The Performance of Distributed Applications: A Traffic Shaping Perspective

Jasper A. Hasenoot
jasper@hasenoot.com
Leiden University
The Netherlands

Jan S. Rellermeyer
rellermeyer@vss.uni-hannover.de
Leibniz University Hannover
Germany

Alexandru Uta
a.uta@liacs.leidenuniv.nl
Leiden University
The Netherlands

ABSTRACT

Widely used in datacenters and clouds, network traffic shaping is a performance influencing factor that is often overlooked when benchmarking or simply deploying distributed applications. While in theory traffic shaping should allow for a fairer sharing of network resources, in practice it also introduces new problems: performance (measurement) inconsistency and long tails. In this paper we investigate the effects of traffic shaping mechanisms on common distributed applications. We characterize the performance of a distributed key-value store, big data workloads, and high-performance computing under state-of-the-art benchmarks, while the underlying network's traffic is shaped using state-of-the-art mechanisms such as token-buckets or priority queues. Our results show that the impact of traffic shaping needs to be taken into account when benchmarking or deploying distributed applications. To help researchers, practitioners, and application developers we uncover several practical implications and make recommendations on how certain applications are to be deployed so that performance is least impacted by the shaping protocols.

CCS CONCEPTS

• **Networks** → Network reliability.

KEYWORDS

networked application performance; traffic shaping

ACM Reference Format:
Jasper A. Hasenoot, Jan S. Rellermeyer, and Alexandru Uta. 2023. The Performance of Distributed Applications: A Traffic Shaping Perspective. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3578244.3583733

1 INTRODUCTION

Cloud performance is highly volatile [26] either due to hardware fail-slow phenomena [38], noisy neighbors [7] or interactions with providers policies [82]. The latter are commonly instantiated by cloud providers to maximize utilization of available hardware resources, such as servers or networks, while ensuring that a minimum Service Level Agreement (SLA) is met. This allows for the maximization of profits at minimal cost for the provider.

ICPE '23, April 15–19, 2023, Coimbra, Portugal
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0068-2/23/04.
https://doi.org/10.1145/3578244.3583733

Cloud providers can generally consistently provide RAM [75] and – to a lesser degree – (v)CPUs [27, 75], however, the available network bandwidth is only shown as a *maximum available* amount [32, 80], which is not guaranteed [33, 75, 80]. While it is possible to provide a minimum bandwidth guarantee, in practice this impacts over-subscription [7] and is therefore not specified by current cloud providers. As a result, the end-user performance predictability is severely compromised [75, 78], as in practice performance distributions are long-tailed [18, 82].

To ensure a fair sharing of network resources between tenants, and thus ensuring a Quality of Service (QoS), cloud providers utilize traffic shaping techniques like priority queues and token buckets. The former allow for prioritizing certain network flows over others, causing an imbalance between flows of different priorities. This could be used to ensure that latency-sensitive traffic flows (e.g. Voice over IP; VoIP) get processed with a lower latency, for example [10]. The introduction or removal of such flows may therefore cause sudden changes in network performance – provided the network is nearing congestion – when using a traffic flow with a lower priority. The latter allows for limiting the available bandwidth to each host, by limiting the amount of data able to be sent in a certain time frame. The token bucket refills at a preset rate, up to a maximum buffer [42, 43]. This allows for short bursts of high bandwidth traffic – which appear uncapped – provided there is little network usage in between, allowing for the bucket to refill.

Variance and inconsistency in network performance due to multi-user resource sharing across servers and networks [75], combined with over-subscription on network links [34] as well as traffic shaping [82], can impact the performance of applications relying on the consistency of these resources [6]. This, in turn, impacts any performance evaluations or generally experiments, which may lead to unfair, unrepeatable or incorrect results. Evaluation of distributed applications *without* regard for the impact of variable network performance therefore may not be representative of the performance in a real-world cloud environment with shared resources.

Since cloud users and experimenters have no control over those mechanisms, their impact on application performance is difficult to assess. As a consequence, our community lacks a deep understanding of the effects of traffic shaping on cloud-based distributed applications. In this article we showcase and quantify the severity of such effects on a variety of workloads with the goal of aiding practitioners, experimenters and cloud providers with novel insights on how applications interact with traffic shaping and giving practical advice on how to counteract these effects.

We create a comprehensive benchmarking and experimental harness using setups similar to state-of-the-art cloud environments, with physical and virtualized resources (e.g. switches), as well as commonly used traffic shaping techniques. The addition of a suite of distributed applications covering multiple domains (e.g., key-value store, big data, and high-performance computing) allows for real-world representative results to be obtained.

We enforce shaping techniques using either dedicated network equipment, such as switches, or on-host virtual appliances. The former allows for a more granular control over network flows [64]. The latter is achieved using a kernel-bypass network processor such as the Data Plane Development Kit (DPDK) [22]. Our experiments are deployed on CloudLab [25], which allows to create dedicated links between nodes as well as user-managed switches to apply the QoS traffic shaping measures.

This article makes the following contributions: **(1)** We design a novel experimental setup to uncover the interaction between state-of-the-art distributed applications and traffic shaping. Our application-independent framework[1], is tailored to CloudLab but adaptable to a broader context. **(2)** After running over a thousand individual experiments and analyzing application behavior, we provide novel insights into the effect of traffic shaping on distributed applications spanning different domains. **(3)** We give recommendations and actionable insights for the benchmarking of distributed applications in the context of traffic shaping, as well as determining the optimal environment when deploying them.

2 BACKGROUND: CLOUD NETWORKS

The Software Defined Networking (SDN) [50] paradigm has forever changed the design of cloud- and datacenter networks. SDN separates the data plane – which processes network traffic – from the control plane, which configures it. Via this decoupling, the control of the network can be centralized for rapid deployment. Network equipment supporting SDN may either be hardware-based or software-based, like Open vSwitch [71] (OVS). Most modern clouds nowadays implement SDNs [16, 30] to enable better control over the networking infrastructure. The control plane configures the SDN switches through OpenFlow [60], specifying "flows" through the network based on packet header data [72], which may optionally be modified [65]. The Data Plane Development Kit (DPDK) [22] allows for network packet processing to bypass the kernel entirely, providing speed at the cost of reduced configurability. Combining this with OVS into OVS-DPDK [71] alleviates some of this cost, as OVS can be configured using OpenFlow [60] at runtime.

Quality of Service & Traffic Shaping. Quality of Service allows for distinguishing between applications based on communication requirements [10, p. 6]. Network flows can be treated with different priorities or be given more or less bandwidth. This in turn can also influence ping or jitter [10] In practice, bandwidth between flows is divided using priority queues, with bandwidth caps being enforced through token buckets [12, 19]. In public Cloud environments this increases fairness and gives rise to multiple bandwidth tiers [32, 80].

Priority Queue. Priority Queues divide bandwidth between traffic flows using strict and/or (weighted) fair queues [44]. The former allows high priority flows to starve other flows by using all available bandwidth [44] The latter shares bandwidth between flows using weights [68], with higher priority resulting in a larger bandwidth share. The flow priority is stored in the IP header's DSCP [36] field. It has four priority tiers and three "drop probability" tiers. This is used to drop packets if the available bandwidth is exceeded [83].

Token Bucket. Token Buckets are used to limit the maximum flow bandwidth. They are defined by the Committed Burst Size (CBS) and Committed Information Rate (CIR): the bucket size and refill rate, respectively [42, 43]. Both are defined in bits or packets ("tokens") within this context [42, 43]. Traffic transmitted on a flow does not exceed the specified CIR on average [42, 43], though it can be exceeded temporarily. Transmitted traffic subtracts "tokens" from the token bucket proportionally to the packet size or count [42, 43]. If traffic rates exceed the CIR, the built-up tokens in the bucket are first depleted before the traffic is shaped down to the CIR [42]. Adding a second token bucket, defined by the Excessive Burst Size (EBS) and Peak Information Rate (PIR) [43], allows for a larger burst where traffic is sent with a higher (DSCP) drop probability once it exceeds CIR. This way more bandwidth may be utilized. The PIR is only used if remaining available bandwidth allows it, the CIR remains as the minimal bandwidth that is guaranteed [43].

3 EXPERIMENT DESIGN: HOW TO QUANTIFY TRAFFIC SHAPING EFFECTS ON DISTRIBUTED APPLICATIONS

To understand the implications of traffic shaping on modern cloud-based applications, we design a set of large-scale, extensive experiments. The main goals of this article and of the design of our large-scale experiments are:

(1) Designing a networking setup that is realistic, akin to modern cloud datacenter networks.
(2) Implementing state-of-the-art traffic shaping techniques on top of the aforementioned networks.
(3) Running a suite of modern, state-of-the-art benchmarks and applications against state-of-the-art traffic-shaped cloud networks.
(4) Understanding how traffic shaping mechanisms, policies, and parameters influence the aforementioned modern benchmarks and applications.

To understand the interaction between traffic shaping and workloads, we measure achieved performance against or relative to a baseline without traffic shaping or even without interference traffic. Next to this, we assess performance `consistency`. We define as `consistent` the performance which exhibits low variability, and as `inconsistent` highly-variable performance, e.g. with long tails.

We use the CloudLab [25] platform, which allows users to reserve nodes and create private (user-managed) networks. We do this through a user-shareable profile, which contains the number and type of nodes used, the connecting network and other resources, such as storage. We provide each node with a pre-configured Docker

Node: xl170	
CPU	10-core Intel E5-2640v4 (2.4GHz)
RAM	64GB ECC DDR4-2400 (4x 16GB)
Disk	Intel DC S3520 480 GB 6G SATA SSD
NIC	Two Dual-port Mellanox ConnectX-4 25 GB NIC (PCIe v3.0, 8 lanes)

Table 1: The xl170 node [13] on CloudLab [25].

[1]available at https://github.com/JasperAH/ts-ds-resources

image, which – together with the profile – allows us to create identical experiment environments for our experiments, enabling reproducibility. For the networking setup, we describe a solution in the form of a Docker overlay network which uses OVS-DPDK to model the behavior of common public clouds on a regular network, including the generation of interference traffic generation and traffic shaping settings. We conduct all experiments using Docker containers backed by our OVS-DPDK setup as this allows us to apply traffic shaping both on-host and on-switch, both with (network-)resource contention present. In all experiment setups we use the xl170 nodes, described in Table 1. These contain Mellanox ConnectX-4 NICs, which support DPDK [63]. They are connected to a user-managed Mellanox MSN2410-BB2F switch using 10Gb/s Ethernet links which are allocated via a NetScout 3903 L1 switch managed by CloudLab.

3.1 Docker Overlay Network

We use the Docker overlay network as the basis for our experiments, which is backed by OVS-DPDK on each host. This virtual network lays on top of the physical user-managed network provided by CloudLab. We use a single Consul [40] cluster store instance, which tracks the network status across nodes, as well as a Docker and OVS-DPDK instance on each node [2]. These instances communicate with Consul over a separate control network. This way we create a realistic setup, similar to those found in Cloud environments, where there is a combination of an on-host (kernel-bypass) and a physical network, with on each node a system that allows for the creation of virtualized environments [17, 47].

When creating virtual networks or adding containers to Docker, all network-related events are passed to the OVS-DPDK instance through the use of an OVN Docker overlay driver [79] on each host. This enables regular Docker commands to manage the network. The driver translates the network events to OpenFlow, used to control OVS-DPDK. We slightly modified the overlay driver to make it compatible with current library versions and Python3.

Using this setup, we can automatically assign IP addresses within the overlay network IP pool to our containers upon creation, which allows them to communicate using the user-managed network. To allow our containers to automatically connect, we first create an OVS-DPDK backed network in Docker, and specify its name in the `docker-compose` file for each container.

3.2 Interference Traffic

Since traffic in networked environments varies over time, we created interference traffic in our experimental setup that exhibits similar behavior. We based our traffic pattern on the traffic generated in the experiments simulating cloud environments ran by Ballani et al. [6]: our interference traffic is normally distributed around $\frac{\text{max\_bandwidth}}{2}$. To make sure our higher and lower interference bandwidths are used more frequently, we choose our standard deviation such that our distribution is slightly wider than the feasible values between $[0, \text{max\_bandwidth}]$.

To generate the interference traffic according to this distribution we use Iperf3 [24], which batches packets to create small (bidirectional) bursts of traffic. We use each sampled bandwidth for 5 seconds to account for wind-up time between switching bandwidths, after which we sample a new value and repeat the process.

Experiment Profile	**Description**
No Interference	No changes
Normally Distributed Interference	Normally distributed interference traffic on node interfaces
Token Bucket	Token bucket on node interfaces in addition to interference
Priority Queue	Priority queue alternating between high and low priority on node interfaces in addition to interference
Token Bucket & Priority Queue	Combination of the above

Table 2: Experiment profiles with regard to traffic shaping and interference. We execute experiments on each of the profiles above.

By carrying out experiments multiple times or over a prolonged period we ensure the setup is exposed to a range of bandwidths.

3.3 Traffic Shaping

We use the same set of traffic shaping profiles in all of our experiments, described in Table 2. The interference traffic we described in Section 3.2 is present in all profiles except the "no interference" profile. We apply the token bucket and priority queue settings to the switch ports connected to all nodes, with the priority queue settings having high and low priority for different ports.

Priority queues on-switch work on a port-by-port basis. Therefore we cannot subject traffic emerging from a node to different QoS parameters. However, we *can* subject external traffic sources transmitting to the same node or port to them. We therefore configure DSCP priorities on the source ports of traffic. We choose two configurations: Equal priority and low/high priority. In the first we use equal (or no) DSCP values for the traffic, whereas in the second we use traffic class 1 (low) and 4 (high) to grant priority to alternating nodes. We omit drop probability as it is superseded by traffic class. All queues that we use are Weighted Round Robin (WRR) queues, as strict priority queues could cause lower priority traffic to never be transmitted. We apply priority queues to switch ports in an alternating fashion, such that our hosts transmitting interference traffic between them have different priorities. In our experiment topologies, which we describe in Sections 3.5, 3.6 and 3.7, the interference traffic we show between nodes has one node on higher priority than the other if we use unequal priority. We realize priority queues on-host in a similar way using OVS-DPDK, giving flows different DSCP traffic classes – as we mentioned previously – in an alternating fashion.

Traffic shaping is configured using a single bucket, with a CBS of 100k and a CIR of 1000M. This way we utilize most of the available bandwidth, reducing the impact the bandwidth reduction itself has on performance – though we do still throttle it to a small degree. This way of shaping allows us to send traffic via micro bursts, consuming the CBS, which should cause our applications that send traffic infrequently to be hardly impacted by the token bucket. The bursts may cause small temporary bottlenecks, however, which we expect to impact the performance of the distributed applications. We apply the token buckets to the switch ports corresponding to the nodes and police only *outgoing* traffic from the node (i.e. *incoming* traffic from the perspective of the switch). This restriction with relation to policing direction also applies to the interfaces used by OVS-DPDK, which we use to apply a per-container token bucket. We use the same CBS and CIR as the token bucket on the regular switch for these token buckets.

Finally we use a hybrid traffic shaping approach, using both the physical switch as well as the OVS-DPDK backed on-host switch. In this approach we use either an on-host token bucket and an on-switch priority queue, or vice versa. We do not alter the configurations of either – they are as we described previously – which allows us to observe the effect of shared responsibility on the performance of the applications as well. We expect that this allows for more fine-grained control using on-host shaping, while we could use the on-switch shaping to reduce the load on the host to take over course-grain traffic shaping tasks.

If we were to use the dual-bucket setup we described in Section 2, we would be able to utilize the full link speed at all times without any baseline throttling. However, since traffic exceeding the CIR is marked using lower DSCP values to accomplish this – increasing the probability packets are dropped – this would cause an overlap between our token bucket and priority queue profiles. We therefore only use the single bucket traffic shaping.

3.4 Distributed Applications

The distributed applications we use in our experiments fall in the following categories: Key-Value Store, Big Data, and High Performance Computing (HPC), thereby creating a representative baseline covering different types of distributed applications typically found in cloud environments nowadays [5, 29, 46, 61]. Concretely, we picked MongoDB [62], Apache Spark [86] and applications using the Message Passing Interface (MPI) [35] respectively.

3.5 Key-Value Store Experiment Setup

We use a synchronizing MongoDB cluster with one primary and two backup nodes, which replicate data from the primary. The MongoDB cluster is reachable from the node we run the benchmark on through a user-managed switch, as we show in Figure **??** of the Appendix. On-switch traffic shaping is implemented here. Between node pairs we generate bidirectional, normally distributed interference traffic, described in Section 3.2. An OVS-DPDK backed Docker overlay spans the network through the user-managed switch, to connect MongoDB, our benchmark, and our Iperf3 noise generation. Our nodes communicate out-of-band to transmit Docker network overlay events, such as containers joining or leaving.

To evaluate the performance of the MongoDB Key-Value Store cluster, we use the MongoDB YCSB benchmark [14]. This contains six predefined experiments [15], which we show in Table 3. They cover a range of use cases using varying relative amounts of read, update and write operations.

3.6 Big Data Experiment Setup

To measure the effect of traffic shaping on a Big Data workload, we use Apache Spark [86] and the HiBench [45] "sparkbench" benchmark. This benchmark comprises real-world workloads from web search, machine learning and analytical query domains, as well as "micro" benchmarks based on the example applications provided with Apache Spark. Our experiment topology consists of four Spark workers with interference traffic between themselves – which we generate as described in Section 3.2 – as well as a separate master node which also houses the `namenode`, as we show in Figure **??**.

In the preliminary experiments we conducted, the Terasort benchmark showed promise due to its reconfigurability with regard to data size, as well as due to its degree of use of communication channels. This makes it more susceptible to changes in bandwidth and latency, which should allow any effect of traffic shaping to be visible through application performance.

3.7 HPC Experiment Setup

Our setup for the High Performance Computing experiment consists of a cluster of MPI nodes with bidirectional interference traffic between them – as we described in Section 3.2 – which we connect through a user-managed switch, as we show in Figure **??**. We run the Docker containers on the MPI nodes using privileged mode, as well as in the host IPC namespace, to ensure that the MPI send & recv calls run without error.

We tested the performance of the cluster of MPI nodes using the HPC Challenge (HPCC) benchmark suite [56], which consists of a set of standardized benchmarks as well as latency and bandwidth related benchmarks [54], which we show in Table 4. These benchmarks cover a range of workloads such that we can evaluate multiple facets of the performance of the system. We run the benchmark suite for 100 times for all the traffic shaping settings we described in Section 3.3, using the "base run" settings. This prohibits us to modify the HPCC source code, which we found was not needed in order to run HPCC on our Docker-based MPI cluster. We configured each node through a `hostfile` to have eight slots, with a maximum of ten. We imposed no memory constraints.

4 RESULTS

After executing the experiments described in Section 3 we ended up with a large collection of over 1000 individual experiments. We discuss a selection of the most relevant and interesting results.

4.1 Data Interpretation

As best practice suggests, we ran experiments sufficiently many times to achieve statistical significance (more details in Section 4.5). After ensuring that our results are stationary and not multi-modal, we decided to present our data in as much detail as possible rather than only a single value such as either median or mean.

Therefore, in most of the graphs presented in this section we focus on the full distribution of data through presenting boxplots. We have decided to use these because they are better at presenting the spread (and hence variance) of data. For experiments where performance exhibits a rather long-tailed distribution, we present additional data for the 95th, 99th, and 99.9th percentiles, which are widely used to describe the *tail* performance of systems.

When comparing boxplots, several aspects are important. First, boxplots where whiskers are more far apart show a distribution that exhibits more variability and longer tails. Therefore, practitioners should always prefer results depicted in boxplots that have tighter ranges as the performance they characterize is more predictable. The same goes for the box part of the boxplots, which characterize

Workload	Description	Example
A	Update heavy workload	Recent actions being recorded in a session store
B	Read mostly workload	Tagging Photos. Tags can be added (update), but reading tags is the most occurring operation
C	Read only	Caches of user profiles which are created elsewhere
D	Read latest workload	News, where people want to read the latest, e.g. user status
E	Short ranges	Forum threads, where each scan retrieves all comments for the thread, using e.g. a thread id
F	Read-modify-write workload	A database allowing for the recording of user data/activity i.e. reading, writing and modifying data.

Table 3: The workloads contained in the YCSB Benchmark Suite, which cover a range of use cases [15]. We run each benchmark using a record and operations count of $1,000,000$**, using a modified version of YCSB which outputs more detailed results.**

Benchmark	Benchmark Focus
HPL [69]	Floating point execution rate for solving a system of linear equations.
DGEMM [21]	Floating point execution rate for double precision real matrix-matrix multiplication.
STREAM [59]	Sustainable memory bandwidth (in GB/s).
PTRANS [4]	Rate of transfer for large arrays of data from multiprocessor's memory.
RandomAccess [48]	Rate of random updates of memory.
FFT [81]	Floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform (DFT).
Latency/Bandwidth (Based on `b_eff` [76])	Latency and bandwidth of network communication using basic MPI routines. The measurement is done during (non-)simultaneous communication and therefore covers two extreme levels of contention that might occur in real applications: no contention and contention caused by each process communicating with a randomly chosen neighbor in parallel.

Table 4: The measurement focus of workloads [55] contained within the HPCC benchmark suite [56]. The suite covers different aspects of HPC performance such that we can obtain a representative measure of system performance. We run each benchmark for 100 passes in the "base run" config, where no altered source code is allowed.

the 25th, 50th and 75th percentiles. Moreover, when directly comparing two distributions practitioners usually focus on a certain percentile. Therefore, for example, one could pick the median (e.g., 50th percentile) for drawing conclusions about certain configurations of the traffic shaping setup. When drawing conclusions about specific traffic shaping techniques we either consider the median performance or tail percentiles (e.g., P95, P99, P99.9).

Finally, in all our experiments, when presenting performance we either focus on throughput (e.g., operations per second, bandwidth etc.), or latency. For the former, higher is better, while for the latter smaller is preferred in practice.

4.2 Key-Value Store

Key-value stores are an integral part of modern cloud workloads [29]. They exhibit a unique network *fingerprint* in that they send and receive many relatively small payloads coming from many (distributed) users.

Hypothesis: Key-value store workloads are generally more sensitive to latency rather than bandwidth.

The results we obtained from the Key-Value Store experiments fell into roughly three categories. The first category encompasses the general improvement of the latency consistency when we apply traffic shaping techniques, with the exception of on-switch token buckets which slightly reduce consistency and have high tail latency. The second category concerns the fact that we observed that traffic shaping techniques have little effect on the latency consistency, with the exception of on-switch token buckets which make it slightly worse. The third category is similar, where we generally observe little influence from the traffic shaping techniques, though in this case we see that the on-switch token bucket has a large negative impact on the latency consistency. In this Section, for each of the categories we highlight some of the results.

4.2.1 General Latency Improvement. We show an example of the first category – a general improvement over the baseline with interference traffic – in Figure 1. All traffic shaping techniques reduce the spread of operation latencies we measured, with the exception of the on-switch token bucket. We observe this in the results of `READ` operations across workloads `B`, `C` and `D`, which are `READ`-heavy workloads. We measured a median latency below 200µs, whereas the median latency of the `READ` or `SCAN` operations we measured in other workloads is much higher, even when we compare the "no interference" baseline latencies. We presume this is due to the fact that their `READ` operations are larger, which reduces the relative effect the traffic-shaping techniques might have.

While the spread of the latencies depicts the on-switch token bucket as equivalent or slightly worse compared to the normally distributed interference without traffic shaping, the tail latencies show a very large difference. As we show in Table 5, these tail latencies are two to three orders of magnitude larger. We presume this is due to the fact that in this case we share the token bucket between tenants (i.e. the benchmark and the interference), causing heavy contention. We also observe a negative impact on tail latency caused by on-host token buckets, achieving a P99.9 twice as high as the experiment without any traffic shaping present. We can slightly improve this by adding a priority queue, which across all experiments improves tail latencies. For this `READ` operation workload in particular, we observe that the on-host priority queue outperforms the other options in both consistency and tail latency.

Conclusion 1. In Key-Value Store workloads, smaller read operations benefit from priority queues in both latency consistency and tail latency.

Conclusion 2. In Key-Value Store workloads, the latency consistency of smaller read operations may benefit from *on-host* token buckets, though at the cost of a higher tail latency. *On-switch* token buckets provide worse consistency and have tail latencies two to three orders of magnitude higher.

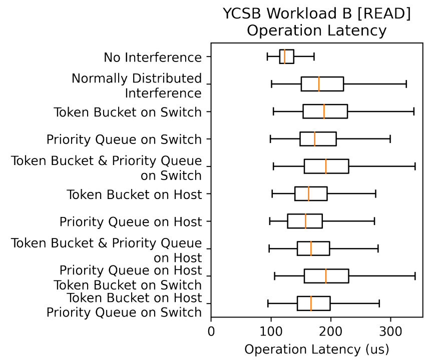

Figure 1: Read latency YCSB workload B (whiskers: ±1.5IQR). Traffic shaping measures generally improve latency, on-switch token buckets have little effect.

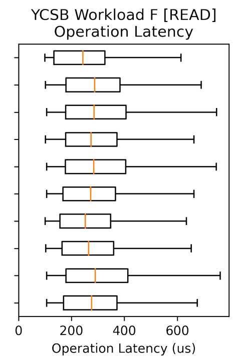

Figure 2: Read latency YCSB Workload F (whiskers: ±1.5IQR). Traffic shaping measures generally have little effect on latency, on-switch token buckets make latency slightly less consistent.

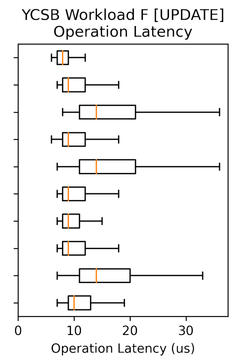

Figure 3: Update latency YCSB Workload F (whiskers: ±1.5IQR). Traffic shaping measures generally have little effect on latency, except the on-host priority queue. On-switch token buckets greatly reduce latency consistency.

Experiment Runtime (µs)	P95	P99	P99.9
No Interference	246	360	551
Normally Distributed Interference	440	633	937
Token Bucket on Switch	430	**205695**	**211199**
Priority Queue on Switch	389	536	759
Token Bucket & Priority Queue on Switch	429	**205823**	**211071**
Token Bucket on Host	394	856	1947
Priority Queue on Host	318	449	681
Token Bucket & Priority Queue on Host	391	653	1708
Priority Queue on Host, Token Bucket on Switch	428	**205695**	**210687**
Token Bucket on Host, Priority Queue on Switch	392	626	1610

Table 5: The tail latencies of the Read operation in YCSB Workload B with shaping present we observed are generally slightly above or below that of interference without shaping, though the on-switch token bucket incurs a tail latency far larger than any other traffic shaping measure.

4.2.2 No Latency Improvement & Large Tail Latency. We show an example of the second category, in which traffic shaping has little effect on latency consistency in general, in Figure 2. Here we once again observe the on-switch token bucket slightly reducing consistency compared to the baseline interference without any traffic shaping present. We also observe this pattern in the READ operation of workloads A and F, the READ-MODIFY-WRITE operation of workload F, and the SCAN and INSERT operations of workload E. These workloads generally take more time than those of the first category we showed in Section 4.2.1, and they are more focused on updating, inserting or modifying records instead of primarily reading them. This shift in focus might also be the reason that we observe a higher baseline "no interference" latency in these workloads, which leaves little room for improvement due to traffic shaping.

Similarly to the READ-focused workloads, the tail latencies we observed for traffic shaping measures using on-switch token buckets are two to three orders of magnitude larger than the other solutions, as we detail in Table **??** (see Appendix). Contrary to these READ-focused workloads, however, we observe that any other traffic shaping measure is equivalent or improves tail latencies compared to the baseline with interference traffic. This is further improved by the addition of a priority queue, except in the case where we implement the token bucket on-host and the priority queue on-switch.

We found that the INSERT operation of workload E is a special case since its on-switch token bucket tail latencies differ from the others, as they are more in line with that of the other traffic shaping techniques. In the case of the on-host token bucket, we found that it even achieves the lowest tail latency of all traffic shaping settings, though the on-host priority queue still provides the most consistent latency. We expect that this may be because workload E uses (short) ranges in its operations. Assuming that inserting a range takes longer than a single insert or update, the latency consistency may be relatively less impacted between traffic shaping settings. We found that the on-host token bucket is better due to the same reason, as the larger size of ranges benefits from per-tenant throttling as this results in a fairer division of bandwidth.

Conclusion 3. In Key-Value Store workloads, read operations in update-heavy workloads benefit from traffic shaping – especially priority queues – mostly by reducing tail latency, though general consistency is hardly impacted. This does not hold for on-switch token buckets which increase tail latencies by two to three orders of magnitude.

Conclusion 4. In Key-Value Store workloads, insert operations inserting (small) ranges – i.e. consecutive records of a sorted attribute – benefit from on-host token buckets to reduce tail latency and exhibit a small impact on consistency and tail latency when using on-switch token buckets.

4.2.3 Large Decrease in Latency Consistency. We generally observed little impact on latency consistency by introducing traffic shaping techniques in the third category, with the exception of the on-switch token bucket. When we added this, it negatively impacted the consistency, as we show in Figure 3. We also observed this kind of behavior in the `UPDATE` operations of workloads `A`, `B` and `F`, as well as the `INSERT` operation of workload `D`. As we found that this `UPDATE`-heavy workload generally has a very low operation latency, and that the negative impact of the on-switch token bucket is relatively large compared to the previous two categories. Still we expect that small improvements may be achieved through the use of the on-host priority queue.

Compared to the other two categories, we observe that tail latencies hardly increase through the use of traffic shaping techniques. Similarly, we observe that the on-switch token buckets generally show an increase in tail latency over the interference without traffic shaping, though this increase is comparatively small, as we detail in Table **??** (see Appendix). Given the structure of the experiment, we think it is likely that the `UPDATE` operations hardly gets impacted by the on-switch token bucket, since this meters incoming traffic from the perspective of the switch, i.e. traffic sent by the nodes. While our interference traffic is bidirectional, the sending part of the benchmark has no contention with it due to this reason. We presume that only the confirmation response from the MongoDB node has a possibility of getting delayed due to contention, which would explain why we observe no large tail latencies for the on-switch token bucket configurations. We could then explain the overall increase in latency – both in the tail and consistency – as a flat overhead seemingly introduced by the on-switch token bucket.

4.3 Big Data

Widely-spread in current clouds, through offerings such as Databricks, Snowflake or AWS EMR, big data workloads are an integral part of the cloud workload landscape nowadays. While most workloads are CPU-intensive [66], several are sensitive to traffic shaping [82], especially due to changes in bandwidth. This is because these workloads usually send large payloads for either fully reading remote files or exchanging intermediate data through shuffles.

Hypothesis: The network-bound big data workloads make large data transfers and are therefore sensitive to bandwidth rather than latency.

Not all benchmarks are equally dependent on the underlying network [66], we found that it would be very difficult to show the effect of traffic shaping on an application hardly using the network over the noise of background interference traffic. We found that Terasort, by comparison, is network-bound. We thus present results achieved when running Terasort using a large dataset.

We found that the results of the HiBench Terasort benchmark, as we show in Figure 4, show that an increase in throughput directly translates to a decrease in runtime. The on-switch token bucket has a negative impact on performance, though it retains similar performance consistency to the results achieved on interference traffic without traffic shaping. This behavior is expected, as all communication between nodes (i.e. the benchmark traffic as well as the interference traffic) gets throttled by the token bucket, increasing contention. When we add a priority queue to the token bucket, however, we observe a decrease in performance consistency, especially so when it concerns an on-switch priority queue. This decrease in consistency is greater than we would expect when compared to the slight decrease in consistency resulting from adding only the (on-switch) priority queue.

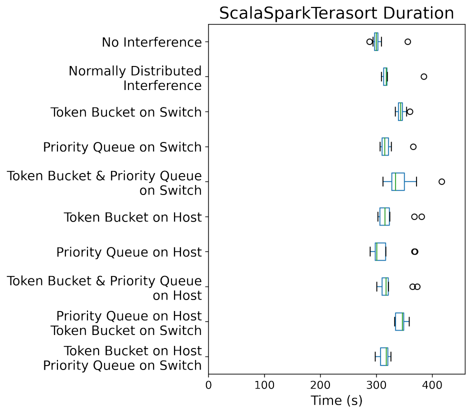

Figure 4: Execution time of Terasort under the effects of different traffic shaping settings (whiskers: ±1.5IQR). We find that only the combination of on-switch priority queue and token bucket shows a decrease in consistency.

The cause is the difference in priority between nodes causes workloads to sometimes send to nodes at a higher priority than the interference traffic, leading to an increase in observed performance. However, the opposite may also occur, where the interference traffic has higher priority. We find that these differences are then exacerbated by adding the token bucket, which (in the presence of lower interference) could lead to higher tails in performance, as well as lower tails when higher interference is present.

Conclusion 5. In Big Data workloads, the throughput related performance loss caused by the introduction of the on-switch token bucket may increase variability caused by the on-switch priority queue. This happens in cases where any node may connect to any other node, with not all nodes having equal traffic shaping settings.

4.4 High Performance Computing

HPC primitives are widely used nowadays in the cloud for many applications, such as high-frequency trading [53] or, weather or seismic simulation [67], or more recently, deep-learning [3, 11, 77]. These applications send and receive both large and small payloads, but, more importantly use networks as a means for synchronizing computation.

Hypothesis: HPC workloads are sensitive to both latency and bandwidth as they send both small and large payloads. Moreover, synchronization primitives (e.g., barriers) are especially sensitive to latency increases such as those given by packet buffering.

As we previously mentioned in Section 3.7, we used the HPCC benchmark suite to benchmark MPI performance in our High Performance Computing experiment. This suite consists of many subbenchmarks, each of which has their corresponding results. We identified roughly four categories, which are as follows. In this Section, for each category we highlight and discuss some results:

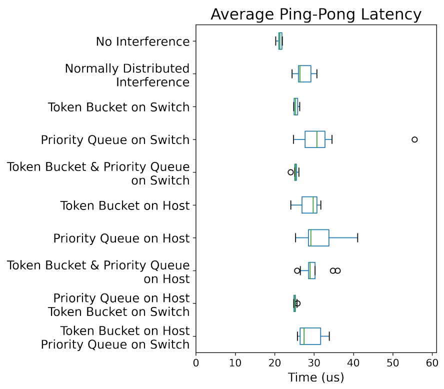

Figure 5: The on-switch token bucket improves performance consistency over no-shaping interference. Priority queues make it worse. When we combine the two, consistency increases further, while such improvements are not observed for on-host token buckets. Whiskers are ±1.5IQR.

- Performance of Token Bucket and Priority Queue combinations
- Negative Impact of the On-Switch Token Bucket
- Token Bucket Improvement
- Minimal Effect

4.4.1 Performance of Token Bucket and Priority Queue combinations. We found that in multiple HPCC sub-benchmarks the combination of a token bucket and priority queue results in better performance when compared to the baseline, where only the normally distributed interference without any traffic shaping is present. We primarily found that this increase presented itself in either increased consistency, decreased consistency with higher performance tails, or equivalent consistency at a higher performance level. We did not see this improvement across all combinations of token bucket and priority queue locations – i.e. both on-host, both on-switch or split up either way – however. In some cases, we found that the performance of the token bucket and priority queue combination even gets worse. In this Section we highlight some of these combinations.

The first combination concerns the on-host priority queue and the on-switch token bucket. As we show in Figure 5, both the average and maximum ping-pong latencies are very consistent when we use the on-switch token bucket, which is further improved upon when we add a priority queue. When we use the same configuration with an on-host token bucket we observe worse performance, however. We see similar behavior when measuring the average bandwidth, though we find that here the influence from the token bucket is far less. Do note that we measured this using dedicated latency and bandwidth-measuring benchmarks, which do not measure any other aspects of the system.

We were surprised by the latency results in particular, since they directly contradict the results we obtained in YCSB on MongoDB in Section 4.2. We expect this may be caused by the fact that the ping-pong latency is measured between all nodes, while the YCSB benchmark communicates with a single MongoDB entrypoint. Because this spreads the load, we presume that the impact from contention at the token bucket might be smaller. Another point of difference we found is that YCSB measures the entire operation latency, not merely the network latency.

Since this HPCC ping-pong benchmark has negligible operation cost on the nodes, as well as minimal required IP-packet size – the packet size is 8 byte – we find that the chance for contention to occur due to the token bucket is simply lower. Iperf3, by comparison, has a packet size of 8KB for TCP and 1470 byte for UDP [37]. With standard Ethernet frames having a maximum size of 1500 byte by default [23] (we do not use 9000 byte jumbo frames), the TCP packets we send using Iperf3 have to be split up into multiple large frames. All our Iperf3 traffic is therefore far larger than the traffic we generate in the ping-pong experiment. This might cause these small frames to be able to pass the token bucket whereas the Iperf3 frames would be dropped, when the token bucket nears depletion.

Due to the fact that the Ethernet links have a maximum bandwidth by default, we would expect to also observe this behavior when the token bucket is absent, which we do not. As we show in the bandwidth benchmark results in Figure 6 there is only a small difference in consistency between the presence and absence of the on-host token bucket. We expect that the Iperf3 traffic is allowed to fully saturate the Ethernet link, leaving less chance for smaller ping-pong packets to pass, increasing their latency due to head-of-line blocking.

Conclusion 6. In HPC workloads, very small IP-packets or Ethernet frames may benefit from on-switch (i.e. shared) token buckets when competing traffic consists of large Ethernet frames.

The second combination concerns the increase in performance at the cost of consistency when we use both the on-switch priority queue and token bucket. As we show in Figure 7, the throughput increases by adding the on-switch token bucket. We find that it manages to approach the performance of the benchmark without any interference present when the on-switch priority queue is added, which allows higher throughput peaks though at a far lower consistency. This behavior is not repeated for any other HPCC benchmark, however.

We find it plausible that the StarSTREAM "Add" benchmark benefits from the same perks as the latency benchmark previously did, in that the packet sizes are small enough to be allowed past the token bucket. The overhead added by on-host processing – which we did not observe in the latency benchmark – leads us to believe that this HPCC benchmark is more susceptible to other forms of resource contention (e.g. CPU). We also find that the decrease in consistency by adding the on-switch token bucket may be explained by the difference in priority of the node, as it varies between nodes. We would expect traffic from higher priority nodes to be less likely to be dropped than that from lower priority nodes, and given the fact that we use a star topology, we do meet multiple priority combinations. In our experiment, the MPI node on which we started the HPCC benchmark had a higher priority, which could explain the higher peaks.

A third combination is one where we observe *worse* performance when using the on-switch token bucket and on-switch priority queue, contrary to the performance we see in the StarSTREAM "Add" benchmark. As we show in Figure 8, this is likely due to the on-switch token bucket. Because the HPCC "Single" benchmarks runs on a (randomly selected) single node, we would expect hardly any network influence to be possible. We presume that the measured

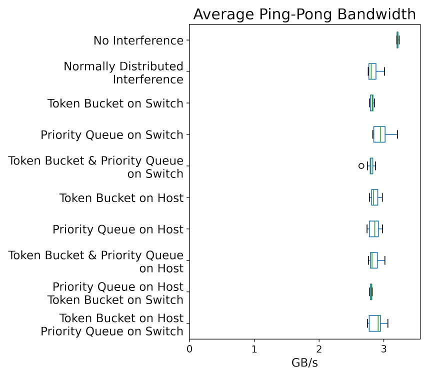

Figure 6: The on-switch token bucket improves performance consistency over no-shaping interference. On-switch priority queue degrades performance consistency. Combined on-switch token bucket with an on-host priority queue improves consistency. Whiskers are ±1.5IQR.

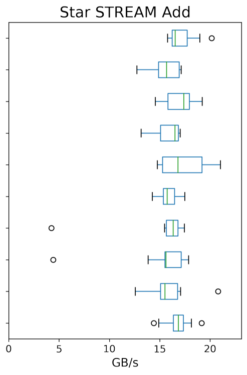

Figure 7: The on-switch token bucket with the addition of an on-host priority queue performs better for the StarSTREAM "Add" benchmark only. Whiskers are ±1.5IQR.

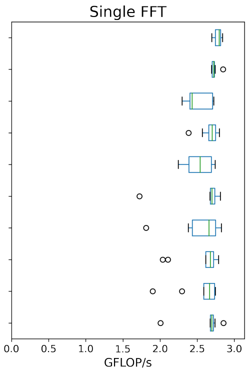

Figure 8: The on-switch token bucket, both by itself and combined with the on-switch priority queue, performs worse on "Single" FFT and RandomAccess. Whiskers are ±1.5IQR.

latency inconsistency is due to the transferring of data from the main HPCC node to the selected benchmark node at the start and end of the experiment, though this should not have this large an influence. We observe this behavior in both the Single FFT and RandomAccess benchmarks. While we find these results surprising – since the "Single" benchmarks are inherently non-distributed – we omit further analysis of these results.

4.4.2 Negative Impact of the On-Switch Token Bucket. Some of the HPCC experiments showed severe impact when we added the on-switch token bucket, to the point where in some cases there are orders-of-magnitude differences compared to other results. In Figures 9 and 10 we show two such examples. In the left case, we find that the difference is relatively small, allowing the results of the other experiments to still be visible. The MPI FFT benchmark we show on the right seems very sensitive to the on-switch token bucket in particular, showing far worse performance.

We find this large difference is surprising since the addition of the on-switch token bucket showed an improvement of performance with regard to latency, as we previously showed in Figure 5. We also showed in Figure 6 that the consistency of the bandwidth is improved by the addition of the on-switch token bucket. It is possible that the ping-pong bandwidth we showed there suffers less than the "Naturally Ordered Ring Bandwidth" benchmark of Figures 9 and 10, and that the MPI FFT benchmark is particularly sensitive to reductions in available bandwidth. Considering the previous results we find it exceedingly unlikely that high tail latency would be the cause in this case.

Conclusion 7. In HPC workloads, bandwidth-dependent applications have their performance reduced due to bandwidth contention when an on-switch token bucket is implemented. If bandwidth management is required, usage of an on-host token bucket for more granular control is recommended instead.

4.4.3 Token Bucket Improvement. We find that some of the HPCC benchmarks perform considerably more consistent once a token bucket gets added. The HPCC StarSTREAM benchmark in particular shows improvement on "Copy" and "Scale" when we introduce an on-switch token bucket. Unlike our previous results where the on-switch token bucket caused either a gain or loss of performance, in the case of the HPCC HPL benchmark we find that both the on-switch and on-host token bucket have a positive impact, as we show in Figure 11. When we add a priority queue in this case, we mostly observe a reduction in consistency.

HPL solves a dense linear system [69] and claims to be scalable with respect to computation and communication volume [70]. This latter claim comes with the assumption that there are direct point-to-point links between processors, which have a communication time roughly linearly dependent on the number of items communicated. This is not the case in our experiment since we use a switched network, where no point-to-point connections between nodes exist. When we take this into account, we find that the token buckets increase the reliability (i.e. consistent latency and available bandwidth) of the network, while the priority queues likely interfere with the assumed linear nature of communication, or at least do nothing to improve it. We find that this makes sense, as the traffic from different nodes or containers in the experiment is not treated equally when using priority queues, so a decrease in consistency would be unexpected.

Conclusion 8. In HPC workloads, the applications with stricter assumptions regarding network bandwidth and/or latency are likely to benefit from traffic shaping, token buckets in particular.

4.5 Threats to Validity

The focal points when discussing the validity of research results are construct, internal, and external validity [28, 84]. We discuss these three validity concerns in the following four points.

1. Result Statistical Confidence. As best practice suggests [51, 58, 82], we have run our experiments sufficiently many times such that we achieved confidence in the presented results. This

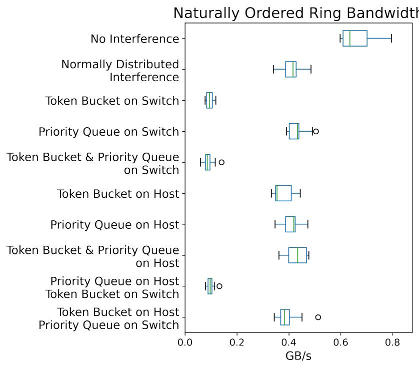

Figure 9: Negative impact due to the on-switch token bucket. This behavior is a stark contrast to the improvement in performance we showed in Figure 5. Whiskers are ±1.5IQR.

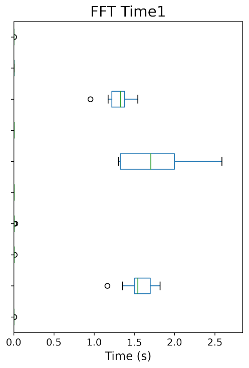

Figure 10: Large negative impact due to the on-switch token bucket. This behavior is a stark contrast to the improvement in performance we showed in Figure 5. Whiskers are ±1.5IQR.

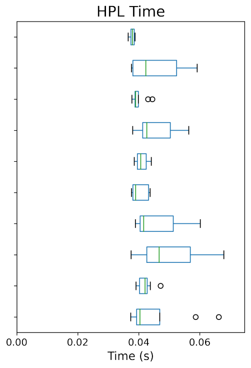

Figure 11: Token buckets consistently improve the performance achieved without any traffic shaping, unlike priority queues, which approximately match it. Whiskers are ±1.5IQR.

resulted in running experiments hundreds to thousands of times until confidence intervals for median performance became sufficiently tight [58]. This resulted in experiments being run many more times than the current status quo which favors few repetitions which do not offer confidence in results [82]. While this still leaves (negligible) potential room for error, we believe it is sufficient for the purpose of this article. Applying state-of-the-art statistical experimentation techniques gives us confidence that the results we achieved and the conclusions we have drawn are accurate.

Moreover, performing many trial and error microbenchmarks and experiments we ensured that the parameters we vary and consider for the traffic shaping techniques are the right ones for generating effects in the applications we considered. To make sure the list of parameters we consider and modify is exhaustive and there are no others that could be relevant we conducted exhaustive literature reviews of the topic [16, 22, 30, 41, 47, 74, 85]. We have described these in detail in Section 2.

2. Cloud Network Setups. Commercial clouds have different and constantly evolving datacenter networks [16, 30, 31]. These networks are likely to differ significantly between providers either in topology, technology or simply customer-facing policies. Hence, researchers or practitioners might find it difficult to quickly experiment with and understand the details behind cloud network implementations. This is either due to provider opaqueness or simply because of the inability to emulate such large-scale systems in lab-based setups. Nevertheless, although commercial cloud networks might seem out of reach, in this article we experiment with all their building blocks – from traffic shaping policies, to kernel-bypassing technology like DPDK. We therefore stress that our results constitute the very basis of understanding the performance of distributed applications under more complex cloud network incarnations.

3. Applicability to Other Application Domains. When designing the experiment presented in Section 3, one of the most important decisions was to thoroughly choose the application benchmark domains. The benchmarks we chose span wide and very important domains such as key-value stores, big data and high-performance computing.

While these domains are broad, the industry-standard benchmarks we used to evaluate their performance are both extensive and thorough, providing detailed results on many facets of their performance. Other domains which we do not cover – e.g. microservice-based architectures or distributed machine learning – can have overlapping network requirements to ours, such as low latency or high bandwidth, which may be subject to additional variance constraints. Moreover, we believe behavior exhibited by other applications, such as machine learning could be similar to applications we used in this work. For example, widely-used machine learning frameworks use MPI-based primitives for transferring data [8, 11].

While their corresponding performance impacts may not be completely equivalent, when we consider the spread of domains comprising our results we find that despite this, our findings could serve as a basis for estimating performance impact in these domains when the characteristics of their network usage are known.

4. Over-fitting to well-known Benchmarks. Overfitting systems to specific applications is a phenomenon many practitioners fall victim to. We argue that in this work the benchmarks we used are a means to an end – namely measuring effects when applying traffic shaping. Moreover, the overall method presented here is generic, and can be applied easily to other application domains as well by adding more benchmarks in the suite.

The benchmarks cover many facets of application performance, and we run them using mostly default settings. We have not done any benchmark specific tuning, and we have kept the benchmark settings identical between experiments. We do not benchmark a solution where tuning it to perform better on those specific benchmarks would be relevant or show better results. We merely aim to show differences in performance when traffic shaping is applied. The benchmarks are therefore a tool and not a measure of the quality of our work. Despite this, other benchmarks and/or other applications will likely exhibit some differences in performance changes, due to the nature of them having (slightly) differing network requirements. However, as the reliance on networks will show similarities, so will the results, which should follow the general trend shown in our results.

5 PRACTICAL IMPLICATIONS

When we take the results generated in this work into account, there are practical implications that one needs to consider when experimenting, benchmarking or deploying in shared cloud environments. We note that in all cases part of the distributed application performance is at the mercy of the cloud providers, since the traffic shaping policies for the shared network are not set in stone and may therefore be subject to change. Even after careful evaluation, it may be possible that after some time a different solution would provide better performance, everything else being equal.

1. Benchmark the to-be-deployed application. We found that the response of distributed applications to the addition of traffic shaping techniques varies based on both the application and traffic shaping techniques used, as the variety in our Conclusions shows. No one-size-fits-all solution is possible. We therefore advise to benchmark the application in the desired environment on a smaller scale, comparing the results to a (non-shared) controlled environment to quantify the impact. Additionally, if the provider of the shared cloud environment allows for some control over the intermittent traffic shaping, we encourage adding this as a tunable parameter in the benchmark. Finally, different cloud providers have differences in their traffic shaping policies. Should project constraints allow it, we find it worthwhile to compare between providers as well.

2. On-switch Token Buckets negatively impact tail latencies of many applications. We mentioned this in Conclusions 2 and 3. We find that this occurs primarily when implemented in a way where contention may occur, which sharply increases tail latency compared to a baseline without traffic shaping. We therefore advise the use of an on-host token bucket if control of bandwidth is required. This also allows for more granular, per-container control, and impacts tail latencies to a smaller degree, as we mentioned in Conclusion 7. While this does not hold for all types of applications, we found that the introduction of a priority queue may alleviate some of this impact, as we mentioned in Conclusion 1.

3. Applications with small IP packets may benefit from Token Buckets. If the distributed application sends small IP packets or Ethernet frames – compared to other traffic on the shared links – we found that the use of a token bucket may reduce tail latency when we compared it to a baseline without traffic shaping. In cases where tail latency is not reduced, we observed a relatively small increase. We observed this effect in both on-host and on-switch token buckets in Conclusions 2, 4 and 6. Note that this introduces a dependency on the network traffic of other tenants, which are generally outside our control.

4. Consider the assumptions about the network. In the general case, we find that applications with strict assumptions about the network benefit from traffic shaping techniques like priority queues and token buckets, which we mentioned in Conclusion 8. If we make assumptions about the topology of the network (e.g. star or mesh topology), we need to consider that this may influence the performance of the application when traffic shaping is added, as we mentioned in Conclusion 5.

5. Design experiments taking cloud variability into account. We found that both use of token buckets and priority queues – regardless of whether they are on-host or on-switch – may exacerbate the increase of variability caused by contention. While we may be willing to tolerate some variance when getting a rough estimate of viable cloud providers and traffic shaping settings, when we require repeatability and representative results, we need to take additional measures. While some of these recommendations have been made in the past [82], in the context of the acquired results we think they deserve repeating. In the case of repeatability, when we run experiments in shared cloud environments, we find that the presence of traffic shaping may require additional repeats of experiments to improve the reliability of the results. In the case of representative results, we find that specific experiments showing the change in performance in the presence of traffic shaping – such as those seen in shared cloud environments – may be necessary.We find that in both cases it may be beneficial to specifically benchmark the network latency and bandwidth over time – alongside other sources of contention – and provide this information in addition to the acquired results.

6 RELATED WORK

In this section we describe the relation of our contributions to the following categories of related work.

Effect of Network Conditions on Distributed Applications. Previous work has looked at the effect of sudden changes in WAN topology or traffic shaping on the performance of distributed applications [52, 57], as well as the impact of virtualization of network equipment, in particular when caused by CPU contention [78]. We further elaborate on the analysis of the effect the network has on distributed applications in this work by analyzing the impact of traffic shaping in particular.

(On-Host) Congestion Control in Data center Networks. Several articles have previously discussed ways to realize congestion control within data center networks, taking latency and bandwidth requirements, as well as impact on local resources (CPU, memory) into account [41, 47, 74, 85]. They achieve this either through on-host shaping techniques or VM placement. We further build upon this by determining the impact of such techniques on the distributed applications using the network, and we attempt to minimize this through e.g. hierarchical traffic shaping, which divides responsibilities between the on-host vSwitch and the traditional network switch.

Repeating Experiments in Networked Environments. In order to take countermeasures against variance in networked environments, previous research has looked at randomized multiple interleaved trials [1] and simultaneous benchmarking [9]. Next to this, it has identified inconsistencies in performance in public cloud environments despite or because of countermeasures being in place [82]. In addition to this, we discuss the resulting performance variance in distributed applications due to network contention caused by traffic shaping in particular.

Performance Guarantees. Performance guarantees have previously been measured using either dedicated tools to measure (tail) latency [49] combined with statistical tests, as well as combining simulation and emulation accurately representing a real-world topology to benchmark performance [39]. We further emphasise the importance of such guarantees in this work, given the impact that traffic shaping has on distributed applications.

Data Plane Development Kit. Previous research has expanded upon DPDK by processing traffic based on expected processing time [20]. It has also been compared to other state-of-the-art kernel-bypass network processors [73]. We utilize OVD-DPDK as a kernel-bypass network processor, omitting extensions to it and benchmarks of it.

7 CONCLUSION

Traffic shaping is a given in modern cloud datacenters. Cloud workloads are therefore inherently affected even though the experimenters who deploy them may not be aware. This leads to performance inconsistency, long-tailed performance and difficulty in determining what datacenter setups benefit applications.

In this article we set up to understand the performance implications of state-of-the-art traffic shaping mechanisms on a wide range of distributed applications. To this end we developed a comprehensive experimental harness that mimics real-world behavior while running real-world applications. Our comprehensive sets of experiments revealed novel insights into the non-trivial interaction between traffic shapers and applications. We offer a set of actionable insights, helping practitioners run benchmarks, deploying applications and making sense of performance.

ACKNOWLEDGEMENTS

The work in this article was in part supported by The Dutch National Science Foundation NWO Veni grant VI.202.195.

REFERENCES

[1] Ali Abedi and Tim Brecht. 2017. Conducting Repeatable Experiments in Highly Variable Cloud Computing Environments. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering* (L'Aquila, Italy) *(ICPE '17)*. Association for Computing Machinery, New York, NY, USA, 287–292. https://doi.org/10.1145/3030207.3030229

[2] Yaser Ahmed. 2018. https://www.intel.com/content/www/us/en/developer/articles/technical/using-docker-containers-with-open-vswitch-and-dpdk-on-ubuntu-1710.html.

[3] Ammar Ahmad Awan, Khaled Hamidouche, Akshay Venkatesh, and Dhabaleswar K Panda. 2016. Efficient large message broadcast using NCCL and CUDA-aware MPI for deep learning. In *Proceedings of the 23rd European MPI Users' Group Meeting*. 15–22.

[4] David Bailey, Mark Baker, Michael Berry, Jack Dongarra, Vladimir Getov, Ian Glendinning, Charles Grassl, Tom Haupt, Tony Hey, Roger Hockney, and et al. 1996. PARKBENCH MATRIX KERNEL BENCHMARKS. https://www.netlib.org/parkbench/html/matrix-kernels.html.

[5] Raj Bala, Bob Gill, Dennis Smith, David Wright, and Kevin Ji. 2021. Magic Quadrant for Cloud Infrastructure and Platform Services. https://www.gartner.com/doc/reprints?id=1-271OE4VR&amp;ct=210802&amp;st=sb.

[6] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. 2011. Towards Predictable Datacenter Networks. In *Proceedings of the ACM SIGCOMM 2011 Conference* (Toronto, Ontario, Canada) *(SIGCOMM '11)*. Association for Computing Machinery, New York, NY, USA, 242–253. https://doi.org/10.1145/2018436.2018465

[7] Hitesh Ballani, Keon Jang, Thomas Karagiannis, Changhoon Kim, Dinan Gunawardena, and Greg O'Shea. 2013. Chatty Tenants and the Cloud Network Sharing Problem. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX Association, Lombard, IL, 171–184. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/ballani

[8] Yixin Bao, Yanghua Peng, Yangrui Chen, and Chuan Wu. 2020. Preemptive all-reduce scheduling for expediting distributed DNN training. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 626–635.

[9] Lubomír Bulej, Vojtěch Horký, Petr Tuma, François Farquet, and Aleksandar Prokopec. 2020. Duet Benchmarking: Improving Measurement Accuracy in the Cloud. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (Edmonton AB, Canada) *(ICPE '20)*. Association for Computing Machinery, New York, NY, USA, 100–107. https://doi.org/10.1145/3358960.3379132

[10] Andrew Campbell, Geoff Coulson, and David Hutchison. 1994. A Quality of Service Architecture. *SIGCOMM Comput. Commun. Rev.* 24, 2 (apr 1994), 6–27. https://doi.org/10.1145/185595.185648

[11] Minsik Cho, Ulrich Finkler, David Kung, and Hillery Hunter. 2019. Blueconnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy. *Proceedings of Machine Learning and Systems* 1 (2019), 241–251.

[12] Cisco. 2010. Defining QoS Queues. https://www.cisco.com/assets/sol/sb/Switches_Emulators_v2_2_015/help/nk_configuring_quality_service16.html.

[13] CloudLab. 2021. The Cloudlab Manual - Hardware. https://docs.cloudlab.us/hardware.html.

[14] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing* (Indianapolis, Indiana, USA) *(SoCC '10)*. Association for Computing Machinery, New York, NY, USA, 143–154. https://doi.org/10.1145/1807128.1807152

[15] Brian Frank Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2019. YCSB Workloads. https://github.com/brianfrankcooper/YCSB/tree/master/workloads.

[16] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, et al. 2018. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization. In *15th USENIX symposium on networked systems design and implementation (NSDI 18)*. 373–387.

[17] Mike Dalton, David Schultz, Ahsan Arefin, Alex Docauer, Anshuman Gupta, Brian Matthew Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, Jake Adriaens, Jesse L Alpert, Jing Ai, Jon Olson, Kevin P. DeCabooter, Marc Asher de Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat. 2018. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018*. USENIX Association, Renton, WA, 373–387.

[18] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (2013), 74–80.

[19] Dell EMC Inc. 2017. User's Configuration Guide - Dell EMC Networking N-Series N1100-ON, N1500, N2000, N2100-ON, N3000, N3100-ON, and N4000 Switches.

[20] Henri Maxime Demoulin, Joshua Fried, Isaac Pedisich, Marios Kogias, Boon Thau Loo, Linh Thi Xuan Phan, and Irene Zhang. 2021. When Idling is Ideal: Optimizing Tail-Latency for Heavy-Tailed Datacenter Workloads with PerséPhone. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 621–637. https://doi.org/10.1145/3477132.3483571

[21] Jack Dongarra, Iain Duff, Jeremy Du Croz, and Sven Hammarling. 1989. LAPACK: Linear Algebra PACKage - DGEMM. https://www.netlib.org/lapack/explore-html/d1/d54/group__double__blas__level3_gaeda3cbd99c8fb834a60a6412878226e1.html.

[22] DPDK Project. 2021. Data Plane Development Kit. https://www.dpdk.org/.

[23] Mike Duckett, Jerome Moisand, Tom Anschutz, Diamantis Kourkouzelis, and Peter Arberg. 2006. Accommodating a Maximum Transit Unit/Maximum Receive Unit (MTU/MRU) Greater Than 1492 in the Point-to-Point Protocol over Ethernet (PPPoE). RFC 4638. https://doi.org/10.17487/RFC4638

[24] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu. [n. d.]. Iperf - the ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/.

[25] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. USENIX Association, Renton, WA, 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[26] Dmitry Duplyakin, Alexandru Uta, Aleksander Maricq, and Robert Ricci. 2020. In Datacenter Performance, The Only Constant Is Change. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 370–379.

[27] Jamie Ericson, Masoud Mohammadian, and Fabiana Santana. 2017. Analysis of Performance Variability in Public Cloud Computing. In *2017 IEEE International Conference on Information Reuse and Integration (IRI)* (San Diego, CA, USA). IEEE, San Diego, CA, USA, 308–314. https://doi.org/10.1109/IRI.2017.47

[28] Robert Feldt and Ana Magazinius. 2010. Validity threats in empirical software engineering research-an initial survey.. In *Seke*. 374–379.

[29] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *Acm sigplan notices* 47, 4 (2012), 37–48.

[30] Andrew D Ferguson, Steve Gribble, Chi-Yao Hong, Charles Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, et al. 2021. Orion: Google's {Software-Defined} Networking Control Plane. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 83–98.

[31] Dan Gibson, Hema Hariharan, Eric Lance, Moray McLaren, Behnam Montazeri, Arjun Singh, Stephen Wang, Hassan MG Wassel, Zhehua Wu, Sunghwan Yoo, et al. 2022. Aquila: A unified, low-latency fabric for datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 1249–1266.

[32] Google Cloud. 2022. Configuring a VM with higher bandwidth. https://cloud.google.com/compute/docs/networking/configure-vm-with-high-bandwidth-configuration.

[33] Google Cloud Platform. 2022. Egress Bandwidth. https://cloud.google.com/compute/docs/network-bandwidth#vm-out.

[34] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication* (Barcelona, Spain) *(SIGCOMM '09)*. Association for Computing Machinery, New York, NY, USA, 51–62. https://doi.org/10.1145/1592568.1592576

[35] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. 1996. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Comput.* 22, 6 (1996), 789–828. https://doi.org/10.1016/0167-8191(96)00024-5

[36] Daniel B. Grossman. 2002. New Terminology and Clarifications for Diffserv. RFC 3260. https://doi.org/10.17487/RFC3260

[37] Vivien Gueant. [n. d.]. iPerf = iPerf3 and iPerf2 user documentation. https://iperf.fr/iperf-doc.php.

[38] Haryadi S Gunawi, Riza O Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, et al. 2018. Fail-slow at scale: Evidence of hardware performance faults in large production systems. *ACM Transactions on Storage (TOS)* 14, 3 (2018), 1–26.

[39] S. Guruprasad, R. Ricci, and J. Lepreau. 2005. Integrated network experimentation using simulation and emulation. In *First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*. IEEE, Trento, Italy, 204–212. https://doi.org/10.1109/TRIDNT.2005.21

[40] HashiCorp. 2022. Consul by HashiCorp. https://www.consul.io/.

[41] Keqiang He, Eric Rozner, Kanak Agarwal, Yu (Jason) Gu, Wes Felter, John Carter, and Aditya Akella. 2016. AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 244–257. https://doi.org/10.1145/2934872.2934903

[42] Dr. Juha Heinanen and Dr. Roch Guerin. 1999. A Single Rate Three Color Marker. RFC 2697. https://doi.org/10.17487/RFC2697

[43] Dr. Juha Heinanen and Dr. Roch Guerin. 1999. A Two Rate Three Color Marker. RFC 2698. https://doi.org/10.17487/RFC2698

[44] Mong-Fong Homg, Wei-Tsong Lee, Kuan-Rong Lee, and Yau-Hwang Kuo. 2001. An adaptive approach to weighted fair queue with QoS enhanced on IP network. In *Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology. TENCON 2001 (Cat. No.01CH37239)* (Singapore), Vol. 1. IEEE, Singapore, 181–186 vol.1. https://doi.org/10.1109/TENCON.2001.949576

[45] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2011. The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis. In *New Frontiers in Information and Software as Services*, Divyakant Agrawal, K. Selçuk Candan, and Wen-Syan Li (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 209–228.

[46] IBM Cloud Team. 2020. Top 7 most common uses of cloud computing. https://www.ibm.com/cloud/blog/top-7-most-common-uses-of-cloud-computing.

[47] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster. 2015. Silo: Predictable Message Latency in the Cloud. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 435–448. https://doi.org/10.1145/2785956.2787479

[48] David Koester and Bob Lucas. 2009. RandomAccess. https://icl.utk.edu/projectsfiles/hpcc/RandomAccess/.

[49] Marios Kogias, Stephen Mallon, and Edouard Bugnion. 2019. Lancet: A self-correcting Latency Measuring Tool. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 881–896. https://www.usenix.org/conference/atc19/presentation/kogias-lancet

[50] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2015. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* 103, 1 (2015), 14–76. https://doi.org/10.1109/JPROC.2014.2371999

[51] Jean-Yves Le Boudec. 2010. *Performance evaluation of computer and communication systems*. Vol. 2. Epfl Press Lausanne.

[52] Luca Liechti, Paulo Gouveia, João Neves, Peter Kropf, Miguel Matos, and Valerio Schiavoni. 2019. THUNDERSTORM: A Tool to Evaluate Dynamic Network Topologies on Distributed Systems. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, Lyon, France, 241–24109. https://doi.org/10.1109/SRDS47363.2019.00034

[53] John W Lockwood, Adwait Gupte, Nishit Mehta, Michaela Blott, Tom English, and Kees Vissers. 2012. A low-latency library in FPGA hardware for high-frequency trading (HFT). In *2012 IEEE 20th annual symposium on high-performance interconnects*. IEEE, 9–16.

[54] Piotr Luszczek and Jack J Dongarra. 2012. HPC Challenge Benchmark. https://icl.utk.edu/hpcc/index.html.

[55] Piotr Luszczek and Jack J. Dongarra. 2016. HPCC Benchmark Suite Measurements. https://icl.utk.edu/hpcc/faq/index.html#90.

[56] Piotr Luszczek, Jack J Dongarra, David Koester, Rolf Rabenseifner, Bob Lucas, Jeremy Kepner, John McCalpin, David Bailey, and Daisuke Takahashi. 2005. *Introduction to the HPC challenge benchmark suite.* Technical Report. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

[57] Massimiliano Marcon, Marcel Dischinger, Krishna P Gummadi, and Amin Vahdat. 2011. The local and global effects of traffic shaping in the internet. In *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*. IEEE, 1–10.

[58] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. 2018. Taming performance variability. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 409–425.

[59] John D. McCalpin. 1991-2007. *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. Technical Report. University of Virginia, Charlottesville, Virginia. http://www.cs.virginia.edu/stream/ A continually updated technical report. http://www.cs.virginia.edu/stream/.

[60] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (mar 2008), 69–74. https://doi.org/10.1145/1355734.1355746

[61] Microsoft. 2022. What is cloud computing? A beginner's guide: Microsoft azure. https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/#uses.

[62] MongoDB Inc. 2021. MongoDB Documentation. https://docs.mongodb.com/manual/introduction/.

[63] Thomas Monjalon. 2022. DPDK supported NICs. https://core.dpdk.org/supported/nics/.

[64] Sung-Whan Moon and K.G. Shin. 2001. Implementing traffic shaping and link scheduling on a high-performance server. In *Proceedings Seventh IEEE Real-Time Technology and Applications Symposium*. IEEE, Taipei, Taiwan, 216–225. https://doi.org/10.1109/RTTAS.2001.929888

[65] Open Networking Foundation. 2015. OpenFlow Switch Specification – Version 1.5.1 (Protocol version 0x06). https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[66] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making sense of performance in data analytics frameworks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 293–307.

[67] Dhabaleswar Kumar Panda, Hari Subramoni, Ching-Hsiang Chu, and Mohammadreza Bayatpour. 2021. The MVAPICH project: Transforming research into high-performance MPI library for HPC community. *Journal of Computational Science* 52 (2021), 101208.

[68] A.K. Parekh and R.G. Gallager. 1993. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking* 1, 3 (1993), 344–357. https://doi.org/10.1109/90.234856

[69] A Petitet, R C Whaley, J Dongarra, and A Cleary. 2018. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. https://www.netlib.org/benchmark/hpl/.

[70] A Petitet, R C Whaley, J Dongarra, and A Cleary. 2018. HPL Scalability Analysis. https://www.netlib.org/benchmark/hpl/scalability.html.

[71] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. 2015. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 117–130. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[72] Ben Pfaff, Justin Pettit, and Jean Tourrilhes. 2021. ovs-fields – protocol header fields in OpenFlow and Open vSwitch. http://www.openvswitch.org/support/dist-docs/ovs-fields.7.pdf.

[73] Nikolai Pitaev, Matthias Falkner, Aris Leivadeas, and Ioannis Lambadaris. 2018. Characterizing the Performance of Concurrent Virtualized Network Functions with OVS-DPDK, FD.IO VPP and SR-IOV. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering* (Berlin, Germany) *(ICPE '18)*. Association for Computing Machinery, New York, NY, USA, 285–292. https://doi.org/10.1145/3184407.3184437

[74] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. 2017. Carousel: Scalable Traffic Shaping at End Hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 404–417. https://doi.org/10.1145/3098822.3098852

[75] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. 2010. Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance. *Proc. VLDB Endow.* 3, 1–2 (sep 2010), 460–471. https://doi.org/10.14778/1920841.1920902

[76] Gerrit Schulz and Rolf Rabenseifner. 2016. Effective bandwidth (B_EFF) benchmark. https://fs.hlrs.de/projects/par/mpi//b_eff/.

[77] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. 2018. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems* 31 (2018).

[78] Ryan Shea, Feng Wang, Haiyang Wang, and Jiangchuan Liu. 2014. A deep investigation into network performance in virtual machine based cloud environments. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. IEEE, Toronto, ON, Canada, 1285–1293. https://doi.org/10.1109/INFOCOM.2014.6848061

[79] Gurucharan Shetty. 2015. OVN-Docker-overlay-driver. https://github.com/shettyg/ovn-docker/blob/master/ovn-docker-overlay-driver.

[80] Julie Solon and Marilyn Flax. 2021. Amazon EC2 instance network bandwidth. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-network-bandwidth.html.

[81] Daisuke Takahashi. 2020. FFTE: A Fast Fourier Transform Package. http://www.ffte.jp/.

[82] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. 2020. Is Big Data Performance Reproducible in Modern Cloud Networks?. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 513–527. https://www.usenix.org/conference/nsdi20/presentation/uta

[83] Walter Weiss, Dr. Juha Heinanen, Fred Baker, and John T. Wroclawski. 1999. Assured Forwarding PHB Group. RFC 2597. https://doi.org/10.17487/RFC2597

[84] Hyrum K Wright, Miryung Kim, and Dewayne E Perry. 2010. Validity concerns in software engineering research. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*. 411–414.

[85] Ye Yang, Haiyang Jiang, Yulei Wu, Yilong Lv, Xing Li, and Gaogang Xie. 2021. C2QoS: CPU-Cycle based Network QoS Strategy in vSwitch of Public Cloud. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, Bordeaux, France, 438–444.

[86] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* 59, 11 (oct 2016), 56–65. https://doi.org/10.1145/2934664