described before offer all types of services required by the workflow. Moreover, each data center is highly specialized in one service, which is offered at a 50% discount, making it the cheapest – Rutgers discounted the Encode service, AWS_East Transcode, AWS_West Segment, and Chameleon Encrypt. The source of the live media streams is the Rutgers site.

Single-Cloud vs Multi-Cloud. In this experiment, we evaluate the behavior of our approach, which uses a federated infrastructure, and compare it with a traditional approach, which uses a single data center. Specifically, we want to observe the maximum number of concurrent streams that the system can process, given certain QoS guarantees, and the way resources are used. We consider that the available computational capacity is equivalent in both scenarios, i.e. a single data center has the same resources as the aggregated of all federated data centers, as described in Table I. The site in the single-cloud scenario is called AWS_East and it shares its location. The utility function used to select resources (Equation 3 and 4) is heavily weighted towards minimizing completion time. Results are summarized in Figure 4.

Figures 4a and 4e collect the results regarding the maximum number of live media streams that can be processed and delivered within QoS requirements. Figures 4a shows that a traditional approach, labeled single-cloud, was able to accept a maximum of 20 concurrent stream requests without violating the QoS requirements. However, we observe that a federated infrastructure, labeled multi-cloud, with the same computational capability was able to accept up to a 150% more workload, totaling around 50 concurrent streams – as described in Figure 4e. In either scenario, we only accepted those streams that, according to our scheduler, could be delivered within QoS constraints, the rest were rejected.

Figures 4b and 4f collect the utilization of each data center. Figure 4b shows that in the single-cloud scenario the utilization of the data center was up to 30%, which amounts to using 92 cores out of 308 available. On the other hand, Figure 4f shows that in the multi-cloud scenario the workload was allocated across different sites, balancing the resource utilization and minimizing waiting times. The resource utilization of each data center was around 50%, except Rutgers that was fully utilized. In overall, the multi-cloud scenario used up to 174 cores out of 308 available. Since there was plenty of



(a) Clouds Involved (b) Cost One Chunk (c) Admission Control (d) Utilization (e) Network Utilization

Figure 5: Summary of Experimental Results in Multi-Cloud – Minimizing Cost.

computational resources available in both scenarios, we conclude that the computational capacity was not the main reason for not accepting more streams.

Next, Figures 4c and 4g show the utilization of the network over the time of the experiment. We observe how the network utilization increased with the number of streams accepted. We considered a fair sharing of the network, hence the higher the number of concurrent streams using a network link, the slower each one was able to transfer its data. We observe that in both scenarios the maximum number of concurrent transfers was around 20 per link. This shows that the network was the main limiting factor that prevented us for accepting more workload. For this reason, having multiple data centers allowed us to accept more workload.

Finally, Figure 4d collects aggregated results of the processing time required for each video chunk. We observe that the median walltime to deliver each chunk was a 26% higher for the single-cloud scenario than for the multi-cloud scenario. Walltime was measured as the time since a video chunk was inserted into the system until it was processed and delivered. A significant portion of the walltime was spent transferring data, which in median represented a 66% for the single-cloud scenario and a 44% for the multi-cloud scenario. In the single-cloud scenario, the median transfer time is higher than in the multi-cloud one due to a rapid congestion of the network starting at time 80 compared to time 200 in the multi-cloud one. Moreover, the median execution time in both scenarios was similar, which again shows that the network was the main issue when processing live streams in a single data center. Finally, we observe that the median overhead required to orchestrate the execution of the workflows was less than a second.

Minimize Cost. This experiment uses the same multi-cloud scenario described above, but, in this case, we set the weights of the utility function towards minimizing the cost of computation. In particular we want to observe if our solution is able to find the way of reducing the execution cost of each stream workflow by composing services that are geographically distributed across clouds. Results are summarized in Figure 5.

Figures 5a shows the number of data centers involved in the execution of each stream. We can observe that a large number of stream workflows were composed by using services offered by different geographically distributed sites. As we mentioned in Section IV, each site had one service cheaper than the rest, hence our service selection strategy was able to schedule the streams not only taking into consideration the utilization of different sites, but also considering the price. More importantly there was no violation of the QoS of any stream during the experiment – as shown by Figure 5c. This figure also shows that we accept a similar number of streams. Figure 5b shows that the cost of executing a video chunk has been reduced up to a 19% compared with the multi-cloud scenario that was looking to minimize the completion time of each chunk. When minimizing cost, the framework tried to use the cheapest resources that would not violate the deadline, achieving a median execution time per stream of four seconds. However, a policy that tries to minimize computational time oftentimes compute faster than required, achieving a median execution time of two seconds, which increases costs. Finally, Figures 5c, 5d, and 5e show similar results as the ones obtained in the previous multi-cloud experiment – Figures 4e, 4f and 4g.

VI. Related Work

Service composition in cloud environments remains an active research topic [10]. The ever increasing popularity of the cloud computing model has resulted in a rapidly increasing number of services and service providers. Next, we highlight the most relevant work in the context of multimedia service composition for cloud environments. In [5], Cheng proposes a cloud-based media processing platform that uses prediction to minimize resource usage without affecting the QoS. Unlike our work, this solution is centralized and do not consider using multi-cloud infrastructures. Similar centralized approaches include [1, 14, 17].

On the other hand, decentralized service composition solutions include, SpiderNet [8], which allows multi-constrained QoS assurances and load balancing for service composition, and Dynamis [12], which uses an archi-

tecture similar to SpiderNet, but focuses on efficient service discovery probing mechanisms. Both [8] and [12] use peer-to-peer overlay to coordinate service composition and ensure global QoS constraints. The main difference with our work is that we consider multi-cloud environments where service providers can offer a specific service with multiple QoS guarantees matching their underlying computational resource availability. Additionally, our discovery mechanism is based on publish-subscribe to minimize the number of messages interchanged when looking for candidates.

VII. Discussion

The results presented in the evaluation section show how data-driven workflows can benefit from decentralized approaches that opportunistically take advantage of geographically distributed data-centers. We observed that our proposed approach not only is able to dynamically discover service availability and their characteristics, but it is also able to effectively use that information to steer the workload towards those data-centers that better fit users' needs. As a consequence, we can increase the amount of workload that the infrastructure can process while ensuring QoS guarantees. In our experiments, we observed that the network was the main bottleneck that limited the amount of workload that the infrastructure was able to accept. Hence, acquiring higher control over the network with custom VPNs and novel technologies such as SDN and NFV can have a major impact on the performance of the infrastructure, although may increase operational costs.

We also observed how our distributed workflow coordination approach is able to leverage distributed service offering and compose workflows across multiple data centers to reduce operational costs. In general terms, this is advantageous in marketplace scenarios where different service providers offer different services with different prices and characteristics. Alternatively, in cases where all service providers offer all type of services required by a workflow and we aim at minimizing computational time, we observed that executing the whole workflow within a single data center can be more appropriate. However, even in cases like this, being able to transparently allocate workflows through a federation of service providers can be beneficial to better balance the workload, increase the resilience of your infrastructure, and deal with failures and downtimes.

VIII. Conclusion

We presented a framework to enable distributed coordination of workflows across geographically distributed data centers. Our solution is able to dynamically discover and compose services to create dynamic workflows that can better support the required QoS levels. We performed a set of experiments using a live media workflow and show how our distributed architecture allows us to process more concurrent stream requests than a traditional centralized approach without compromising QoS guarantees. We observed that the main issue with a centralized approach is that the network link receiving the data may become a bottleneck when a large number of clients are requesting video streams concurrently. Our next steps include enhancing our autonomic capabilities to opportunistically modify already created workflows over time to improve the utilization of the infrastructure or take advantage of recently discovered services. Additionally, we would like to introduce other types of low priority workload and define policies that help to increase the utilization of the computational resources.

Acknowledgments: This research is supported in part by NSF via grants ACI 1339036, ACI 1441376, and Ericsson gift grant. Chameleon is supported by NSF OCI-1419152. We thank Dr. Esma Yildirim for her contribution at early stages of this work. The research at Rutgers was conducted as part of the RDI^2.

References

[1] A. Beyer, B. Cheng, J. Franke, and K. Sembiring. Media processing platform as a service in the cloud. In *Workshop on Multiuser Services for Social TV at EuroITV'13*, 2013.
[2] F. Bonomi, R. Milito, J. Zhu, et al. Fog computing and its role in the internet of things. In *Wkshp on Mobile Cloud Computing*, 2012.
[3] F. Bronzino, C. Han, Y. Chen, et al. In-network compute extensions for rate-adaptive content delivery in mobile networks. In *IEEE Intl. Conf. on Network Protocols (ICNP)*, 2014.
[4] J. Chen, R. Mahindra, M. A. Khojastepour, et al. A scheduling framework for adaptive video delivery over cellular networks. In *ACM Intl. Conf. on Mobile computing & networking*, 2013.
[5] B. Cheng. Mediapaas: A cloud-based media processing platform for elastic live broadcasting. In *IEEE CLOUD*, 2014.
[6] J. Diaz-Montes, M. AbdelBaky, M. Zou, and M. Parashar. Cometcloud: Enabling software-defined federations for end-to-end application workflows. *IEEE Internet Computing*, 19(1):69–73, 2015.
[7] A. Ghafoor and J. Yang. A distributed heterogeneous supercomputing management system. *Computer*, 26(6):78–86, 1993.
[8] X. Gu and K. Nahrstedt. Distributed multimedia service composition with statistical qos assurances. *IEEE Transactions on Multimedia*, 8(1):141–151, 2006.
[9] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, et al. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.
[10] A. Jula, E. Sundararajan, and Z. Othman. Cloud computing service composition: A systematic literature review. *Expert Systems with Applications*, 41(8):3809 – 3824, 2014.
[11] A. Khokhar, V. Prasanna, et al. Heterogeneous computing: challenges and opportunities. *Computer*, 26(6):18–27, 1993.
[12] F. A. Samimi and P. K. Mckinley. Dynamis: Dynamic overlay service composition for distributed stream processing. In *SEKE*, 2008.
[13] M. Satyanarayanan, P. Simoens, et al. Edge analytics in the internet of things. *IEEE Pervasive Computing*, 14(2):24–31, 2015.
[14] K. Sembiring and A. Beyer. Dynamic resource allocation for cloud-based media processing. In *Wkshp on Network and Operating Systems Support for Digital Audio and Video*, pages 49–54, 2013.
[15] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, et al. Apache hadoop yarn: Yet another resource negotiator. In *ACM Symposium on Cloud Computing*, page 5, 2013.
[16] A. Verma, L. Pedrosa, M. Korupolu, et al. Large-scale cluster management at google with borg. In *ACM European Conference on Computer Systems*, 2015.
[17] Amazon Elastic Transcoder. https://aws.amazon.com/elastictranscoder/.
[18] Chameleon Project. https://www.chameleoncloud.org/.