Toward an Elastic Data Transfer Infrastructure

Joaquin Chung, Zhengchun Liu, Rajkumar Kettimuthu, Ian Foster
Argonne National Laboratory. USA
Email: jchung@mcs.anl.gov, zhengchun.liu@anl.gov, kettimut@mcs.anl.gov, foster@anl.gov

***Abstract*—Data transfer over wide area networks is an integral part of many science workflows that must, for example, move data from scientific facilities to remote resources for analysis, sharing, and storage. Yet despite continued enhancements in data transfer infrastructure (DTI), our previous analyses of approximately 40 billion GridFTP command logs collected over four years from the Globus transfer service show that data transfer nodes (DTNs) are idle (i.e., are performing no transfers) 94.3% of the time. On the other hand, we have also observed periods in which CPU resource scarcity negatively impacts DTN throughput. Motivated by the opportunity to optimize DTI performance, we present here an elastic DTI architecture in which the pool of nodes allocated to DTN activities expands and shrinks over time, based on demand. Our results show that this elastic DTI can save up to $\sim$95% of resources compared with a typical static DTN deployment, with the median slowdown incurred remaining close to one for most of the evaluated scenarios.**

I. Introduction

Science workflows that move large data over wide area networks [1] can perform badly on campus networks configured to protect business systems. The Science DMZ [2] network design pattern addresses this problem by configuring a portion of a campus network at or near the campus network perimeter with equipment, configuration, and security policies optimized for high-speed scientific data transfer. This model has helped improve data transfer performance for many science workflows. However, our analyses of $\sim$40 billion GridFTP command logs totaling 3.3 exabytes transferred, as well as 4.8 million transfers logs collected by the Globus transfer service from 2014/1/1 to 2018/1/1, show that much room remains for improvement in data transfer infrastructure [3].

One limitation of the current Science DMZ model is its statically provisioned data transfer nodes (DTNs), computers that are dedicated for wide area data transfers. Many science workflows are bursty, and thus their computing and network demands fluctuate significantly over time [4–7]. To handle these fluctuating demands from different departments and projects efficiently, university campuses often consolidate compute resources into an elastic private cloud. We argue that such an elastic infrastructure is required for data transfers as well, because statically provisioned DTNs result in either underutilization of resources or insufficient resources.

We present here the design and implementation of an elastic data transfer infrastructure (DTI) for a dynamic Science DMZ that leverages elastic resources for large data transfers. We evaluated our elastic DTI implementation using Docker containers, GridFTP, and `globus-url-copy`. Our results show that the elastic DTI can save up to 95% of resources compared with a typical DTN deployment, while the median slowdown remains close to 1 most of the time. Moreover, half of all transfers finish faster in the elastic DTI than in the typical DTN deployment that we used as baseline, as the elastic DTI is able to assign additional resources to those transfers.

The remainder of this paper is organized as follows. Section II provides background and motivation. Sections III and IV describe the architecture and design, respectively. Section V presents our implementation choices, and Section VI shows the evaluation results. Section VII summarizes our approach and its benefits and briefly outlines future work.

II. Background and Motivation

We first provide background on the Science DMZ model and discuss the limitations of the current Science DMZ model, which serve as the motivation for our work.

A. Science DMZ

The Science DMZ [2] network design pattern allows research institutions to support high-speed wide area data transfers. Local area networks (LANs) at most institutions are not designed to support large science data flows. Thus, the Science DMZ is deployed at the network perimeter, in order to minimize the number of network devices in the data path to the wide area network. It engages the following resources [8]:

- Dedicated data transfer nodes (DTNs) with network capabilities that match that of the wide area network (WAN) and that run high-performance data transfer tools such as Globus GridFTP [9]
- Performance monitoring hosts with network-monitoring software such as perfSONAR [10] to conduct both active and passive network measurements
- Security policies and tools that can be applied specifically to the science-only traffic

B. Static DTNs, resource wastage, and resource shortage

We previously [3] analyzed GridFTP server usage logs from about 1,800 DTNs. Working with data from these DTNs for the year 2017, we marked a DTN as active during a specific minute if it participated in at least one transfer at some point during that minute; otherwise we marked it as idle. Figure 1 shows the cumulative distribution of the time that DTNs are active. Clearly, the percentage of active time is low: on average, DTNs are completely idle (i.e., no transfers) 94.3% of the time, and 80% are active less than 6% of the time.

However, some endpoints are heavily used. Figure 2 shows the aggregate throughput and CPU utilization for a heavily

978-1-7281-2451-3/19/$31.00 ©2019 IEEE
DOI 10.1109/eScience.2019.00036

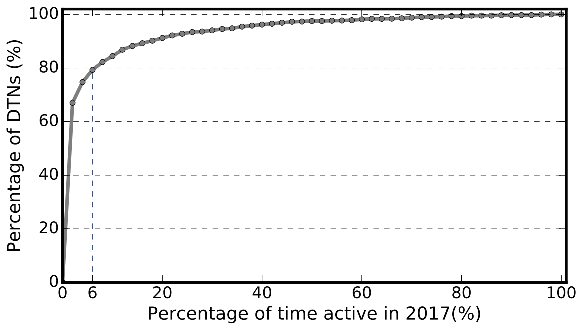

Fig. 1: Cumulative distribution of idle time percentage: 80% of DTNs were active less than 6% of the time [3]

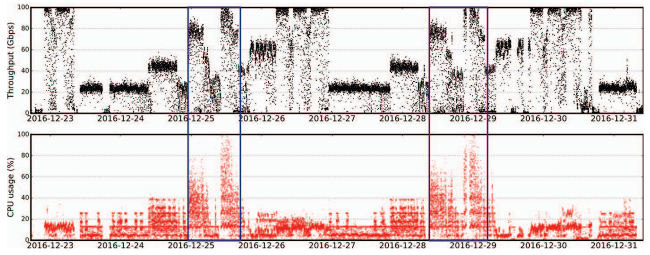

Fig. 2: Data transfer throughput vs. DTN CPU usage over eight days. We highlight two periods in which performance appears bottlenecked due to inadequate CPU resources [1].

used DTN for an eight-day period. We note that the CPU is potentially the bottleneck in the two highlighted regions.

Thus DTN resources are grossly underutilized in many cases but are sometimes insufficient. Both issues can be addressed by integrating DTN and compute resources and using cloud technologies to manage both in an elastic manner.

III. Architecture

To address the resource wastage and shortage of dedicated and static DTNs, we propose an elastic data transfer infrastructure (DTI) that expands and shrinks dynamically to conserve resources. To realize an elastic DTI in a nondisruptive fashion, we propose to reserve minimal resources that will work as a dedicated DTN; we call this a *thin dedicated DTN*, which is a minimal resource replica of a regular DTN. Whenever the load on the elastic DTI increases above a certain threshold, additional resources will be dynamically allocated from a pool. Likewise, when the load decreases below a certain threshold, excess resources will be sent back to pool.

The elastic DTI architecture, shown in Figure 3, is composed of an orchestrator and agents. The orchestrator runs on the thin dedicated DTN, while agents run on the thin dedicated DTN as well as on each on-demand DTN.

The **orchestrator** comprises a statistics collector, which aggregates utilization measurements taken by the agents in each DTN; a decision engine, which uses the utilization statistics to decide whether to (de)provision resources; and a set of tool-specific modules for making configuration changes

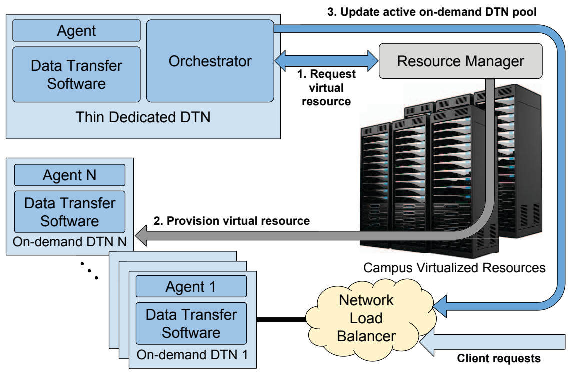

Fig. 3: Elastic DTI architecture

to transfer tools when DTN resources are (de)provisioned (e.g., change the endpoint definition in Globus). The orchestrator interacts with existing resource managers on the campus cyberinfrastructure through standard APIs.

An **agent** on each DTN collects resource utilization statistics. Communication between this agent and the statistics collector on the orchestrator can be implemented via either a polling or push model. In the former, the orchestrator periodically requests statistics from each agent; in the latter, agents send measurement updates continuously to the orchestrator.

IV. Design

The architecture proposed in Section III raises many questions. What type of resources are more suitable for realizing an elastic DTI, and how do we (de)provision them? When do we (de)provision these resources? Where do we (de)provision resources? How do we guarantee correct network connectivity between the WAN and these resources?

A. Virtualized Infrastructure

The minimal resource unit that can be provisioned dynamically is a core and portion of memory. Virtual machines and containers [11] can be used to (de)provision resources on demand. Containers are more attractive because of their short (de)provisioning times.

Regarding how to (de)provision these resources, two scenarios exist. In scenario (1), the campus already has a virtualized infrastructure with resource managers such as OpenStack [12], CloudStack [13], or Kubernetes [14] to serve the needs of their users. In Scenario (2), no virtualized resources are on campus. For scenario (1), the orchestrator can (de)provision resources for on-demand DTN using the appropriate APIs. For scenario (2), a portion of unused DTN resources can be made available through a container orchestration layer to scavenger jobs—jobs that can be killed as and when data transfer load increases.

B. Decision-Making

The question of when to provision or remove resource can be answered by applying different schemes on the decision engine. These schemes can be as simple as using thresholds over a monitored resource usage metric (e.g., CPU utilization) or provisioning a new resource for every new data transfer; or they can be as complex as decision-making based on machine learning. A simple static threshold may work in such cases where the system load does not vary much. However, there is no "one size fits all" configuration for each of the parameters because the external load and user behavior change over time and vary from endpoint to endpoint.

C. Resource Management

High-performance computing resource managers and container orchestration systems (e.g., Kubernetes) place a new resource on a host machine that has sufficient capacity to meet resource requirements. Usually, this capacity is defined in terms of available cores and memory, without taking into account available network bandwidth. For an elastic DTI, however, available network capacity is crucial. Placing an on-demand DTN on a network-congested node is counterproductive. We may be able to extend Kubernetes' code to take into account network throughput for where to place a new resource.

D. Dynamic Network Provisioning:

Statically provisioned DTNs reside on the Science DMZ, close to the WAN perimeter. What is needed for the elastic DTI is a way to dynamically connect virtualized resources with the WAN. We propose to reserve a pool of virtual LANs that connect the perimeter with the elastic DTI and to dynamically create (or delete) network paths whenever the orchestrator adds (or removes) a new container. Software-defined networking (SDN) may help, but we will adopt alternative approaches for campuses with no SDN support.

E. Design Space

TABLE I: Elastic DTI Design Space

When	Where
Usage threshold	Available core count
Usage threshold	Available network capacity
Upon arrival of new transfer	Available core count
Upon arrival of new transfer	Available network capacity

Table I shows the design space for our elastic DTI (see Section V for our implementation choices). Assuming that existing resource managers can be used to (de)provision resources, dynamic network paths can be (de)provisioned with resource, and containers are the minimal resource unit, our design space is reduced to two dimensions: when and where to (de)provision. For *when*, we consider two scenarios: *usage threshold* and *upon arrival of new transfer*. The "usage threshold" approach follows a time-share model in which one container can serve many transfers. When usage of the elastic DTI containers exceeds a threshold, we request more resources from the resource manager; we remove an active resource when utilization goes below a low threshold. In the "upon arrival" approach, we deploy a new container for every new transfer, and remove the container once the transfer is finished. For *where* to (de)provision, we consider the current state of the art in resource managers (i.e., consider the available core count) and an enhanced metric that considers the available network capacity or CPU usage.

V. Implementation

In this section we present our implementation choices for an elastic DTI. Our orchestrator and agents collect statistics on CPU usage, network throughput, and active transfers. We chose Globus GridFTP as our data transfer software because the extensive logs we had already collected [3] helped us create traffic traces for the evaluation. We used containers as the type of resources we can (de)provision and Docker as the campus resource manager. For this proof of concept, we implemented the communication between the orchestrator and agents using a polling model. Section IV presented two scenarios on how to (de)provision resources. We focus here on scenario 1 because many campuses are turning to virtualized resources.

We implemented the orchestrator and agents in Python and the communication between them using gRPC [15]. We also interact with Docker through a Python library and gRPC communication. The agents collect CPU utilization per container and network utilization statistics per bare metal server, each once per second. The orchestrator polls agents every second, computes thresholds, and decides whether a container needs to be provisioned or removed.

VI. Evaluation

We evaluated our architecture in the Chameleon cloud [16] testbed, which provides bare metal nodes with 48 cores of an Intel Xeon CPU E5-2670 v3 with 2.30 GHz and 128 GB of RAM. Four of these bare metal nodes served as the underlay of our elastic DTI experiments. Our minimal resource unit is a Docker container with one core and 2.667 GB of RAM. All containers in the same bare metal node share the same 10 Gigabit Ethernet card. Our evaluations use a trace with transfers whose characteristics follow that of real datasets (both file size and dataset size) [3] and whose arrival times follow a Poisson distribution with $\lambda = 3$ seconds.

We evaluated our elastic DTI implementation using the design space parameters described in Table I (i.e., when and where to (de)provision). For when to provision, we used both static thresholds (defined as percentages of average CPU utilization per container) and provisioning upon arrival of a new transfer. For where to provision, we used the core (container) count and network throughput usage of the bare metal nodes as our metrics. These parameters generate four possible elastic DTI schemes as shown in Table II.

TABLE II: Elastic DTI Schemes

Scheme	When	Where
CPU+Count	CPU Usage	Core count
CPU+Net	CPU Usage	Network throughput
U.A.+Count	Upon arrival of new transfer	Core count
U.A.+Net	Upon arrival of new transfer	Network throughput

We first created a baseline that matches the specifications of a typical DTN deployment (i.e., 12 cores and 96 GB of RAM per bare metal node) [17], and ran our trace on it. For

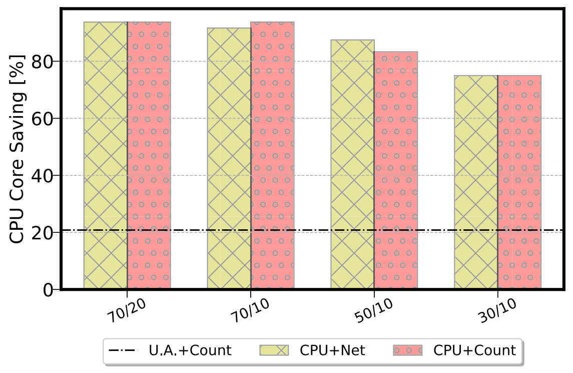

Fig. 4: CPU resources saved vs. typical DTN deployment

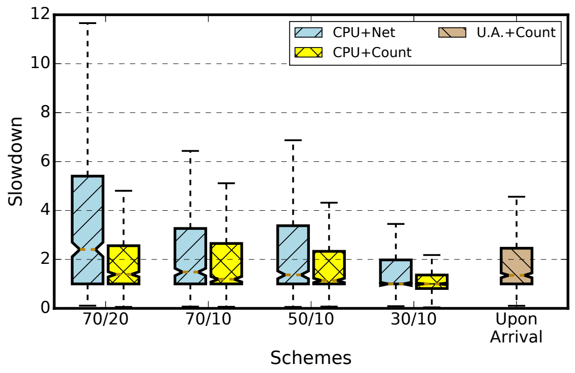

Fig. 5: Slowdown for elastic DTI schemes relative to baseline

our experiments, we measured the cores saving with respect to the baseline. We also measured the slowdown of each elastic DTI scheme in terms of transfer completion time. For the static threshold schemes, we derived four *(de)provisioning schemes* defined by a combination of high and low thresholds, and we used the *"HIGH/LOW"* notation for naming those schemes. We kept the low CPU usage threshold at 10% for the first three schemes and varied the high usage threshold as 30%, 50%, and 70%; for the fourth scheme the low threshold was 20% and the high threshold was 70%. Although we evaluated the four main schemes, we do not present the results for U.A.+Net because this scheme has large slowdowns and it actually uses ∼2X more resources than the baseline does.

Initial results show that at most ∼95% of the CPU core resources can be saved when compared with a typical DTN deployment (see Figure 4), with a median slowdown close to one for most of the threshold-based schemes (see Figure 5). When using the U.A.+Count scheme, however, CPU savings reach 20%, with a maximum slowdown close to ∼4.5X. We infer from Figures 4 and 5 that a tradeoff exists between increasing the number of resources saved and reducing slowdown. For instance, when using the CPU+Count 30/10 scheme, the maximum slowdown can be as low as ∼2X, but the core savings is ∼75%. On the other hand, the CPU+Count 70/20 scheme shows a maximum slowdown as high as ∼5X, but the core savings is ∼95%. In general, using core count for where to (de)provision produces better results in terms of slowdown compared with the network throughput metric. This may be because lags in the statistics measurement system give an inaccurate view of the elastic DTI system.

VII. Conclusion

The Science DMZ as a network design pattern has had a notable impact on the science community by improving the performance of wide area file transfers significantly. Yet it falls short in efficient utilization of data transfer nodes. We presented the design, implementation, and initial evaluation of an elastic data transfer infrastructure that grows and shrinks based on demand. We realized an instantiation of this elastic DTI in the National Science Foundation's Chameleon testbed and showed that up to ∼95% of the CPU resources can be saved when compared with a typical DTN deployment. Furthermore, the median slowdown incurred by the elastic DTI transfers remains close to one for most of the schemes. For future work, we will further investigate other metrics for when and where to (de)provision resources. We propose to evaluate adaptive thresholds for the elastic DTI schemes.

Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science contract DE-AC02-06CH11357.

References

[1] R. Kettimuthu *et al.*, "Transferring a petabyte in a day," *4th International Workshop on Innovating the Network for Data Intensive Science*, pp. 1–11, 2017.
[2] J. Crichigno *et al.*, "A comprehensive tutorial on Science DMZ," *IEEE Communications Surveys & Tutorials*, 2018.
[3] Z. Liu *et al.*, "Cross-geography scientific data transfer trends and user behavior patterns," in *27th ACM Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2018.
[4] R. F. da Silva *et al.*, "On the use of burst buffers for accelerating data-intensive scientific workflows," in *12th Workshop on Workflows in Support of Large-Scale Science*. ACM, 2017, pp. 2:1–2:9.
[5] Z. Liu *et al.*, "A mathematical programming- and simulation-based framework to evaluate cyberinfrastructure design choices," in *IEEE 13th International Conference on e-Science*, 2017, pp. 148–157.
[6] "Nuclear physics network requirements workshop, 2008," http://science.energy.gov/∼/media/ascr/pdf/program-documents/docs/Np_net_req_workshop.pdf.
[7] Y. Sun *et al.*, "Experience with bursty workflow-driven workloads in LEAD science gateway," in *TeraGrid Conference*, 2008.
[8] K. Chard *et al.*, "The Modern Research Data Portal: A design pattern for networked, data-intensive science," *PeerJ Computer Science*, vol. 4, p. e144, 2018.
[9] W. Allcock *et al.*, "The Globus striped GridFTP framework and server," in *ACM/IEEE Conference on Supercomputing*. IEEE, 2005, pp. 54–.
[10] B. Tierney *et al.*, "perfSONAR: Instantiating a global network measurement framework," *SOSP Workshop on Real Overlays and Distributed Systems*, 2009.
[11] linuxcontainers.org, "Linux containers," https://linuxcontainers.org/.
[12] "Openstack," https://www.openstack.org, accessed: 2018-01-29.
[13] "Apache Cloudstack," https://cloudstack.apache.org/, accessed: 2018-01-29.
[14] "Kubernetes - Production Grade Container Orchestration," https://kubernetes.io/, accessed: 2019-05-20.
[15] "gRPC," https://grpc.io/, accessed: 2018-12-14.
[16] "Chameleon Cloud," https://www.chameleoncloud.org/.
[17] "Data Transfer Node Reference Implementation," https://fasterdata.es.net/science-dmz/DTN/reference-implementation/, accessed: 2018-12-14.