Unicorn: Unified Resource Orchestration for Multi-Domain, Geo-Distributed Data Analytics

Paper ID: 3
Qiao Xiang[+‡], X. Tony Wang[+‡], J. Jensen Zhang[+],
Harvey Newman[⋄], Y. Richard Yang[+‡], Y. Jace Liu[+],
[‡]Yale University, [+]Tongji University, [⋄]California Institute of Technology,
{qiao.xiang, xin.wang, yang.r.yang}@yale.edu, jensen@jensen-zhang.site,
newman@hep.caltech.edu, yang.jace.liu@linux.com

Abstract

As the data volume increases exponentially over time, data-intensive analytics benefits substantially from multi organizational, geographically-distributed, collaborative computing, where different organizations contribute various yet scarce resources, *e.g.*, computation, storage and networking resources, to collaboratively collect, share and analyze extremely large amounts of data. By analyzing the data analytics trace from the Compact Muon Solenoid (CMS) experiment, one of the largest scientific experiments in the world, and systematically examining the design of existing resource management systems for clusters, we show that the *multi-domain, geo-distributed, resource-disaggregated* nature of this new paradigm calls for a framework to manage a large set of distributively-owned, heterogeneous resources, with the objective of efficient resource utilization, following the autonomy and privacy of different domains, and that the fundamental challenge for designing such a framework is: *how to accurately discover and represent resource availability of a large set of distributively-owned, heterogeneous resources across different domains with minimal information exposure from each domain?* Existing resource management systems are designed for single-domain clusters and cannot address this challenge. In this paper, we design Unicorn, the first unified resource orchestration framework for multi-domain, geo-distributed data analytics. In Unicorn, we encode the resource availability for each domain into resource state abstraction, a variant of the network view abstraction extended to accurately represent the availability of multiple resources with minimal information exposure using a set of linear inequalities. We then design a novel, efficient cross-domain query algorithm to discover and integrate the accurate, minimal resource availability information for a set of data analytics jobs across different domains. In addition, Unicorn also contains a global resource orchestrator that computes optimal resource allocation decisions for data analytics jobs. We discuss the implementation of Unicorn and present preliminary evaluation results to demonstrate the efficiency and efficacy of Unicorn. We will also give a full demonstration of the Unicorn system at SuperComputing 2017.

1. Introduction

As the data volume increases exponentially over time, data-intensive analytics benefits substantially from multi-organizational, geographically-distributed, collaborative computing, where different organizations (also called domains) contribute various yet disaggregated resources, *e.g.*, computation, storage and networking resources, to collaboratively collect, share and analyze extremely large amounts of data. One important example of this paradigm is the Compact Muon Solenoid (CMS) experiment at CERN [1], one of the largest scientific experiments in the world. The CMS data analytics system is composed of over 150 participating organizations, including national laboratories, universities and other research institutes. By analyzing the data analytics trace from the Compact Muon Solenoid (CMS) experiment over a 7-day period and systematically examining the design of existing resource management systems for clusters, we show that the *multi-domain, geo-distributed, resource-disaggregated* nature of this new paradigm calls for a framework to manage a large set of distributively-owned, heterogeneous resources, with the objective of *efficient resource utilization, following the autonomy and privacy of different domains.*

In particular, our trace analysis shows that (1) over 35% of data analytics jobs are *remote jobs*, *i.e.*, jobs that require different types of resources from different domains for execution; (2) the 90% quantile of the job execution time of remote jobs is approximately 38.9% longer than that of local jobs, *i.e.*, jobs that only require resources from a single domain for execution; and (3) the data transfer traffic is saturating the CMS network, leaving limited networking resources (*i.e.*, less than 15%) for data analytics traffic. These observations show that resources in multi-domain, geo-distributed analytics are highly disaggregated, *i.e.*, unbalanced distributed across domains. Although there is much related work on resource manage-

ment for clusters and data centers, such as [2–12], they are mostly designed for managing resources in single-domain clusters, and cannot accomplish the aforementioned goal for multi-domain, geo-distributed data analytics. In particular, these systems typically adopt a graph-based abstraction to represent the resource availability in clusters. In this abstraction, each node in the graph is a physical node representing computation or storage resources and each edge between a pair of nodes denotes the networking resource connecting two physical nodes. This abstraction is inadequate for multi-domain, geo-distributed data analytics systems for two reasons. First, it *compromises the privacy of different domains* by revealing all the details of resources in each domain. Secondly, *the overhead to keep the resource availability graph up to date* is too expensive due to the heterogeneity and dynamicity of resources from different domains. Some systems such as HTCondor [2] adopts a simpler abstraction that only represents computation and storage resources in multi-domain clusters. This approach, however, leaves the orchestration of networking resources completely to the transmission control protocol (TCP), which has long been known to behave poorly in networks with high bandwidth-delay products including multi-domain, geo-distributed data analytics systems, and hence is inefficient. Through trace analysis and related work study, we identify the fundamental design challenge for designing an orchestration framework for multi-domain, geo-distributed data analytics is: *how to accurately discover and represent resource availability of a large set of distributively-owned, heterogeneous resources across different domains with minimal information exposure from each domain?*

In this paper, we design Unicorn, the first unified resource orchestration framework for multi-domain, geo distributed data analytics. In Unicorn, the resource availability of each domain is abstracted into resource state abstraction, a variant of the network view abstraction [13] extended to accurately represent the availability of multiple resources with minimal information exposure using *a set of linear inequalities*. With this intra-domain abstraction, Unicorn uses a novel, efficient cross-domain resource discovery component to find the accurate resource availability information for a set of data analytics jobs across different domains with minimal information exposure, while allowing each domain to make and practice their own resource management strategies. In addition, Unicorn also contains a global resource orchestrator that computes optimal resource allocation decisions for data analytics jobs.

This paper makes the following **main contribution**:

- we study the novel problem of resource orchestration for multi-domain, geo-distributed data analytics and identify the *cross-domain resource discovery challenge* as the fundamental design challenge for this problem through systematic trace-analysis and vigorously related work investigation;
- we design Unicorn, the first unified resource orchestration framework for multi-domain, geo-distributed data analytics. Unicorn provides the resource state abstraction for each domain to accurately represent its resource availability with minimal information exposure in the form of a set of linear equalities, a novel, efficient cross-domain resource discovery component to provide the accurate, minimal resource availability information across different domains, and a global resource orchestrator to compute optimal resource allocations for data analytics jobs;
- we discuss the implementation details of Unicorn and perform preliminary evaluations to demonstrate the efficiency and efficacy of Unicorn. We will also present a full demonstration of Unicorn at SuperComputing 2017.

The rest of the paper is organized as follows. We analyze the data analytics trace of the CMS experiment, discuss the inadequacy of existing resource management systems and identify the key design challenge for multi-domain, geo-distributed data analytics systems in Section 2. We introduce the system setting and give an overview of the Unicorn framework in Section 3. We then present the details of two key components of Unicorn, cross-domain resource discovery and representation and global resource orchestration, in Section 4 and 5, respectively. We discuss the implementation details in Section 6 and evaluate the performance of Unicorn in Section 7. We conclude the paper and discuss the next steps of Unicorn in Section 8.

2. Motivation and Challenge

Analytics trace from the CMS experiment. We collect the trace of approximately 479 thousand data analytics jobs from the CMS experiment, one of the largest scientific experiments in the world, over a period of 7 days. From this trace, we find that over 35% of jobs consumes resources across different domains, *i.e.*, these jobs use the computation node and the storage node located at different domains which are connected by networking resources across multiple domains. We call these jobs *remote jobs*, compared with *local jobs* which only use resources within one single domain. This result indicates the *resource disaggregation* in the CMS network, *i.e.*, the unbalanced distribution of storage and computation resources. We also plot the cumulative distribution function of job execution time for this set of traces as shown in Figure 1. We observe that the 90% quantile of job execution time for remote jobs has an extra 38.9% higher latency than local jobs. In addition, we observe that the cross-domain networking resources available for data analytics are very limited because the CMS data transfer traffic is saturating the limited networking resources, *e.g.*, the cross-domain data transfer network traffic of the same 7-day period has a total amount of 8785 terabytes while the cross-domain data analytics traffic is only 1404 terabytes. This observation indicates the scarcity of networking resources available for data analytics in the CMS network. All these results demonstrate that in order to support low-latency, multi-domain, geo-distributed data analytics, it is not only necessary, but crucial to design a multi-domain resource orchestration system.

Related work. There exists a rich literature in the field of resource management of clusters [2–12]. YARN [4]

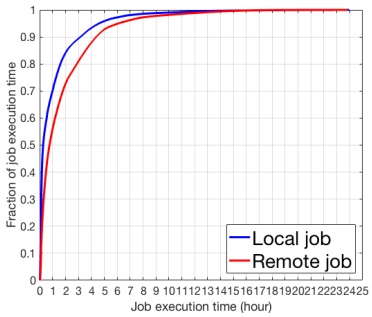

Figure 1: The CDF of job latency local and remote jobs.

is the core resource management framework of Hadoop. Mesos [3] is a platform designed to share resources among multiple cluster computing frameworks, *e.g.*, MapReduce [14], Spark [15], MPI and etc. Google designs a system called Borg [5] to orchestrate the cluster resources for its proprietary data analytics frameworks. Microsoft (*i.e.*, Apollo [6]) and Facebook (*i.e.*, Corona [7]) also develop similar systems tailored to their data analytics needs. These systems are all designed for managing resources in single-domain clusters and adopt a graph-based abstraction to represent the resource availability in clusters. In this abstraction, each node in the graph is a physical node representing computation or storage resources and each edge between a pair of nodes denotes the networking resource connecting two physical nodes. This abstraction is inadequate for multi-domain, geo-distributed data analytics systems for because (1) it compromises the privacy of different domains by revealing all the details of resources in each domain; and (2)the overhead to keep the resource availability graph up to date is too expensive due to the heterogeneity and dynamicity of resources from different domains.

There are also some efforts towards resource management for multi-domain clusters. HTCondor [2] proposes a ClassAds programming model, which allows different resource owners to advertise their resource supply and the job owners to advertise the resource demand. The CMS [1] experiment currently uses HTCondor and glideinWMS [8] to manage a set of distributively owned computing resources in a globally distributed system. These systems only focus on managing storage and computing resources in clusters, while the recent study shows that computation, storage and networking resources have approximately the same probability to become the bottleneck affecting the performance of data-intensive analytics jobs [16]. By leaving the orchestration of networking resources completely to TCP, which has been known to behave poorly in networks with high bandwidth-delay products including multi-domain, geo-distributed data analytics systems, the abstraction adopted by these systems is also inefficient.

Another line of work called geo-distributed data analytics is also related. Solutions in this field include (1) moving the input dataset to a single data center before the computation [17, 18] and (2) placing different amounts of tasks at different sites depending on dataset availability to achieve a better parallelization and hence a lower latency [9–12]. The main focus of these solutions is to optimize the usage of a set of dedicated networking resources. The design of these systems cannot be applied to multi-domain, geo-distributed data analytics where different types of resources owned by different owners need to be orchestrated.

Design challenge. The discussion above shows the urgent need for an efficient resource orchestration framework to support multi-domain, geo-distributed data analytics systems such as CMS. And by investigating the limitations of existing resource management systems, we identify the key design challenge for such a framework is *how to accurately discover and represent resource availability of a large set of distributively-owned, heterogeneous resources across different domains with minimal information exposure from each domain?* To this end, we design the Unicorn framework to manage a large set of distributively-owned, heterogeneous resources for multi-domain, geo-distributed data analytics systems. Unicorn achieves efficient resource utilization while allowing the autonomy and privacy of different domains through a novel resource state abstraction, an efficient cross-domain discovery and representation component and a global resource orchestration component, which will be discussed in the next few sections.

3. Overview

In this section, we introduce the system setting for multi-domain, geo-distributed data analytics and give an overview of the Unicorn framework and its workflow.

System settings. We consider a data analytics system composed of multiple organizations (domains). Each domain contributes a certain amount of computation, storage and networking resources for all the users in the system to store, transfer and analyze large-volume datasets. The storage and computation resources are typically physical servers, virtual machines or containers. The networking resources are typically switches and links. Domains that only contribute networking resources are called *transmission domains* and domains that also contribute computation and storage resources are called *leaf domains*. Figure 2 gives an example of such a system. In this example, domain A, B, E and F are all leaf domains while domain C and D are transmission domains.

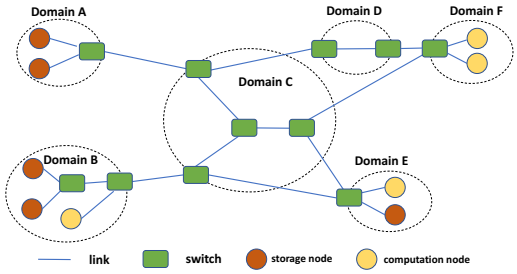

Figure 2: An example of multi-domain, geo-distributed data analytics system. Domains A, B, E and F are **leaf domains**. Domains C and D are **transmission domains.**

A data analytics task is typically decomposed into a set of jobs J whose precedence relation is specified by a

directed acyclic graph (DAG). A task is finished if and only if the last job in the decomposed DAG is finished. Each job j has requirements on storage and computation resources, *e.g.*, number of CPUs, size of memory, input dataset and etc. We use $(stg, comp)$ to denote a pair of candidate storage and computation resources satisfying the requirement of j. The orchestration system is in charge of selecting one $(stg, comp)$ pair for each job j and allocating the selected storage and computation resources and the networking resources connecting them for executing j.

Unicorn architecture. We present the architecture of Unicorn in Figure 3. On top of all the domains, Unicorn provides a logically centralized controller to orchestrate resources for data analytics jobs. This controller includes a cross-domain resource representation and discovery component and a global resource orchestration component. Residing in each domain are a domain resource manager and a set of job execution agents.

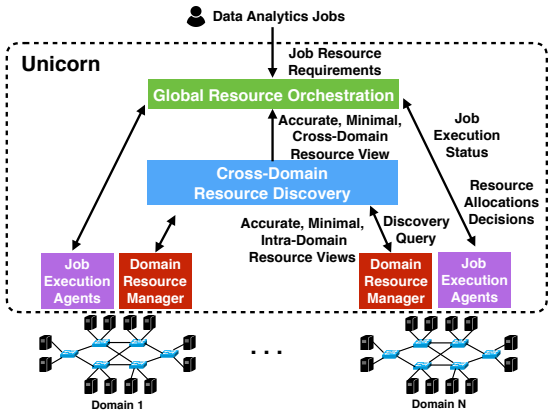

Figure 3: The architecture of Unicorn.

Unicorn provides a novel abstraction called resource state abstraction, a variant of network view abstraction [13]. This abstraction uses a set of linear inequalities to accurately represent the availability of different resources in each domain with minimal information exposure. When a set of data analytics jobs J are submitted to the Unicorn controller, the cross-domain resource discovery and integration component issues discovery queries, *i.e.*, path queries and resource queries, to the domain resource manager at each domain to retrieve the intra-domain resource view of each domain encoded in the resource state abstraction. It then assembles and compresses the responses into an accurate, minimal cross-domain resource view. This view, together with the resource requirements of j, is then used by the global orchestration component to make global, optimal resource allocation decisions and send to the job execution agents at corresponding domains. The execution agents enforce the received decisions, *e.g.*, starting the corresponding program, rate limiting the data accessing bandwidth and etc., and send the job execution status back to the Unicorn controller as feedback. In the next few sections, we present the design details of key components of Unicorn.

4. Cross-Domain Resource Discovery and Representation

In this section, we present our design to address the fundamental challenge of accurately discovering and representing a large set of distributively-owned, heterogeneously resources with minimal information exposure of resource owners. In particular, we introduce a novel abstraction to represent intra-domain resource availability and design an efficient discovery mechanism to discovery resource availability across different domains.

4.1. Intra-Domain Resource State Abstraction

Basic idea. Unicorn framework provides an abstraction called resource state abstraction to accurately represent the availability of multiple resources for a set of data analytics jobs using *a set of linear inequalities*. This is a variant of the network view abstraction [13]. In particular, we consider a set of data analytic jobs J that wants to consume a set of physical resources R (*i.e.*, computation, storage and networking) based on a set of pre-defined policies P. If a resource attribute $attr$ is *capacity-bounded*, *i.e.*, a resource r can only provide this attribute with a certain capacity (denoted as $C^{r,attr}$) and each job j consuming r can only get a portion of this attribute (denoted as $c_j^{r,attr}$), the resource availability of R for J on this attribute can be expressed as:

$$\sum_{j \in J(P,r)} c_j^{r,attr} \leq C^{r,attr}, \forall r \in R, \tag{1a}$$

$$c_j^{R,attr} = f(P, attr, c_j^{r,attr}), \forall (j, r \in R), \tag{1b}$$

$$c_j^{r,attr} = g(P, attr, c_j^{r',attr}), \forall (j, r \in R, r' \in R \{r\}). \tag{1c}$$

In this representation. Equation (1a) indicates that the total amount of $attr$ of resource r consumed by all the jobs cannot exceed the supply capacity of r on $attr$, where $J(P, r)$ is the set of jobs that are allowed to consume j based on the policy set P. Equation (1b) represents the total capacity of $attr$ that j can get from the whole set of resources R (denoted as $c_j^{R,attr}$) by a pre-defined linear function of $c_j^{r,attr}$, whose form depends on $attr$ and P. Equation (1c) represents the relation between the amount of $attr$ a job j can get from two resources r and r' by a pre-defined linear function, whose form depends on $attr$ and P. One of the most common capacity-bounded resource attributes is bandwidth.

If a resource attribute $attr$ is *capacity-free*, *i.e.*, each j consuming r who provides this attributes can get the same capacity $C^{r,attr}$ at the same time, the resource availability of R for J on this attribute can be expressed as:

$$c_j^{R,attr} = h(P, R, attr, j), \forall j \in J, \tag{2}$$

where the value of $c_j^{R,attr}$ is computed by a pre-defined function $h(P, R, attr, j)$ whose form depends on $attr$ and P. Note that this function does not need to be linear because the value of the right-hand side can be directly computed in this availability representation. Examples of such capacity-free resource attributes include propagation delay, hop-count, and etc.

Example. We use the physical topology in Figure 4 to illustrate how resource state abstraction works. Suppose

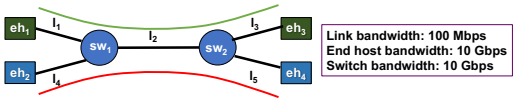

Figure 4: An example to illustrate the resource state abstraction.

two jobs j_1 and j_2 need to read data from storage node eh_1 to computation node eh_3 and from eh_2 to eh_4, respectively. The routing policy for the data flow of each job is also shown in the figure. For simplicity, we only focus on the bandwidth attribute for each resource, *i.e.*, end host, switch and link. Following the definition in Equation (1), the resource availability of this topology for j_1 and j_2 can be expressed as:

$$\begin{aligned}
c_{j_1}^{l_i} &\le 100Mbps, && i = 1,3,\\
c_{j_2}^{l_i} &\le 100Mbps, && i = 4,5,\\
c_{j_1}^{l_i} + c_{j_2}^{l_i} &\le 100Mbps, && i = 2\\
c_{j_1}^{sw_k} + c_{j_2}^{sw_k} &\le 10Gbps, && k = 1,2\\
c_{j_1}^{eh_m} &\le 10Gbps, && m = 1,3,\\
c_{j_2}^{eh_m} &\le 10Gbps, && m = 2,4,\\
c_{j_1}^R = c_{j_1}^{l_i} &= c_{j_1}^{sw_k} = c_{j_1}^{eh_m}, && i = \{1,2,3\}, \forall j, m = \{1,3\},\\
c_{j_2}^R = c_{j_2}^{l_i} &= c_{j_2}^{sw_k} = c_{j_2}^{eh_m}, && i = \{2,4,5\}, \forall j, m = \{2,4\},\\
c_{j_1}^{l_i} &= c_{j_1}^{eh_m} = 0, && i = \{4,5\}, m = \{2,4\},\\
c_{j_2}^{l_i} &= c_{j_2}^{eh_m} = 0, && i = \{1,3\}, m = \{1,3\},
\end{aligned} \tag{3}$$

Computing minimal, equivalent resource state abstraction. The representation of resource availability defined in Equations (1)(2) is accurate and complete, but may result in a large set of linear inequalities with redundant information. In a simple topology in our illustration example, there are already over 20 inequalities. Directly sharing them with a centralized controller or other domains would introduce a large communication overhead and expose unnecessary private information about each domain, *e.g.*, domain topology and policies. To minimize the resource information exposure of a domain, the domain resource manager of Unicorn adopts a lightweight, optimal algorithm to compress the original set of linear inequalities into a minimal, equivalent set of linear inequalities, which has the same feasible region as the original set but with a much smaller number of constraints. The basis of this compression algorithm is simple: given an original set of linear inequalities $C : \mathbf{Ax} \le \mathbf{b}$, we iteratively select one constraint $c \in C : \mathbf{a}^T\mathbf{x} \le b$ and calculate the optimal solution of problem $y \leftarrow \max \mathbf{a}^T\mathbf{x}$, subject to, $C - \{c\}$. If b is smaller than the resulting y, c is an indispensable constraint in determining the feasible region and will be put into the minimal, equivalent constraint set C'. Otherwise, c is a redundant constraint. The optimality of this algorithm can be proved via contradiction. Applying this algorithm to the example above, we may find that the minimal, equivalent set of linear inequalities has only one inequality: $c_{j_1}^R + c_{j_2}^R \le 100Mbps$. This reduction from over 20 inequalities to only one shows the power of our optimal compression algorithm.

4.2. Cross-Domain Resource Discovery

The resource state abstraction allows each domain to represent the accurate resource availability for a set of data analytics jobs using a set of linear inequalities with minimal information exposure, but it still requires the knowledge of all available computation, storage and networking resources, *i.e.*, the domain topology, and the domain policy to construct the original abstraction. As a result, it is non-trivial to extend it for resource discovery cross-domains, when a job needs to consume resources located in different domains, *e.g.*, the storage node and computation node assigned to the same job may be located in two different domains and are connected by network links across multiple domains. This is because information such as domain topology and policy is usually private to each domain itself and is not allowed to be passed around different domains. In this subsection, we present the details of our design to tackle this challenge and extend resource state abstraction for cross-domain resource discovery.

Basic idea. The key insight of our design is simple yet powerful: if we can "chop" the networking resources connecting a $(str, comp)$ candidate pair for job j based on the domains they belong to, as shown in Figure 5, we can then ask the domain resource manager of each domain to compute and represent the resource availability for j in each domain independently.

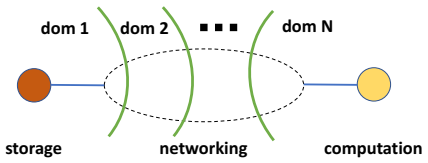

Figure 5: Chop the networking resources by domain.

With this insight, we design the cross-domain resource discovery process of Unicorn whose workflow is shown in Figure 6. In particular, Unicorn performs cross-domain resource discovery for a set of candidate $(stg, comp)$ pairs for a set of job J in three key steps. The first step is the path query process, in which the Unicorn controller issues path queries to the domain resource manager to recursively get a *domain path* in the form of

$$\begin{aligned}(dom_1, srcIP, egress) \rightarrow (dom_2, ingress, egress)\\ \rightarrow \ldots, (dom_N, ingress, dstIP),\end{aligned} \tag{4}$$

for each candidate $(storage, computation)$ node pair. We design path query to be executed in a recursive way to avoid the Unicorn controller becoming a performance bottleneck. The second step is the "chopping" process, which transforms the domain paths for all the $(stg, comp)$ candidate pairs, into a set of segments, *i.e.*, the chopping results, with the form of

$$(dom_i, F_i, F_i.ingress, F_i.egress), \tag{5}$$

for each domain, where F_i denotes the set of all $(stg, comp)$ candidate pairs whose connection use the network resource in domain i. Thirdly, the Unicorn controller sends each chopped segment to the corresponding domain resource manager to issue one resource query for each segment, collects the accurate, minimal intra-domain resource view for each segment, and assembles and further compresses these intra-domain resource views into an accurate, minimal cross-domain resource view representing the cross-

domain resource availability for a set of candidate $(stg, comp)$ pairs for a set of job J.

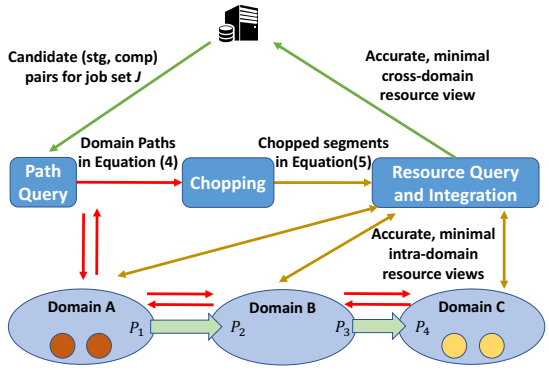

Figure 6: Workflow of cross-domain resource discovery.

Path query. We present the pseudocode of the path query process in Algorithm 1. The path query is a recursive query process. In particular, the path query algorithm requires the input of *domain*, which domain the query should be sent to, F, a set of $(stg, comp)$ candidate pairs whose connection use the network resource in *domain*, and *Ingress*, the set of ingress points each candidate pair is entering *domain* from. It starts from the Unicorn controller group the whole set of F into multiple disjoint subsets based on where the storage resources for this subset of pairs are located, and send one path query for each subset to each corresponding domain. When a domain resource manager receives such a query, it first computes the egress point, the next domain, and the ingress point of next domain for each candidate pair f (Line 3-4). Then the set F is grouped into several disjoint subsets based on the next domain of each pair f (Line 5). For each subset F_i whose next domain is not null, the current resource manager adds the current domain into the domain path for F_i and issues another path query to the domain resource manager at $F_i.nextDom$ to get the remaining part of the whole domain path (Line 8-12). If the next domain of F_i is null, it means that the computation resources of these $(stg, comp)$ pairs are in the current domain, *i.e.*, the domain path reaches the destination, and the domain manager simply returns such information to the querying party. During the path query process, each domain only provides the egress points, the next domains and the ingress points for $(stg, comp)$ candidate pairs without revealing any topology or policy information.

Chopping, resource query and integration. For the sake of integrity, we present the pseudocode of chopping and resource query and integration together in Algorithm 2. In particular, when the Unicorn controller receives the domain path for each $(stg, comp)$ candidate pair, it can use this information to chop each path by domains and get the chopping results in Equation (5) (Line 5-12). Then the Unicorn controller can perform efficient resource queries to collect the intra-domain resource view from each domain (Line 13-14) and integrate them into a single accurate, minimal cross-domain resource view using the same compression algorithm for intra-domain resource state abstraction (Line 15).

Algorithm 1: The algorithm of path query.

```
1  Function domPathQuery(domain, F, Ingress)
2      domPathResponse ← ∅;
3      foreach f ∈ F do
4          (f.egress, f.nextDom, f.nextDomIngress) ←
           getNextDomain(f);
5      {F_1, F_2, ...,} ← F.groupBy(f.nextDom);
6      foreach F_i do
7          if F_i.nextDom! = null then
8              domPathResponse ←
9              domPathResponse∪
10             (domain, F_i.egress)⊕
11             {domPathQuery(F_i.nextDom, F_i,
12             F_i.nextDomIngress)};
13         else
14             domPathResponse ←
15             domPathResponse ∪ {(F_i, null)};
16     return domPathResponse;
```

Algorithm 2: The algorithm of chopping, resource query and integration.

```
1  Function resourceQuery(F, F.domainPath)
2      resourceView ← ∅;
3      foreach domain do
4          domain.F ← ∅;
5      foreach f ∈ F do
6          hIdx ← 0;
7          dom ← getDom(f.domainPath, hIdx);
8          do
9              dom.F ← dom.F ∪ {f};
10             hIdx ← hIdx + 1;
11             dom ← getDom(f.domainPath, hIdx);
12         while dom ≠ null;
13     foreach domain do
14         resourceView ← resourceView ∪
           {resourceQueryByDomain(domain, F)};
15     resourceView ← MECS(resourceView);
16     return resourceView;
```

This resource query process is efficient due to the following lemma:

Lemma 1. *Given a set of candidate $(storage, computation)$ node pairs for a job set of J, Unicorn achieves the minimal number of resource queries at each domain.*

Proof 1. *With the domain path for each $(str, comp)$ candidate pair, the chopping process yields a set of segments defined in Equation (5), one segment for each domain. Hence the Unicorn controller only needs to generate one resource query for each domain if the corresponding F_i is not empty, which completes our proof.*

Schedulability. The cross-domain resource discovery process in Unicorn provides an accurate view of resource availability across domains with minimal exposure of private information. One important question left, however,

is whether this view provides full a schedulability of resources for a logically centralized orchestrator. We answer this question with the following theorem.

Theorem 1. *When all the resources represented in the final resource state abstraction queried from the cross-domain discovery process in Unicorn can be fully controlled on the edge,* i.e.*, all the attributes of each resource can be controlled by end host, the resource view provided by RSDP provides a full schedulability of resources to a centralized resource orchestrator.*

We omit the proof of this theorem due to the space limit.

5. Global Resource Orchestration

With the accurate, minimal cross-domain resource view, Unicorn performs global resource orchestration to compute optimal resource allocation decisions for a given set of jobs J. The modular design of Unicorn allows different allocation algorithms to be deployed. For simplicity, we consider a set of jobs J with no precedence from the same task, *i.e.*, all the jobs can be executed in parallel. We leave a more generic problem formulation as future work. We assume that each computation resource has infinite computation power, *i.e.*, the data accessing delay reading data from storage resources over networking resources to computation resources is the only bottleneck determining the delay for each workflow. For each job $j \in J$, let Stg_j denote the set of storage resources storing a copy of the input dataset of j, $Comp_j$ denote the set of computation resources that can execute j, v_j denote the volume of input dataset of j, and t_j denote the data accessing delay of j. We also use b_j^{mn} to denote the data access bandwidth for job j from storage resource m to computation resource n, and a binary variable I_j^{mn} to denote if j is assigned storage resource m and computation resource n simultaneously or not. Note that the global resource orchestration component relies heavily on the cross-domain resource discovery component in Section 4. To illustrate this argument, we first give a formulation of the global optimal resource allocation problem *without cross-domain resource discovery* as follows:

$$\text{minimize} \quad max_{j\in J}\{t_j\} \tag{6}$$

subject to

$$\sum_{\{j\in J|n\in Comp_j\}} \sum_{m\in Stg_j} I_j^{mn} \le 1, \qquad \forall n \in N, \tag{7a}$$

$$\sum_{m\in Stg_j} \sum_{n\in Comp_j} I_j^{mn} = 1, \qquad \forall j \in J, \tag{7b}$$

$$\frac{v_j}{\sum_{m\in Stg_j} \sum_{n\in Comp_j} b_j^{mn} I_j^{mn}} = t_j, \qquad \forall j \in J, \tag{7c}$$

$$A_1(BI) \le C_1. \tag{7d}$$

$$A_2(BI) \le C_2. \tag{7e}$$

$$\dots \tag{7f}$$

$$A_K(BI) \le C_K. \tag{7g}$$

In this formulation, Equation (6) indicates that the global resource allocation problem aims to minimize the data accessing delay for the whole set of jobs F. Equation (7a) ensures that for each computation resource, at most one job can be assigned. Equation (7b) ensures that only one computation resource and one storage resource are assigned for each job j. Equation (7c) calculates the data accessing delay for each job j. These constraints, *i.e.*, Equations (7a)(7b)(7c) are job-specific, *i.e.*, they express the requirements of data analytics jobs and can be changed accordingly based on different job requirements. The constraints in Equations (7d)(7e)(7f)(7g) are resource-specific, which depends not only on jobs' resource requirements, but also on the attributes provided by resources from each domain.

Though this formulation is accurate itself, its key limitation is that without a cross-domain resource discovery process, it is infeasible to find the resource-specific constraints in Equations (7d)(7e)(7f)(7g). On the contrary, the cross-domain resource discovery in Unicorn copes with this issue by providing the following constraint to accurately represent the resource availability for a given set of jobs with minimal information exposure.

$$A(BI) \le C. \tag{8}$$

With this formulation, the global optimal resource allocation problem *with cross-domain resource discovery* can be precisely defined as:

$$\text{minimize} \quad max_{j\in J}\{t_j\} \tag{9}$$

subject to

$$\text{Equations (7a)(7b)(7c)(8).} \tag{10a}$$

Solution. The multi-domain resource allocation problem defined above is complex in that it involves binary decisions, non-linear constraints and a complex objective function. To solve this problem, we first linearize the binary decision variables, then use a standard optimization solver to find the solution to the relaxed non-linear optimization problem, and then round-up the linearized decision variables back to the $\{0, 1\}$ feasible space to get the final resource allocation decisions. Because the cross-domain resource discovery process in Unicorn provides the resource view across domains with a minimal set of linear inequalities, the time overhead to solve the relaxed non-linear optimization problem is typically reasonable. We leave the task of finding a more efficient algorithm for this problem as future work.

6. Implementation

In this section, we discuss the implementation details of the Unicorn framework. The system implementation includes the following components:

Resource discovery protocol. We design and develop a query-based resource discovery protocol by extending the Application-Layer Traffic Optimization (ALTO) protocol [19], to deliver the resource state abstraction from each domain to the Unicorn controller. The protocol provides two major services: *path query service* and *resource query service*. The former is used for delivering next hop information to from domain resource managers the Unicorn controller. The latter is used for executing intra-domain resource queries. Table 1 summarizes the basic view of the two services.

Service	Path Query	Resource Query
HTTP Method	POST	POST
Media Type	`application`	`application`
Accept Subtype	`alto-flowfilter+json`	`alto-flowfilter+json`
Content Subtype	`alto-nextas+json`	`alto-pathvector+json`
Function	Implement $getNextDomain()$ in Algorithm 1.	Implement $resourceQueryByDomain()$ in Algorithm 2.

Table 1: Unicorn Resource Discovery Protocol

Domain resource manager. We build the prototype implementation of the domain resource manager on top of the OpenDaylight SDN controller [20]. From the view of the Unicorn controller, the domain resource manager works as a web service which provides the resource discovery protocol. From the view of the OpenDaylight controller, the resource manager is a consumer to re-process the topology, the traffic statistics, the intra-domain resource information and the inter-domain routing information.

The implementation includes two sub components: An OpenDaylight application running in the Karaf container; and a Python-based web service to provide the resource discovery protocol. The OpenDaylight application uses the API provided by Model-Driven SAL framework to read the real-time network information from the OpenDaylight DataStore. The two sub components communicate via RPC with each other. So the web service component is decoupled with the OpenDaylight and can be adapted to any other network management platform.

To implement the resource query service, we use the Python web service to look up the raw resource state for the given flow set from the OpenDaylight back end. Our native OpenDaylight application collects the topology and forwarding rules from the `network-topology` and `opendaylight-inventory` model of the DataStore, and computes the intra-domain resource state from these information. In our Python web service, we use GLPK as the underlying LP solver to calculate the minimal equivalent resource state abstraction described in Section 4.1. The solver API is wrapped by PuLP so that we could switch to other LP solvers like CPLEX and Gurobi without many modifications.

We implement the path query service as a BGP compatible service. The domain resource manager reads the inter-domain routing information from the OpenDaylight DataStore and converts it to the *BGP RIB* (Routing Information Base) format to respond the path query. The native OpenDaylight could support multiple inter-domain routing protocols by implementing their adapters. In this prototype, we only implement the BGP adapter which feeds the next-hop information of the inter-domain routing from the `bgp-rib` model.

Cross-domain resource discovery. The cross-domain resource discovery implements the two algorithms, path query (Algorithm 1) and resource query (Algorithm 2) and aggregate resource state abstraction from multiple domains to provide a aggregated resource state abstraction to the Global Resource Orchestration. It provides a high-level API `getGlobalResourceView` which accepts a set of node pairs $(srcIP, dstIP)$ as the queried flow set, and returns a set of linear inequalities as the global resource view. In addition, it also provides some lowel-level APIs including: `getDomainPath` that implements the Algorithm 1 and returns the domain path; and `getDomainResource` that retrieves the intra-domain resource view from a domain via resource discovery protocol.

Global resource orchestration. We implement the global resource orchestrator to subscribe to the analytics job management database. Once new jobs are inserted into the database, the orchestrator fetches them, performs cross-domain resource discovery and then make resource allocation decisions. It provides numerous Python APIs for developing new resource allocation algorithms. Therefore it is flexible for administrators to update the resource allocation policy. Our current orchestrator makes resource allocation decisions by solving the optimization problem defined in Section 5.

7. Performance Evaluation

We evaluate the performance of Unicorn through trace-based simulations. In particular, we focus on the efficiency of Unicorn in (1) discovering and represent a cross-domain resource view with minimal information exposure; and (2) performing global resource allocation decisions for data analytics jobs. All the simulations are conducted on a laptop with two 1.6GHz Intel i5 Cores and a 4GB memory.

7.1. Methodology

We emulate three multi-domain data analytics networks with different number of domains and topologies. For each setting, we first randomly select one topology from Topology Zoo [21] and let that topology be the domain-level topology with each node represent a single domain. And we also generate the intra-domain topology, *i.e.*, switches and the intra-domain links, for each domain. The emulated multi-domain topologies are labeled as Arpanet (composed of 4 domains), Aarnet (composed of 19 domains) and Chinanet (composed of 42 domains). We set the available link bandwidth within each domain to be 0.2-1Gbps and the available link bandwidth between domains to be 2-4Gbps. And we assume the I/O bandwidth of storage and computation resources are way larger than the bandwidths of links. We assume each domain's intra-domain and inter-domain routing policies both use the typical routing policies, *i.e.*, the shortest path routing, except that the former is on the router level and the latter is on the domain level. We vary the number of data analytics jobs J from the same

task to be from 5 to 30, each of which requires reading 1000 gigabytes of data.

7.2. Results

Cross-domain resource discovery and representation. We first present the compression ratio of the Unicorn in discovering and representing the accurate, minimal intra- / cross- domain resource views. This is computed as by dividing the number of linear inequalities in the accurate, minimal intra- / cross- domain over the number of original, linear inequalities used to represent the resource availability across domains. Figure 7 shows this compression ratio in a 19-domain data analytics network derived from the Aarnet topology [21] with different number of data analytics jobs, and Figure 8 shows this ratio under different number of domains when fixing the number of jobs to be 20. From these results we observe that the average compression ratio of intra-domain resource view is only around 60-70% while that of the cross-domain resource view is around 25-45%. These show that Unicorn provides a highly compact view of cross-domain resource availability for data analytics jobs. The higher compression ratio in the cross-domain view is because a multi-domain data analytics network provides more resources for data analytics jobs, *i.e.*, there are fewer jobs sharing the same set of resources. On the other hand, the fact that the highest cross-domain compression ratio is still 45% shows that even with more resources, jobs sharing the same set of resources is still a common situation, indicating the necessity and importance for discovering the accurate, minimal resource availability across domains.

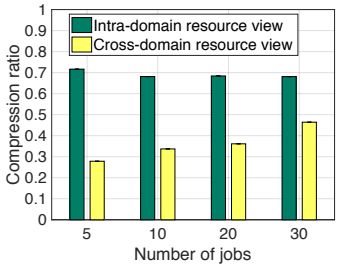

Figure 7: Compression ratio of intra-domain resource view and cross-domain resource view with varying numbers of jobs.

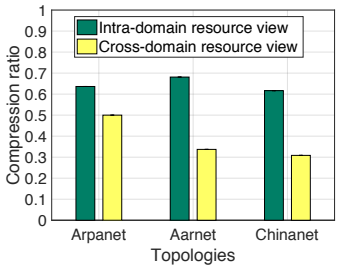

Figure 8: Compression ratio of intra-domain resource view and cross-domain resource view with different topologies.

We also plot the number of linear inequalities in the intra- /cross- domain view discovered by Unicorn in Figure 10 and Figure 10. We see that as the number of domains and the number of jobs grow, the number of linear inequalities in the accurate, minimal resource view computed by Unicorn increases at a very slow rate, which demonstrates the scalability of Unicorn.

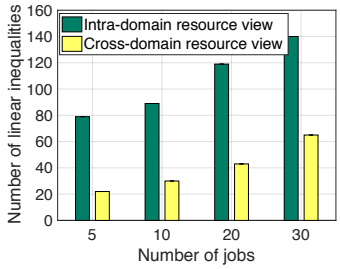

Figure 9: Number of linear inequalities in intra-domain resource view and cross-domain resource view with varying numbers of jobs.

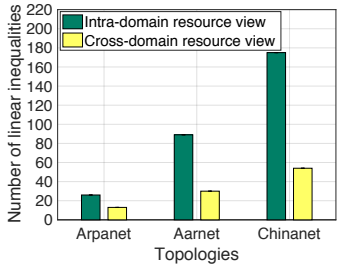

Figure 10: Number of linear inequalities in intra-domain resource view and cross-domain resource view with different topologies.

Global resource orchestration. We next demonstrate the efficiency of Unicorn in performing global resource orchestration for data analytics jobs. In particular, we focus on the latency of a task composed of a job set J, which is computed as the longest execution time of all jobs. In our evaluation, we assume all the computation nodes have the same computation power, hence we only need to focus on minimizing the maximal data accessing delay among all jobs, as defined in Equation (6). We compare the task latency provided by Unicorn with that provided by a domain-path based resource allocation scheme, which allocates computation and storage resources for a job based on the shortest AS path and use the classic max-in fairness mechanism to allocate bandwidth among data accessing flows of analytics jobs. We summarize the results under the combinations of different multi-domain topologies and different numbers of jobs in Table 2. We see that Unicorn provides an up to 65% task latency reduction in all cases. This shows that Unicorn provides a significant latency reduction for multi-domain data analytics.

8. Conclusion and Future Work

Summary. In this paper, we identify the objective and the fundamental challenge for designing a resource orches-

Topology \ #Jobs	5	10	20	30
Arpanet	31%	24%	27%	65%
Aarnet	27%	46%	55%	10%

Table 2: The reduction of task latency of Unicorn over the domain-path allocation scheme with max-min fairness.

tration system for multi-domain, geo-distributed data analytics system through analyzing the data analytics trace from one of the largest scientific experiments in the world and examining the design of existing resource management systems for single-domain clusters. We design Unicorn, the first unified resource orchestration framework for multi-domain, geo-distributed data analytics systems. Unicorn realizes the accurate, cross-domain resource availability discovery with minimal information exposure of each domain through the RSDP and a novel, efficient cross-domain resource availability query algorithm. Unicorn also provides a global resource orchestrator to compute optimal resource allocation decisions for data analytics tasks. We present the implementation details and the preliminary evaluation results of Unicorn.

Prototype and full demonstration at SuperComputing 2017. The source code and more comprehensive evaluation results of Unicorn will be open-sourced at [22]. A full demonstration of the Unicorn prototype will be given at SuperComputing 2017. In this demonstration, we will demonstrate the efficiency and efficacy of Unicorn on cross-domain resource discovery and global resource allocation in a multi-domain, geo-distributed data analytics system involving the Caltech booth, the USC booth and the UNESP booth at the conference exhibition, the SCinet network, and the Caltech testbed at Pasadena.

Acknowledgement

We thank Shenshen Chen, Shiwei Chen and Kai Gao for helpful discussion during the work. The Yale team was supported in part by NSF grant #1440745, CC*IIE Integration: Dynamically Optimizing Research Data Workflow with a Software Defined Science Network; Google Research Award, SDN Programming Using Just Minimal Abstractions; NSFC #61672385, FAST Magellan. The Yale team is also sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. The Tongji team was supported by China Postdoctoral Science Foundation #2017-M611618; NSFC #61702373. The Caltech team was supported in part by DOE/ASCR project #000219898, SDN NGenIA; DOE award #DE-AC02-07CH11359, SENSE, FNAL PO #626507; NSF award #1246133, ANSE; NSF award #1341024, CHOPIN.

Referencese

[1] T. C. Collaboration, The CMS experiment at the CERN LHC, Journal of Instrumentation 3 (08). doi:10.1088/1748-0221/3/08/S08004.

[2] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the Condor experience, Concurrency and computation: practice and experience 17 (2-4) (2005) 323–356.

[3] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, I. Stoica, Mesos: A platform for fine-grained resource sharing in the data center, in: NSDI, 2011.

[4] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al., Apache Hadoop YARN: Yet another resource negotiator, in: SoCC, ACM, 2013, p. 5.

[5] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at Google with Borg, in: EuroSys, ACM, 2015, p. 18.

[6] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, L. Zhou, Apollo: Scalable and coordinated scheduling for cloud-scale computing, in: OSDI, 2014, pp. 285–300.

[7] Under the hood: Scheduling MapReduce jobs more efficiently with Corona, http://on.fb.me/TxUsYN, [Online; accessed: 09-May-2017].

[8] I. Sfiligoi, D. C. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, F. Wurthwein, The pilot way to grid resources using glideinWMS, in: CSIE, IEEE, 2009, pp. 428–432.

[9] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, G. Varghese, WANalytics: Analytics for a geo-distributed data-intensive world, in: CIDR, 2015.

[10] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, I. Stoica, Low Latency Geo-distributed Data Analytics, in: SIGCOMM, ACM, 2015, pp. 421–434. doi:10.475/123.

[11] C.-C. Hung, L. Golubchik, M. Yu, Scheduling jobs across geo-distributed datacenters, in: SoCC, ACM, 2015, pp. 111–124.

[12] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, S. Wang, Rapier: Integrating routing and scheduling for coflow-aware data center networks, in: INFOCOM, 2015.

[13] K. Gao, Q. Xiang, X. Wang, Y. R. Yang, J. Bi, Nova: Towards on-demand equivalent network view abstraction for network optimization, in: IWQoS 2017, 2017.

[14] D. Jeffrey, G. Sanjay, MapReduce: simplified data processing on large clusters, Communications of the ACM.

[15] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, in: HotCloud'10.

[16] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, B.-G. Chun, V. ICSI, Making sense of performance in data analytics frameworks, in: NSDI, 2015, pp. 293–307.

[17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: Experience with a globally-deployed software defined WAN, in: SIGCOMM'13.

[18] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven WAN, in: SIGCOMM, ACM, 2013.

[19] R. Alimi, Y. Yang, R. Penno, RFC 7285, Application-layer traffic optimization (ALTO) protocol (2014).

[20] J. Medved, R. Varga, A. Tkacik, K. Gray, Opendaylight: Towards a model-driven SDN controller architecture, in: Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, 2014.

[21] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo 29 (9) 1765–1775, 00249.

[22] Public repository of unicorn, https://github.com/snlab/Unicorn.