Calibers: A Bandwidth Calendaring Paradigm For Science Workflows

Fatma Alali[1,3], Nathan Hanford[2,3], Eric Pouyoul[3], Raj Kettimuthu[4], Mariam Kiran[3], Ben Mack-Crane[5], Brian Tierney[3], Yatish Kumar[5] and Dipak Ghosal[2,3]

[1]*Department of Electrical and Computer Engineering, University of Virginia, Virginia, USA. Email: fha6np@virginia.edu*
[2]*Department of Computer Science, University of California, Davis CA, USA. Email: nhanford@ucdavis.edu, dghosal@ucdavis.edu*
[3]*Energy Sciences Network, Lawrence Berkeley National Laboratory, California, USA, Email: lomax@es.net, mkiran@es.net, bltierney@es.net*
[4]*Mathematics and Computer Science Division, Argonne National Laboratory, Chicago, USA. Email: kettimut@anl.gov*
[5]*Corsa Technology. Email: ben.mackcrane@corsa.com, yatish.kumar@corsa.com*

Abstract

Many scientific workflows require large data transfers between distributed instrument facilities, storage and computing resources. To ensure that these resources are maximally utilized, R&E networks connecting these resources must ensure that an inherently unpredictable network behaves predictably. In practice, this amounts to the per-application over-provisioning of network resources in an attempt to guarantee that adequate throughput is provided to users. This often results in resource under-utilization over time. One promising solution is the use of deadlines and bandwidth calendaring. In this approach, "fair" resource allocation is replaced with deadline-based resource allocation. However, these approaches often suffer from issues in efficiently regulating resource allocation and failure modes. Therefore, our solution, Calibers, approaches bandwidth calendaring and deadline-awareness in a different way. Calibers uses shaping, metering, and pacing at the edge of the network and end-system to provide participating clients the ability to schedule bandwidth reservations without having to worry about network noise from non-participating clients. Calibers can also fail back to the fair resource allocation of underlying transport protocols if necessary. For example, if a non-participating flow somehow enters the core of the network, or a sudden network change causes the available bandwidth to be exceeded, the underlying transport protocol congestion avoidance implementation will be able to handle the congestion as it normally would. Furthermore, Calibers provides a novel simulation method and resource allocation algorithm.

In this paper, we present the prototype architecture for Calibers using a central controller with distributed agents to dynamically pace flows at the ingress of the network to meet deadlines. Using Globus/Grid-FTP, we experimentally demonstrate that pacing can be used to meet data transfer deadlines which cannot be achieved using TCP. Finally, we present dynamic flow pacing algorithms that maximize acceptance ratio of flows for which deadlines can be met while maximizing network utilization. Our results show that simple heuristics, optimizing locally on the most bottlenecked link, can perform almost as well as heuristics that attempt to optimize globally.

Keywords: Bandwidth Calendaring, Flow Pacing, Software Defined Network (SDN), TCP, Dynamic Flow Pacing, Simulation Analysis, Traffic Shaping

1. Introduction

Scientific analysis in experiments such as high-energy physics or climate modeling, usually involve extremely complex workflows to ensure successful and reliable results. These workflows include a number of tasks, involve multiple actors, software and infrastructures, that work together as a workflow from data generation to delivery. For example, in the Advanced Light source (ALS) data is generated from multiple detectors which is then collected on an NERSC supercomputing data center via high-speed network connections. It is imperative that the data is delivered in a timely manner, with minimum loss, such that further computations can be performed using supercomputing resources that have to be *a priori* reserved. In order that the supercomputing resources are maximally utilized, this requires the network service to allow deadlines for large data transfers.

There are two approaches to ensure that the data transfers can be made with predictable performance and within requested deadlines. One approach is to use

advanced reservations of links, such as OSCARS or open NSA [1], that allow setting up circuits of specified capacities between routers. Advanced reservation schemes require additional time to setup circuits, are only associated with WAN border routers and are difficult to automate due to required user knowledge, network topology and request details. Furthermore, applications do not generate traffic all the time which leads to wasted reserved capacity.

The second approach is to run the network at low utilization and use standard TCP. New TCP protocols, such as TCP Hamilton [2] and BBR-TCP [3], can efficiently adapt to the bottleneck capacity and where multiple competing flows are involved, they *equally* split the bottleneck capacity. However, even with the new TCP algorithms, sustained bottlenecks lead to unpredictable throughput performance and difficulties in arbitrarily splitting bottlenecked bandwidths among competing flows. Finally, as the growth in data transfer volume out-paces the increase in the data link rates, running the network at low utilization is not cost effective [4].

To help accelerate the effort to run the network at high utilization and enable deadline aware data transfers, network automation through Software-Defined Networks (SDN) is being advanced to control network traffic depending on data demand. In principle, SDN allow individual switches to be managed and controlled following centralized traffic engineering principles [5]. Furthermore, SDN switches provide the ability to shape traffic at ingress of the network rapidly and in an on-demand fashion. These features in addition to TCP protocol, or the pacing algorithm at the source nodes [6], together provide the necessary tools to dynamically allocate bandwidth to flows for meeting deadlines while ensuring the network operates at high utilization. There has been network-utilization-focused work along these lines presented in [7, 8]. Simulation-based work on deadline-aware networking has also been carried out in [9, 10]. However, Calibers is not only capable of simulating the performance of these tools, it actually deploys a fully-functional, trans-continental deadline-aware SDN in a quasi-production network testbed, with some non-participating end-systems, to maximize deadline performance while also attempting to maximize link utilization.

This paper aims to implement a centralized traffic engineering approach and control distributed agents at the edge (ingress point of the network) to dynamically pace flow for meeting transfer deadlines, while achieving high network utilization. The dynamic pacing algorithm is able to analyze traffic patterns and follow a rolling horizon model to pace flows at appropriate rates to optimize network performance and meet deadlines. As a result, Calibers not only calenders flow, but also lays the foundation for future work where these capabilities can be coupled with advanced tools to control networks dynamically. Calibers aims to solve the problem of maximizing deadline performance and network utilization while minimizing flow rejection, given some set of network bottlenecks on a link-state routed network. Calibers is protocol-agnostic, and with the use of edge-pacing, is able to function on a network core, regardless of the presence of non-participating end-systems, whose traffic can be shaped at the network edge to relinquish bandwidth for deadline-critical flows.

Following are the main contributions of this work:

1. We describe an architecture that implements bandwidth calendaring for scientific workflows. The architectures leverage SDN switches that can pace flows at the ingress point. The architecture implements a central controller with distributed agents at the edge of the network that monitor flow performance and implement dynamic flow pacing set by the controller.
2. We present experimental results using Globus and GridFTP that show the importance of pacing in achieving deadline aware data transfer service. We compare our results with TCP Hamilton results.
3. We propose different heuristic algorithms based on combining two orthogonal principles - 1) local vs global optimization and 2) Shortest Job First vs Longest Job First (LJF). We perform a preliminary performance comparison of these algorithms with respect to a performance metric efficacy that is defined as the difference between reject rate and network utilization. Our results show that simple heuristics, that optimize locally on the most bottlenecked link can perform almost as well as heuristics that attempt to optimize globally.

The remainder of this paper is organized as follows. In Section 2, we discuss the motivation of our work, specifically the importance of deadline aware data transfers in scientific workflows. We also discuss the importance of pacing and traffic shaping in deadline-aware traffic flow-scheduling. In Section 3, we present the architecture of Calibers in a software defined network. In Section 5 we present experimental results on a preliminary prototype Calibers architecture to demonstrate the effectiveness in meeting deadlines compared standard TCP. In Section 6 we present work on a dynamic flow pacing algorithm and present preliminary simulation results. In Section 4, we present the related work followed

by conclusions and future work in Section 8.

2. Motivation

It is often the case that a large data transfer is inherently deadline-aware. For example, an HPC user may want to ensure that data is present at an HPC site before conducting their experiments. However, with the unpredictability of network resource utilization, this can pose a problem. Latency variation coupled with TCP's typical "sawtooth" behavior can lead to a lack of predictability in meeting deadlines. Furthermore, even when TCP achieves pareto-optimality, this behavior may not always be desired. For example, if one flow must complete faster over a bottleneck link than others, simply fairly sharing the available bottleneck bandwidth may not achieve the desired deadline goal.

Neuman et al. [11] highlighted the need to redesign I/O architecture and network links to cope with the performance of distributed science instruments. Salsano et al. [12] discussed various APIs to optimize packet movement based on user information to improve quality of experience. Also, OpenStack clouds have been investigating how workload can be balanced over geographically distributed data centers. However, to the best of our knowledge, there is still a lack of an implementation system which can demonstrate dynamic traffic shaping [4]. This paper aims to contribute to this research area by studying how flows can be dynamically paced to reduce loss and optimize link performance.

Towards, the above goal, we first demonstrate that even in a controlled, isolated environment, TCP does not provide predictable data transfer rate in the event of congestion. By pacing each data transfer properly, network congestion can be avoided and flow completion can be predicted. The following sections discusses the strategies and algorithm utilized to decide rate flows for transmitting data and how to achieve maximum number of successful flows with network utilization.

2.1. Pacing and Shaping Traffic

Active Queue Management (AQM) has existed on core network routers and switches for decades. In particular, Fair Queuing with Controlled Delay (FQ CoDel) has been implemented in such switches and routers for decades and was introduced in the Linux kernel in 2012 [13]. FQ implementations typically work on the principle of creating some data structure of ongoing flows, and then using a Deficit Round Robin (DRR) approach in order to dequeue packets from their respective buffers. In this fashion, a lightweight, but reasonably "fair" allocation of bandwidth resources occurs between multiple competing flows. CoDel implementations essentially work on the principle of putting hard limits on the real queue size of ongoing flows such that a particular latency target is met. In such a fashion, the *delay* of a flow is *controlled*, which has important implications for minimizing standing queue sizes elsewhere in the network and therefore delivering more predictable performance [6]. However, CoDel can also be used in conjunction with FQ to pace flows under a particular maximum rate without violating fairness principles. This is precisely how we conduct end-system pacing in Calibers.

3. Architecture

Workflow orchestrators provision resources across the network, with assumptions that it does not introduce performance penalties. Calibers is an experimental network service, targeted to higher level resource orchestrators. It focuses on optimizing network resources such that each data flows (i.e. file transfers) performs at least at the minimum average rate over the transfer duration. This allows Calibers to provide deadline delivery guarantees.

The experimental software platform designed in Calibers involves several components:

- A REST/JSON API: Orchestrators use this API to schedule file transfers. They provide source, destination, file size, deadline and maximum I/O rate of the endpoints of the transfer.
- An event publisher: Allows orchestrators to obtain real-time information on the maximum rate the network has allocated to data transfers without experiencing network congestion.
- SDN-based rate shaper: This component enforces flows to not exceed their bandwidth allocation.
- Calibers optimization algorithm: This dynamically adjusts the maximum rate of each flow, ensuring that all flows are on track to meet their respective deadlines. This in turn increases network utilization and maximizes the request admission rate.

Calibers aims to experiment with higher level of network services, file transfer deadlines and demonstrate new paradigms to higher network utilization. It makes several assumptions to realize these goals. These are not always true in a production environment. For example, Calibers assumes that endpoints are sufficiently provisioned with I/O, networking and processing resources.

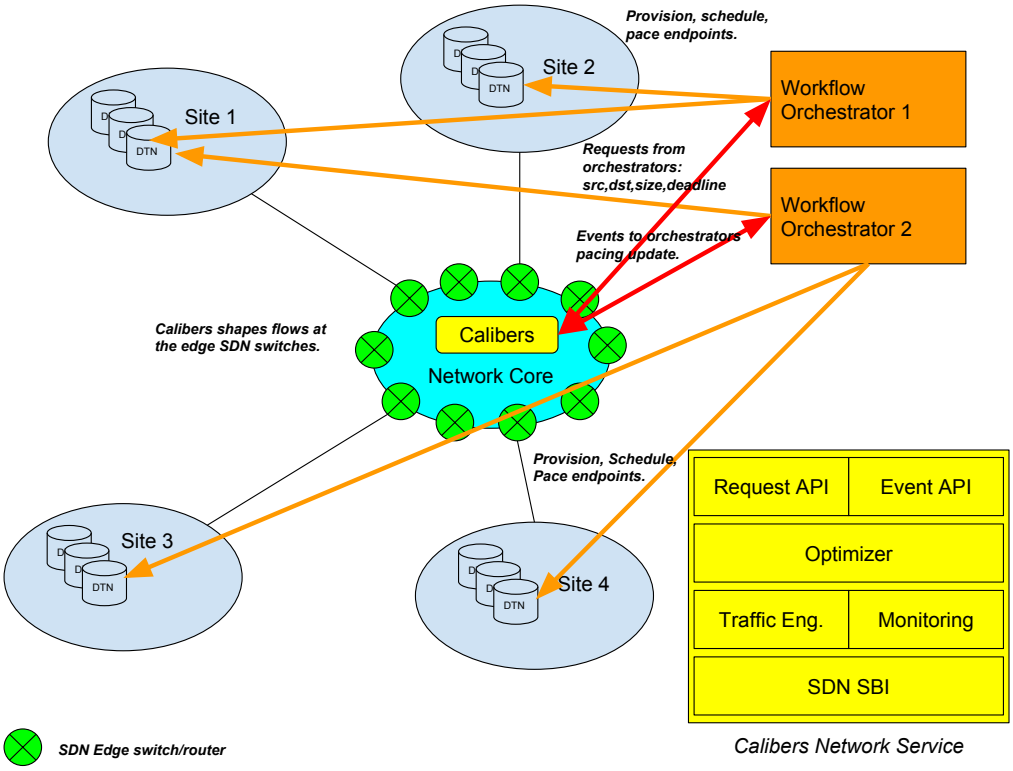


Figure 1: Calibers architecture illustrating the various components.

Another important assumption is that Calibers is given a minimum guaranteed capacity on the overall network and therefore prevents it to be impacted by non Calibers traffic.

The overall architecture of Calibers can be seen in 1. Each site may have one or more Data Transfer Nodes (DTNs). Calibers is leveraged by workflow orchestrators

When the flow pacing rate is dynamically changed at the network edge, it may result in packet loss which may result in throughput loss. Note that TCP algorithms such as H-TCP [2] and BBR-TCP [3] can quickly adapt to these changes. Additionally, new source pacing algorithms based on model predictive control [14] can also be used to ensure that the source quickly adapts to edge pacing rates.

4. Related Work

The over-arching goal of this work is to deliver deadline aware data transfers as a network service, while ensuring high network utilization. We leveraged SDN with the ability to perform dynamic traffic pacing at the network edge. There are a number of recent studies with similar goals. In the following paragraphs, we review the related work and point out the key differences from our work.

There has been a number of prior studies on flow pacing [15, 16, 17]. Broadly speaking, flow pacing can be performed at the source host or at the edge where the access network connects to the core network. The former is referred to as host pacing, or more commonly TCP pacing, while the latter is referred to as edge pacing and can be performed by the network service provider [17]. In this paper, we study edge pacing enabled softwareized SDN switches.

Software-Defined Networks allow dynamic and centralized traffic engineering (TE) via flow pacing. B4 [5] presents Google's effort in leveraging SDN to centralized TE and drive links to near full utilization. As similar study SWAN [18] also improves network utilization of inter-DC WAN by scheduling the service traffic in a centralized manner. However, all of these studies do not consider deadline associated with their transfers. The study in Tempus [7] considered deadlines and developed an optimization framework to maximize the frac-

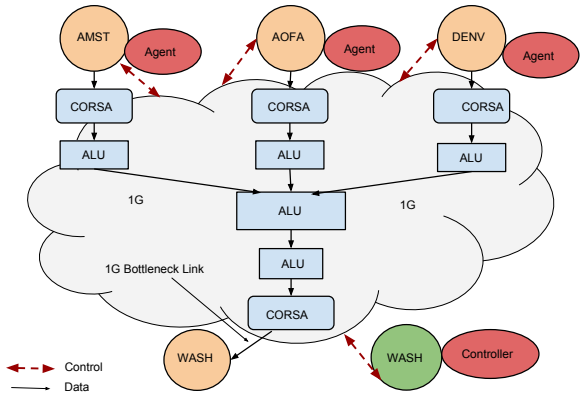


Figure 2: The experimental setup. Data transfer are initiated from Amsterdam (AMST), New York (AOFY) and Denver (DENV) to Washington (WASH). The Controller in a different VM in WASH can interact with the distributed agents to pace the flows. The objects labeled CORSA are SDN edge-switches, and the objects labeled ALU are core routers.

tion of transfer delivered before deadline, ensuring fairness among all requests. This work, however also does not guarantee meeting deadlines.

In a recent study [8], deadlines have been investigated in the context of inter-data center data transfers. Building on a deadline aware network abstraction (DNA) where transfer deadlines can be specified, the study proposes AMEOBA which uses traffic shaping at the source, to meet data transfer deadlines. While this study is the most similar to ours, there are a few key differences. First, this study focuses on scientific workflows. Second, we consider edge pacing. Finally, we show that simple dynamic pacing algorithms that optimizes locally on the most bottleneck link perform as well as more complex algorithms that attempt to optimize globally.

5. Experimental Results

The experimental setup in Figure 2 is based on ESnet's SDN Testbed, a high-speed Wide-Area Network (WAN) SDN-ready testbed spanning two continents. The backbone data rates are guaranteed and setup via dedicated OSCARS circuits.

The testbed closely resembles ESnet's high-speed production network in both hardware and topology, as it is an overlay of the ESnet production WAN [19]. Using ProxMox, a Linux container management system, we define three senders Amsterdam (AMST), New York (AOFY) and Denver (DENV) with varied round-trip latencies to one receiver at Washington (WASH). A controller container was provisioned at Washington (WASH). In order to simulate a real workflow, we use real FTP file transfers with 10 gigabyte data files rather than benchmarking tools. In particular, we used the Globus GridFTP file transfer tool, with the Globus API running on the controller, to synchronize the start of all three transfers [20]. Furthermore our topology closely simulates the case of a bottlenecked receiver obtaining data from three different senders with different geographical locations.

Our preliminary research was focused on determining the feasibility of pacing both at the network's edge with SDN-enabled Corsa switches [21] and within the end-system itself. In this fashion, the Corsa switches act to police bandwidth utilization below a certain threshold. Although this method of restricting bandwidth utilization will not necessarily result in maximum link utilization, this is important for preservation the overall bandwidth allocation. However, in our current implementation, losses at the Corsa switch, due to limits imposed on the queue size, will result in activation of the H-TCP congestion avoidance algorithm. In order to avoid this, pacing at the end-system is used. One could imagine in an active deployment that an end-user would either participate in pacing, or at least be policed and therefore unable to interfere with the pacing of other senders to a bottleneck link. Thus, the orchestrator is capable of limiting interference and preserving the overall workload allocation of the system.

Before we tested the feasibility of pacing with Globus and GridFTP, we conducted experiments testing the effects of pacing on `nuttcp` [22] flows sharing a bottleneck link. The improvement in network utilization can be seen in figures 3 and 4. Figure 3 demonstrates a flow HTCP handles high-RTT bottlenecked flows under normal conditions. Figure 4 shows the same flows through the same bottleneck with pacing. More importantly than utilization, the inherent predictability of pacing can also be seen. Figure 4 also shows a good example of flow *pacing*, where the end-system has been tasked with limiting its sending rate to a predetermined maximum. Recent advancements in pacing, including the effective deployment of the `fq_codel` algorithm in the Linux kernel, has allowed Linux end-systems to take advantage of highly accurate, low overhead flow pacing. As opposed to metering, pacing usually does not incur drops, as the feedback loop in the sending system kernel networking stack usually reacts before a drop occurs.

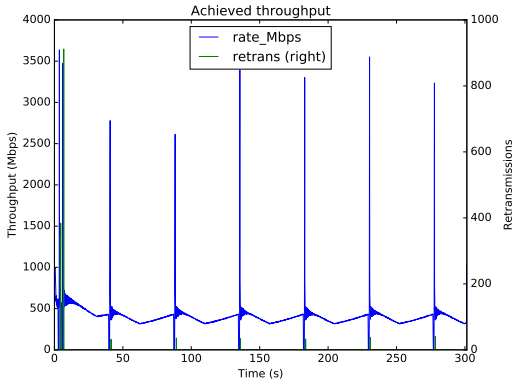


Figure 3: This experiment considers two H-TCP flows sharing a 1 Gbps link with both flows are un-paced. The spikes correspond to the buffer build up prior to loss events.

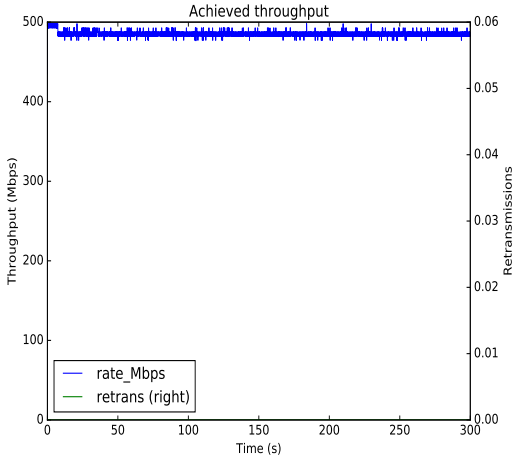


Figure 4: This experiment considers two H-TCP flows sharing a 1 Gbps link and both flows are paced at 500 Mbps.

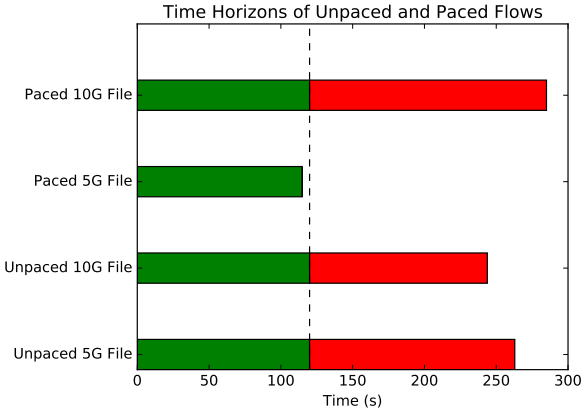


Figure 5: Flow completion times of three hosts sharing a bottleneck link. The vertical line is the deadline for the 5 GB file.

The pacing methods of Calibers can be demonstrated in a hardware SDN testbed scenario which mimics real-world challenges on a smaller scale with regard to throughput. Suppose a 5 GB file must be transferred in a critical window of 2 minutes. At the same time, a larger 10 GB file must be transferred, but with a much less strict deadline. They both share a 500 Mbps bottleneck link. If HTCP's "fair" behavior is utilized, the flows will actually complete in roughly the same time, as the differences in latency counteract the differences in file size. However, if Calibers paces the critical flow to 380 Mbps and the non-critical flow to 120 Mbps, the critical flow completes within its deadline, the available bottleneck bandwidth is relinquished, and the non-critical flow still completes in a similar amount of time. It is precisely this behavior of CoDel-based pacing which Calibers seeks to leverage. In Figure 5, the critical deadline for the 5 GB file is shown with a vertical dashed line. Both the paced and unpaced portions of the above experiment are shown. In both cases, the 5 GB and 10 GB files are transmitted simultaneously. In the unpaced case, much of the line rate is wasted as the two hosts, located thousands of miles from the bottleneck, attempt to negotiate their equal share of the link. In the paced case, however, the link sharing is far more efficient.

At this point, a long exposition of results of testing Calibers with various underlying transport protocols could be undertaken, as Calibers is completely transport-protocol agnostic (it works entirely at the network and link layers of the protocol stack). Various transport protocols achieve different degrees of fairness, and all can be paced and metered by Calibers in a similar manner.

6. Scheduling Algorithm for Dynamic Pacing

The notations used are shown in Table 1. The objective of the scheduler is to decrease the number of rejected data transfer requests while increasing the network utilization. This can be achieved by minimizing the sum of the completion time of all flows. This will push the completion time of all flows to the left (considering a time line), which increases the link utilization and frees up resources to accommodate future requests. The objective function is given by

$$\min \sum_{f_i \in U} t^c_{f_i} \quad \text{subject to}$$

$$\sum_{f_i \in l^F_i} R^{alloc}_{f_i} \leq l^C_i \forall l_i \in involved\_links$$

$$t^c_{f_i} \leq t^d_{f_i} \forall f_i \in involved\_flows$$

Table 1: Notation used.

Symbol	Description
t_{now}	Current scheduling epoch
U	Set of requests
u_i	User i request which is defined by 5 tuples $(IP_{src}, IP_{dst}, S, t^d_{f_i})$ where IP_{src} and IP_{dst} are the source and destination IP addresses, S is the data size, and t_d is the deadline
F	The set of the currently scheduled flows
f_i	flow i $\in F$
p_i	A path p_i is sequence of links corresponds to f_i source and destination pair
$R^{min}_{f_i}$	For flow f_i, this is the minimum rate that will guarantee that the deadline is met. This is $S/(t_d - t_{now})$
$R^{alloc}_{f_i}$	The rate allocated for flow f_i
$R^{slack}_{f_i}$	The slack rate assigned to flow f_i, $R^{slack}_{f_i} = R^{alloc}_{f_i} - R^{min}_{f_i}$
L	The set of the links in the network
l_i	Link i in the network, $l_i \in L$
l^C_i	Link l_i capacity
l^F_i	List of flows span link l_i
$R^{resid}_{l_i}$	The residual capacity for link l_i. $R^{resid}_{l_i} = l^C_i$ - $\sum_{\forall f_i \in l^F_i} R^{alloc}_{f_i}$
$l_{bottleneck}$	The link with the flow that has the maximum t_c
$t^c_{f_i}$	The completion time of flow f_i, which depends on $R^{alloc}_{f_i}$

where *involved_links* is the set of links that the new requests traverse, and *involved_flows* is the set of flows that span the *involved_links*. As the number of requests and the size of the network increases, the solution to the objective function is not guaranteed to converge. Hence, we propose four heuristic schedulers that also aim to minimize the sum of the completion time of the flows.

The scheduler operates at fixed discrete epochs with the following assumptions: (i) each link has a free capacity l^C_i to be used by the scheduler, (ii) start time for each data transfer request is immediate, i.e., the scheduler does not support advance reservation, and (ii) the scheduler updates the network status periodically every scheduling interval (epoch).

The scheduling problem is divided into to subproblems: (i) *new flow*: when a new request arrives, how to decide whether to accept or reject the request? and (ii) *completed flow*: when a request completes, how to distribute the free capacity among the ongoing flows? We study four heuristic algorithms by combining two concepts: (i) global and local optimization and (ii) Shortest Job First (SJF) and Longest Job First (LJF).

In the global approach, the scheduler consider all the flows when distributing any residual capacity. On the other hand, the local approach focus on the bottleneck links in the network and distribute the residual capacity by reallocating locally only the flows that span the bottleneck link(s). The LJF and SJF are known concepts where longest jobs are favored with LJF and shortest jobs are favored when SJF is used. Those concepts are used by both the global and local scheduler in the following way. When the scheduler decides (locally or globally) which flows should be considered when distributing the residual capacity, SJF or LJF will be used to decide the order in which the flows will be assigned the residual capacity.

The scheduler keeps track of multiple parameters as shown in Table 1. One of the most important parameter the scheduler uses to make decision is $R^{min}_{f_i}$, which is the minimum required rate to ensure the flow f_i will not miss its deadline. The pseudo-code of both the global and local-approach is provided in Alg. 1. Both schedulers use the same approach when a new request arrives, however, they differ in the way the capacity is redistributed when a request completes.

6.1. Approach 1: Global Optimization

Sub-problem 1: *new flow*, when a new flow f_i corresponding to request u_i arrives, the scheduler computes $R^{min}_{f_i}$, and checks if the residual bandwidth in the flow path $R^{resid}_{p_i}$ is greater than $R^{min}_{f_i}$, it assign $R^{resid}_{p_i}$ to the new flow as shown in line 2-8 of Alg. 1. The scheduler gives the maximum available rate to the new request instead of giving it $R^{min}_{f_i}$ for two reasons: (i) to increase the link utilization, and (ii) to complete the file transfer as soon as possible in order to free up the resources to accept

future requests. If $R_{f_i}^{min}$ is not available, the scheduler move to the second phase, which is pacing other flows in order to accept the new flow.

The pacing phase is shown in line 10-26 of Alg. 1, where for each link l_i in the path p_i of the flow f_i, the scheduler finds the list of flows l_i^F span link l_i. SJF or LJF concepts are used to decide which flow(s) of the list l_i^F to pace (slow down). When using SJF, the scheduler favors short flows with longest flows being slowed down first and vice versa. The scheduler paces the first flow in the sorted list by taking its slack rate $R_{f_j}^{slack}$ and assigns it to the new flow. If the first flow slack ($R_{f_j}^{slack}$) in the sorted list is less than the new flow required minimum rate ($R_{f_i}^{min}$), then the scheduler takes the slack rate of the second flow in the sorted list until the sum of the slack rates is equal to the new flow required minimum rate, or until there are no more flows in the sorted list. Hence, the request will be rejected because even with pacing, $R_{f_i}^{min}$ cannot be assigned to the new flow.

Sub-problem 2: *completed flow*, at the beginning of each epoch, the scheduler checks if a flow has completed by checking the flow completion time t_c. The flow completion time is a dynamic parameter, that changes based on the allocated rate ($R_{f_i}^{alloc}$). For all the flows completed at the scheduling epoch, the scheduler traverse the path of each completed flow and finds the set of other flows, that span the links in the path (involved_flows). After finding all involved flows, the scheduler now has a global view and starts distributing the residual capacity using SJF or LJF concepts. For example, if SJF is used, the scheduler finds the flow with the shortest completion time and assign to it the residual bandwidth available in its path then move to the next shortest flow and so on. The procedure of the global optimization approach is shown in Alg. 1, line 40-45.

6.2. Approach 2: Local Optimization

As mentioned earlier, the same approach is used when a new request arrives. However, the local scheduler takes a different approach when a flow completes.

Sub-problem 2: *completed flow*, at the beginning of each epoch, the scheduler checks if a flow has completed. The scheduler finds the bottleneck link ($l_{bottleneck}$) from the paths of all completed flows. The link which has a flow with the maximum $t_{f_i}^c$, is the link that will stay busy the longest, hence, is the bottleneck link that might cause future requests to be rejected. If multiple links have a flow with the same maximum $t_{f_i}^c$, then the link with the highest average $t_{f_i}^c$ (without considering the maximum $t_{f_i}^c$) is decided to be the bottleneck link. By freeing up only the bottleneck link the probability of accepting flows in the future increases. The scheduler considers only the flows spanning the bottleneck link when distributing the residual capacity, which in contrast to the global approach, where scheduler considers all flows spanning all links of all completed flows paths. The procedure of the local optimization approach is shown in Alg. 1, line 46-53.

6.3. Algorithm complexity

The complexity of the worst-case scenario for new flows is $O(FL)$ for both the global and local optimization approaches as both algorithms follow the same approach when a new request arrives. When a new request arrives, the algorithm finds the new flow path, then it goes through all the flows traversing each link of the path to find if a residual bandwidth is available. The worst-case is when the path of the new request includes all the links (L), and each link is traversed by all the flows (F).

In the reshape phase when a flow is completed, both the local and global optimization approaches have a worst-case complexity of $O(F^2L)$. The worst-case scenario in the global approach is when the completed flow path consists of all the links of the network (L), which means all the flows (F) are considered for reshaping. Hence, for each flow f_i in F, each link $l_i$in the path p_i of f_i is visited, and each flow traverses l_i is visited to find the residual bandwidth (if any), which yields to a complexity of $O(F^2L)$.

In the local optimization approach, even though only the bottleneck link is considered, the worst-case scenario occurs when the bottleneck link is traversed by all the flows on the network (F). Thus, all the flows will be considered for reshaping which would result on the same complexity of the global optimization. However, as the size of network increases, the probability of having all the flows (or large number of flows) traversing the bottleneck link is lower compared to the global approach where all the flows traversing the paths (many links) of the completed flow(s) are considered.

7. Simulation Analysis

Flow-level simulation was conducted to evaluate the performance of the four schedulers: (i) local-SJF, (ii) local-LJF, (iii) global-SJF, and (iv) global-LJF. The simulator was written using Python and for each simulation setup, 10 runs were executed where in each run 30k requests were generated.

Algorithm 1: Dynamic Pacing Scheduler

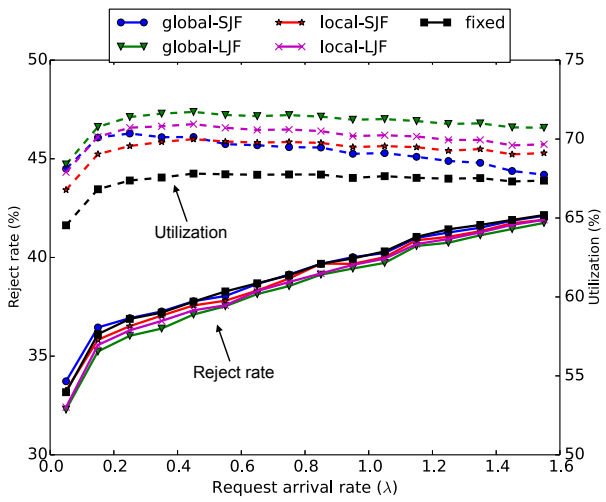
Input: U
remove_completed_flows(t_now)
foreach $u_i \in U$ **do**
  $R_{p_i}^{resid} = min(R_{l_i}^{resid}\ \forall l_i \in p_i)$
  if $R_{p_i}^{resid} < R_{f_i}^{min}$ **then**
    pace(f_i)
  else
    $R_{f_i}^{alloc} = min\{R_{f_i}^{max}, R_{p_i}^{resid}\}$
    return success
Function *pace(f_i)*
  $\forall l_i \in p_i$
  $R_{temp} = 0$
  $R_{temp}^{p_i} = []$
  found = False
  involved_flows = l_i^F
  sorted_involved_flows = sort the list in ascending (if LJF) or descending (if SJF) order based on t_c
  foreach $f_j \in sorted\_involved\_flows$ **do**
    $R_{temp} = R_{temp} + R_{f_j}^{slack}$
    if $R_{temp} \geq R_{f_i}^{min}$ **then**
      found = True
      add R_{temp} to $R_{temp}^{p_i}$
      break
  if *found = True* **then**
    $R_{f_i}^{alloc} = min(R_{temp}^{p_i})$
    return success
  else
    return reject
Function *remove_completed_flows()*
  involved_flows = []
  involved_links = []
  foreach $f_i \in F$ **do**
    if $t_c = t_{now}$ **then**
      $\forall l_j \in p_i$
      remove f_i from l_j^F and F
      add l_j to involved_links
      add to l_j^F involved_flows
    if *local-sched* **then**
      local_reshape(involved_links)
  if *global-sched* **then**
    global_reshape(involved_flows)
Function *global_reshape(involved_flows)*
  $R_{f_i}^{alloc} = R_{f_i}^{min}\ \forall f_i \in$ involved_flows
  sorted_involved_flows = sort the list in ascending (if SJF) or descending (if LJF) order based on t_c
  foreach $f_i \in sorted\_involved\_flows$ **do**
    $R_{p_i}^{resid} = min(R_{l_i}^{resid}\ \forall l_i \in p_i)$
    $R_{f_i}^{alloc} = R_{f_i}^{alloc} + R_{p_i}^{resid}$
Function *local_reshape(involved_links)*
  find $l_{bottleneck}$
  $R_{f_i}^{alloc} = R_{f_i}^{min}\ \forall f_i \in l_{bottleneck}$
  involved_flows = $l_{bottleneck}^F$
  sorted_involved_flows = sort the list in ascending (if SJF) or descending (if LJF) order based on t_c
  foreach $f_i \in sorted\_involved\_flows$ **do**
    $R_{p_i}^{resid} = min(R_{l_i}^{resid}\ \forall l_i \in p_i)$
    $R_{f_i}^{alloc} = R_{f_i}^{alloc} + R_{p_i}^{resid}$

7.1. Simulation Setup

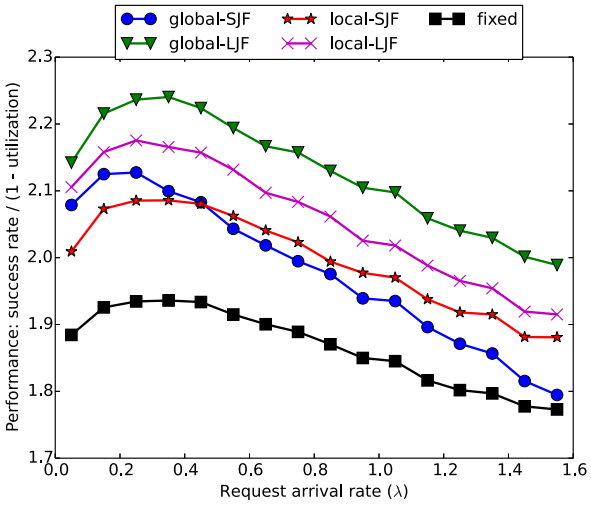
Network. Google's inter-data center network G-scale [5] with 12 nodes and 19 links was used to evaluate. The link capacity l_i^C was set to 10 Gbps for all links in the network.

Workload. Requests were generated as follow: (i) Request inter-arrival time was modeled with an exponential distribution with arrival rate λ varying between 0.05 to 1.6 with step of 0.1, i.e. the mean inter-arrival time between requests varies from 20 sec to 0.625 sec. (ii) Request deadline time was modeled following an exponential distribution with average deadline (t_d) of 1 hour. (iii) As the file size is related to the deadline, it was modeled as follows [8]. First, a transfer rate (i.e. $R_{f_i}^{min}$) is sampled following an exponential distribution with average rate of 100 Mbps. Next the file size was computed as transfer rate $\times$ t_d. This results in a product distribution with a mean file size of 45 GB. (iv) Source and destination pairs were picked uniformly.

Metrics. Three performance metrics were measured: (i) Network utilization is computed by measuring the link utilization per second for each link in the network $l_i^{utilization}(t)$, then taking the average utilization for each link across the entire simulation time ($\forall l_i \in L$, $L_i^{utilization}$ = mean($l_i^{utilization}(t)$)). Finally, the network utilization is measured as the average of $L_i^{utilization}$ for all the links in the network (network utilization = mean[$L_i^{utilization}$ $\forall l_i \in L$]). (ii) Reject rate which is defined as the number of rejected requests divided by the total number of requests. (iii) Performance index is defined as the ratio of the request success rate to the wasted bandwidth (Performance index = $\frac{success\ rate}{(1-Utilization)}$).



(a) Reject rate and network utilization



(b) Performance index

Figure 6: Performance comparison of the four algorithms for the G-scale network.

7.2. Results

Figure 6 shows the performance of five schedulers for the G-scale network with mean file size of 45 GB (transfer rate of 100 Mbps) and an epoch of 3 minutes. The four heuristic schedulers are compared against a fixed scheduler which assigns R_{min} for each flow and does not dynamically change any flow rate.

Figure 6a shows that the reject rate increases as the request arrival rate increases. Also, the global-LJF has the highest utilization while the fixed scheduler has the lowest utilization because the fixed scheduler assigns R_{min} for each flow and does not utilize the residual bandwidth.

Figure 6b shows the performance difference between the five schedulers where the fixed scheduler is achieving the lowest performance. For example, with the low arrival rate of 0.05, the global-LJF scheduler outperformed the fixed scheduler by 20% (2.16-1.80)/1.80).

Many observations can be made on the performance of the four heuristic schedulers, *first*, while the request arrival rate increases, the performance increases until the arrival rate reaches 0.4 where the performance starts decreasing. This is because at a low arrival rate, the number of requests arriving is low, therefore, the link utilization is low. However, as the arrival rate increases, the reject rate increases, which yields to performance degradation.

Second, the global-LJF has a slightly higher performance compared to the local-LJF. The global scheduler consider all the flows when distributing the residual bandwidth. On the other hand, the local scheduler consider only the flows spanning the bottleneck link, which results on smaller number of considered flows. If the considered flows cannot utilize the residual bandwidth (because of other bottleneck links in their paths), then the residual bandwidth is wasted, hence, the performance decreases. With the simulated network, the performance difference between global-LJF and local-LJF is negligible, where global-LJF outperformed local-LJF by only 3%. This shows that redistributing the capacity only in the bottleneck link of the path of the completed flow, is enough to perform as good as when considering all flows traversing the path.

Third, by comparing global-SJF against global-LJF, or local-SJF against local-LJF, we can conclude that LJF schedulers have a better performance. LJF reduces the makespan time of all flows by assigning more rate to the flows with the longest completion time. This results on freeing up the links faster to accommodate future requests. On the other hand, SJF frees up some capacity of the link earlier than LJF but the makespan of the flows stays the same. Also, since SJF favors the flows with the lowest completion time when redistributing the residual capacity, then the probability of the flow finishing during the epoch is higher compared to if LJF is used. Therefore the flow will finish during the epoch and the capacity used by the completed flow will be wasted as no reshaping is done within the epoch. To further explore the performance difference between SJF and LJF, we executed both LJF and SJF schedulers with

a small epoch duration of 1-sec. The results showed a negligible performance difference between SJF and LJF. SJF assigns more bandwidth to the shortest flows, which could result on the completion of the shortest flows at the beginning of the epoch. However, since the epoch is small (1-sec), only a fraction of the bandwidth could be wasted because in the next epoch (after 1-sec) the scheduler will redistribute the residual bandwidth, which increases the performance of SJF.

In summary, the local optimization approach performs as good as the global optimization. The performance when using SJF versus LJF is negligible in short epoch duration (e.g., 1-sec), but the performance improvement shows with longer epoch duration, therefore, we choose local-LJF as the best heuristic for scheduling compared to the other three heuristics.

8. Conclusions and Future Work

Calibers has demonstrated, in an ideal situation of a controlled environment, that TCP congestion avoidance algorithms, while performing well at maximizing network utilization, it cannot provide the desired behavior for workflow orchestration. In particular, TCP relies on network characteristic, such as RTT, packet retransmission, pace flows and ignore flow needs. As result some flows may go faster than they need, while others may go slower than they should, such to meet deadlines and maximize resource utilization. However, the performance of modern TCP, in conjunction with the ability of SDN to implement centralized traffic engineering, allows Calibers to optimize network utilization to provide predictable performance. Our preliminary study on dynamic pacing algorithms suggests that simple heuristics that optimize locally on the most bottlenecked link can perform almost as well as attempts to optimize globally.

In our future work, we will address some of the assumptions that we made in this preliminary work. In particular, we will consider the interaction between deadline aware traffic and background traffic. We will investigate methods to predict the background traffic and leave appropriate network capacity to minimize the impact on the deadline aware traffic. This work will also enhance Calibers to accept deadline aware data transfer request at a future time.

[1] ESnet, Oscars: On-demand secure circuits and advance reservation system `https://www.es.net/engineering-services/oscars/`.

[2] D. Leith, R. Shorten, H-tcp: Tcp congestion control for high bandwidth-delay product paths, draft-leith-tcp-htcp-06 (work in progress) (2008).

[3] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, V. Jacobson, Bbr: Congestion-based congestion control, Queue 14 (2016) 50.

[4] DOE, Doe network 2025: Network research problems and challenges for doe scientists workshop (2016).

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, B4: Experience with a globally-deployed software defined wan, SIGCOMM Comput. Commun. Rev. 43 (2013) 3–14.

[6] J. Gettys, K. Nichols, Bufferbloat: Dark buffers in the internet, Queue 9 (2011) 40:40–40:54.

[7] S. Kandula, I. Menache, R. Schwartz, S. R. Babbula, Calendaring for wide area networks, in: Proceedings of the 2014 ACM Conference on SIGCOMM, SIGCOMM '14, ACM, New York, NY, USA, 2014, pp. 515–526.

[8] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, M. Zhang, Guaranteeing deadlines for inter-data center transfers, IEEE/ACM Trans. Netw. 25 (2017) 579–595.

[9] K. KINOSHITA, M. AIHARA, S. KONO, N. YAMAI, T. WATANABE, Joint bandwidth scheduling and routing method for large file transfer with time constraint and its implementation, IEICE Transactions on Communications E101.B (2018) 763–771.

[10] M. Aihara, S. Kono, K. Kinoshita, N. Yamai, T. Watanabe, Joint bandwidth scheduling and routing method for large file transfer with time constraint, in: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, pp. 1125–1130.

[11] H. Newman, A. M. I. Legrand, R. Voicu, D. Kcira, J. Bunn, High speed scientific data transfers using software defined networking, in: Innovating the Network for Data Intensive Science (INDIS), SC 15.

[12] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, L. Veltri, Information centric networking over sdn and openflow: Architectural aspects and experiments on the ofelia testbed, in: Int. Jour. Comp. and Tele. Net. archive, 57(16), 3207-3221.

[13] D. Raghuvanshi, B. Annappa, M. Tahiliani, On the effectiveness of codel for active queue management, in: Advanced Computing and Communication Technologies (ACCT), 2013 Third International Conference on, pp. 107–114.

[14] D. Fridovich-Keil, N. Hanford, M. Chapman, C. Tomlin, M. Farrens, D. Ghosal, A model predictive control approach to flow pacing for tcp, in: 55th Annual Allerton Conference on Communication, Control, and Computing.

[15] A. Aggarwal, S. Savage, T. Anderson, Understanding the performance of tcp pacing, in: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 3, IEEE, pp. 1157–1165.

[16] D. Wei, P. Cao, S. Low, C. EAS, Tcp pacing revisited, in: Proceedings of IEEE INFOCOM.

[17] H. H. Gharakheili, A. Vishwanath, V. Sivaraman, Comparing edge and host traffic pacing in small buffer networks, Computer Networks 77 (2015) 103 – 116.

[18] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven wan, in: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, SIGCOMM '13, ACM, New York, NY, USA, 2013, pp. 15–26.

[19] Energy Sciences Network, ESnet 100G SDN Testbed `http://es.net/network-r-and-d/experimental-network-testbeds/100g-sdn-testbed`.

[20] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, The globus striped gridftp framework and server, in: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, SC '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 54–.

[21] CORSA, SDN switches, `http://www.corsa.com/solutions`.

[22] B. Fink, R. Scott, nuttcp, v8.1.4, 2016.