POLITECNICO DI TORINO
Repository ISTITUZIONALE

A “Big-Spine” Abstraction: Flow Prioritization With Spatial Diversity in The Data Center Network

Original
A “Big-Spine” Abstraction: Flow Prioritization With Spatial Diversity in The Data Center Network / Cornacchia, A., Bianco, A., Giaccone, P., Sviridov, G.. - ELETTRONICO. - (2024). (Data-Plane Programmability and Edge Network Acceleration in the AI era (NetAccel-AI 2024) Pisa (Italy) 22-24 July 2024) [10.1109/HPSR62440.2024.10635939].

Availability:
This version is available at: 11583/2989653 since: 2024-06-18T13:27:16Z

Publisher:
IEEE

Published
DOI:10.1109/HPSR62440.2024.10635939

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2024 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

25 November 2026

A "Big-Spine" Abstraction: Flow Prioritization With Spatial Diversity in The Data Center Network

Alessandro Cornacchia
Politecnico di Torino

Andrea Bianco
Politecnico di Torino

Paolo Giaccone
Politecnico di Torino

German Sviridov
Huawei Technologies

***Abstract*—Data center networks undergo the coexistence of latency-sensitive *mice* flows and bandwidth-intensive *elephant* flows. Jointly optimizing the performance of both traffic classes poses complex challenges. Existing flow schedulers either rely on detailed flow size information or require numerous physical priority queues (PQs) within network switches, thus facing practical challenges.**

In this work, we propose a novel flow scheduling algorithm, namely Multi-Path Multi-Level Feedback Queueing (MP-MLFQ), to overcome these limitations. MP-MLFQ leverages the spatial diversity and regularity of DCNs to realize a scheduler with numerous *logical* priority levels while occupying as low as 2 physical PQ at each switch port. We designed MP-MLFQ to run atop modern programmable networks, and highlighted how to implement it without modifications at the end-hosts' stacks. Our simulation results show that MP-MLFQ outperforms existing flow size-agnostic solutions in minimizing the flow completion time, when only two PQs are available.

I. Introduction

Modern data centers host highly heterogeneous workloads including cloud-native web applications, machine learning (ML/AI) workloads, online gaming, etc., among others. Consequently, data center networks (DCNs) handle a spectrum of traffic flows characterized by conflicting latency requirements. Small-sized *mice* flows are predominant [1], [2], and are mostly sensitive to network delays, as they typically arise from short-lived interactions (e.g., RPCs across microservices). In contrast, large-sized *elephant* flows, although less frequent, can account for more than 80% of the total network traffic [3], [4] while being able to tolerate longer completion times (e.g., data transfers). Additionally, due to the rapid growth of interactive AI services, *medium*-sized flows of long-lived question&answer streams to the AI systems, also demand latency optimization.

While multiple solutions have been proposed, jointly optimizing flow performance across different types of traffic in DCNs still remains a challenging problem. Existing flow size-aware strategies [5], [6], [7] can achieve near-optimal Flow Completion Time (FCT). However, they assume a priori knowledge about individual flow lengths. In practice, this information is hard to obtain in realistic scenarios, as it would require a coordinated interaction between the applications and the network to explicitly declare the flow size before any connection is opened. This process is tedious for developers that typically wish to treat the network as a commodity. Even when coordination is possible, like for distributed ML training libraries that already integrate primitives to coordinate with the network [8], the actual flow size may be unknown a priori (e.g., it may result from the ML training process itself).

On the contrary, flow size-agnostic approaches [9], [10] can effectively schedule mice and elephant flows dynamically during the flow lifetime, without requiring any prior knowledge of the flow sizes. These approaches track the number of transmitted data units of each flow, and progressively assign flows to lower and lower priority queues, based on the measured size. Nevertheless, their effectiveness largely depends on the number of supported network priority levels, because mice and medium flows initially compete with elephant flows in the same priorities. Since commodity switches are equipped with few priority queues (PQs), these approaches can fall short in practice: either they occupy all available switch queues and interfere with other tasks of traffic isolation [11], [12], thus being unpractical, or assign too coarse-grained flow priorities, thus sacrificing performance.

In this paper, we propose *Multi-Path Multi-Level Feedback Queueing* (MP-MLFQ), a flow scheduling algorithm enabled by programmable switches which lies in the sweet spot between practicality and performance. As a key intuition, MP-MLFQ leverages the spatial diversity of data center network topologies to share the physical PQs across switch ports, thus providing a wider range of *logical* priority levels for flow prioritization while requiring as little as 2 physical PQs at each switch port. MP-MLFQ can achieve performance close to state-of-the-art flow size-agnostic schedulers, but keeps the deployment cost much lower. To the best of our knowledge, no previous work has exploited this feature to improve the performance of flow scheduling in data center networks.

We present the following key contributions:

- we introduce MP-MLFQ, a novel flow scheduling algorithm that leverages the spatial diversity of data center networks to realize numerous logical priority queues, while requiring only few physical queues per switch port (Sec. III);
- we discuss the challenges and solutions related to the implementation of MP-MLFQ on top of programmable data planes (Sec. IV);
- we show via numerical simulations that MP-MLFQ can achieve performance similar to existing algorithms in terms of average and tail FCT, while requiring significantly less physical PQs per switch port (Sec. V). In addition, for the same number of PQs, MP-MLFQ is superior to state-of-the-art by a factor of $1.15\times$.

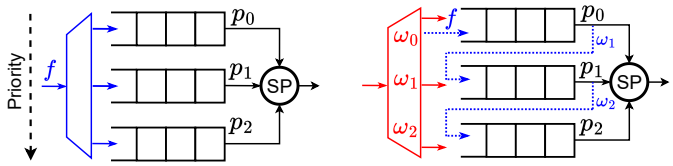


Fig. 1: Comparison between a SP (left) and a MLFQ scheduler (right) with per-flow datapath (blue) and per-packet datapath (red).

II. Background and Motivation

Flow scheduling, i.e., choosing the service order among a set of active flow so as to optimize a given objective — e.g., minimizing the average FCT or the number of missed deadlines or maximizing the average throughput — is imperative for data center network workloads characterized by huge diversity in terms of occupation of link capacity and latency requirements [1], [7]. Flow Completion Time (FCT) is typically considered the key metric, as it represents a direct measure of the performance of network communication and has a direct impact on user experience. In this section we discuss the necessary background related to priority-based size-agnostic flow scheduling in data centers, and motivate our approach to minimize FCT.

A. Flow scheduling in data center networks

If flow sizes are known in advance, Shortest Remaining Processing Time (SRPT) is known to be the optimal scheduling policy [13], [14] to minimize the average FCT. SRPT serves first the flows with the least amount of data units (e.g., packet or bytes), left to transmit. A discretized implementation of SRPT for data center networks has been proposed in pFabric [7], where flows are classified into a finite number of priorities based on their remaining size. Unfortunately, the length of individual flows is typically unknown or difficult to obtaine without undergoing into deep modifications of the application layer [15], which should interact with the network control plane for every new connection. Furthermore, for many applications the flow size is unknown even to the user, until the flow is terminated.

In the absence of knowledge about the length of individual flows, scheduling policies can still try to approximate flow size-aware schedulers. Least Attained Service (LAS) scheduler [16] gives priority to flows with the least amount of *transferred* data units. Differently from the *remaining* size, this information can be locally computed based on per-flow counters. Similarly to SRPT, LAS favors mice flows which are prioritized over large flows, and is known to be optimal when the hazard rate $h(x) = \frac{F'(x)}{1-F(x)}$ of the flow size c.d.f. $F(x)$ is decreasing, which holds for heavy-tailed traffic distributions [17], [18].

B. Multi-Level Feedback Queueing (MLFQ)

Multi-Level Feedback Queueing (MLFQ) is a discretized version of LAS, based on a finite number of priority levels. It is composed of N priority queues served with strict priority (SP) discipline. Let p_i the ith priority queue in decreasing order of priority, with $i = 0, \ldots, N-1$ (i.e., p_0 is the highest priority queue) However, differently from a normal SP scheduler, in MLFQ a flow f is demoted to lower priorities based on the amount of service $b_f(t)$ obtained up to time t. Specifically, $b_f(t)$ is compared against set of pre-defined *demotion thresholds* $\Omega = \{\omega_i\}_{i=0}^{N}$ to determine the priority of flow f, with $\omega_0 = 0$ and $\omega_N = +\infty$.

Fig. 1 compares a SP scheduler (left) and a MLFQ scheduler (right) with three priority queues, p_0, p_1 and p_2. The blue datapath depicts the logical behavior of a MLFQ scheduler (per-flow behavior), while the red datapath shows the actual implementation of a MLFQ scheduler operating on a per-packet level. From a flow perspective, each new flow is initially placed in the highest priority queue p_0 and once it has obtained enough service, it is demoted to the lower priority queue. Then the demotion process continues on the lowest priority queues until the flow completes. In its practical implementation, a MLFQ scheduler operates on a per-packet basis according to the following policy: a packet of flow f received at time t is enqueued in p_i, with i such that $\omega_i \leq b_f(t) < \omega_{i+1}$. Notably, at any given time, packets of the same flow may be stored in different queues. But for enough large size flows, after a while all the packets will be stored in the lowest priority queue p_N.

C. Limitations of MLFQ for commodity switches

The most crucial aspect of an MLFQ scheduler is the availability of a sufficient granularity of priority queues. This is because MLFQ is a discrete approximation of LAS, which implicitly assumes $N \to \infty$. Unfortunately, even when data center switches are equipped with enough physical priority queues, these might not be fully usable as they are typically reserved for other critical traffic isolation tasks. Such tasks include separation of different types of traffic [19] such as RDMA [12], DCTCP [3] and traditional TCP-based traffic, or isolation of other signaling flows. As an example, recent proposals such as FlexPass [11] leverage different PQs to isolate credit-based transports, such as Homa [10], from legacy host-based transports, for a smooth coexistence within the same infrastructure. Others [20] leverage physical queues to approximate advanced disciplines such as Push-In-First-Out (PIFO) queues, which also can occupy a large number of PQs.

MLFQ is ineffective with few PQs. Depending on the choice of the thresholds, the prioritization scheme may not be granular enough to separate mice from medium flows, or medium flows from elephant flows, thus leading to worse performance. Overall, due to limitations in the number of available PQs, MLFQ sacrifices performance or increases complexity, making it unpractical for realistic scenarios.

III. The "Big Spine" Abstraction

In the following, we introduce the design of Multi-Path Multi-Level Feedback Queueing (MP-MLFQ), a scheduling policy capable of emulating LAS scheduling while employing switches with *only two PQs*.

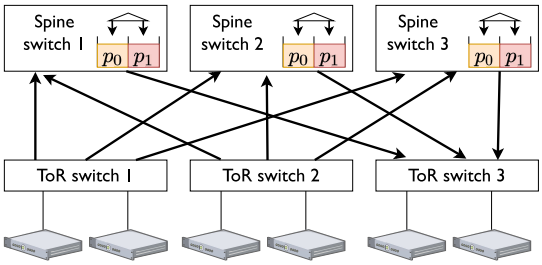


Fig. 2: Traditional MLFQ data center solution with just 2 priority queues (p_0 and p_1) for each spine switches. For simplicity only the traffic flows directed to ToR switch 3 are shown and only the priority queues on the spine switches directed to ToR 3 switch.

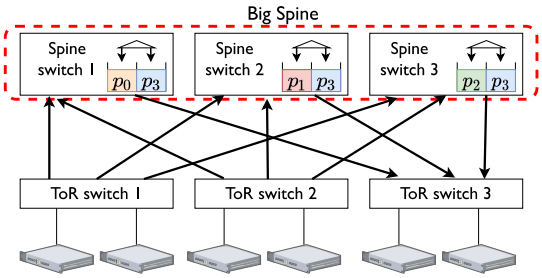


Fig. 3: MP-MLFQ-enabled data center solution with 4 priority queues (p_0, p_1, p_2, p_3) distributed across all the spine switches

A. System overview

We assume the availability of only $N = 2$ physical PQs per switch port, even if the proposed methodology can be extended to a larger number of PQs. To overcome the limitations of other state-of-the-art work that approximates LAS scheduling with discrete solutions such as MLFQ, our solution intelligently combines physical queues from multiple switches to achieve extra *logical* priority queues, i.e., more than two priority levels. The rationale behind MP-MLFQ is based on the following pivotal observations:

1) **DCN spatial diversity**: Data center topologies exhibit a high degree of *spatial diversity*, which we refer to the availability of multiple alternative forwarding paths at every hop in the DCN. This allows to provide high bandwidth and fault tolerance between any pair of servers. In the following, due to its broad adoption, we will consider a 2-layer leaf-spine DCN, in which Top-of-Rack (ToR) switches are connected to the servers and to the spine switches with full interconnection.
2) **DCN regularity**: DCN topologies are extremely regular, with alternative paths between any pair of nodes having the same number of hops and equal bandwidth. For this reason, simple decentralized load balancing techniques, such as Equal Cost Multiple Path (ECMP) or packet spraying [21], are typically adopted to achieve full-bandwidth utilization. Thus, in a leaf-spine DCN, the traffic between any pair of ToR switches is sent across all the spine switches.

MP-MLFQ leverages the above mentioned observations to provide a wider range of priority levels for flow demotion by exploiting the spatial dimension of the DCN.

Fig. 2 shows a traditional implementation of the MLFQ policy, according to which the spine switches deploy MLFQ on the network interfaces directed to the ToR switches. The traffic flows are spread from the ToR switches to the spine switches and the traffic is moved from p_0 to p_1 whenever enough traffic has been received. Notably, to avoid out-of-sequence problems, all the packets belonging to the same traffic flows are routed from the ToR switches to the same spine switch.

On the contrary, MP-MLFQ runs in both DCN layers. At the spine level, MP-MLFQ logically combines the queues of different spine switches to achieve a number of *logical priority levels* higher than the number of actual physical PQs at a single switch. As shown in Fig. 3, all the PQ pairs in the spine switches are seen as a pool of shared PQs, equivalent to considering the spine layer as a single *"Big-Spine"* switch sharing the PQs directed to a particular ToR switch to support MLFQ. In the specific example in the figure, the actual number of PQs is 8 instead of the original 2 present in the original solution. At leaf level, MP-MLFQ treats the ports within a ToR switch directed to spine switches as a single *one-big-port* abstraction. This allows to send the packets of the same traffic flow on a sequence of different spine switches, while preventing out-of-sequence events at the destination server.

Fig. 4 shows a toy example of a top-of-rack switch (ToR) with two ports `if-0` and `if-1`, connected to two spine switches. The packets of a long-size flow traverse all MP-MLFQ priority levels and are transmitted to different spine switches. In this example, the packet color denotes the priority level of a packet and, just as in MLFQ, is dynamically assigned based on the amount of transmitted data units (i.e., attained service) of the flow it belongs to. Every new flow starts by transmitting "yellow-tagged" packets. Flows are demoted to lower priorities once their total attained service exceeds ω_1 and ω_2, eventually transmitting "brown-" and "green-tagged" packets, respectively. Thus, initially "yellow-tagged" packets, are enqueued in the highest physical PQ of port `if-0`, corresponding to the logical PQ p_0. After being demoted to "brown-tagged" packets, instead of being moved to the next PQ in `if-0` as it happens in MLFQ, in MP-MLFQ flows are moved to a PQ in a *different* port. In the example in Fig. 4 it happens to be the first PQ of `if-1` which logically represents p_1. This step, which we refer to as *cross-path demotion* (denoted as ①) involves flow re-routing on a different link and can be repeated for K times, where K is the number of spine switches ($K = 2$ is in our example). Since the re-routed flows receive service at high (local) priority in the subsequent SP schedulers, ① emulates K new logical priorities. Notice that a traditional MLFQ scheduler would instead be limited to 2 priority levels only. Therefore, the granularity used by the brown range would be lost: flows in this range would either remain in the high priority queue for too long, interfering with short flows, or would be demoted to the low priority queue too early, contending with the heavy traffic load brought by elephant

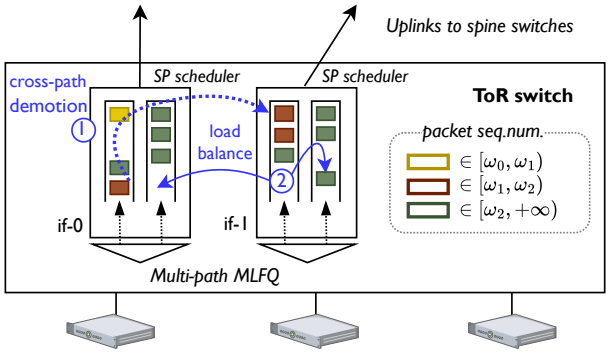


Fig. 4: MP-MLFQ operations on a Top-of-Rack (ToR) switch, in the simplest scenario of 2 connected spine switches and 2 physical PQs per interface. Ports `if-0` and `if-1` share their PQs to construct a global scheduler with logical PQs p_0, p_1, p_2. The blue arrows indicate per-flow datapath.

flows. MP-MLFQ instead achieves higher priority granularity with cross-path demotions, mitigating this problem.

After the cross-path demotion, both brown packets (with $b_f(t) \in [\omega_1, \omega_2]$) and green packets (with $b_f \in [\omega_2, \infty]$) are enqueued in the logical priority queue p_1 in port `if-1`. At this point, MP-MLFQ exploits the second PQ of each port. Since the elephant flows are the major contributors to network utilization, to best exploit the available capacity, the packets of elephant flows are spread across the K paths after the last priority demotion, referred to as *load balance* demotion (step ②). Namely, the longest flows that undergo the last demotion have their packets (green packets) enqueued in one of the K physical low PQs in the ToR switch, corresponding to the $(K+1)$-th logical priority, e.g., p_2 in our example.

B. Implications on load balancing

In MP-MLFQ all flows are initially routed through the same ToR port towards the same spine switch. Consequently, differently from standard load-balancing schemes like ECMP, the proportion of traffic spread across any given path directly depends on the value of the demotion thresholds. As a limiting case, consider the scenario where the first demotion threshold ω_1 is set equal to a single packet. In this case, all flows are early re-routed to port `if-1`, resulting in a load imbalance across the two links.

We avoid such situations and ensure the load is balanced across the available paths by splitting unevenly the traffic of the last priority level to compensate for load imbalance, during step ②. This can be implemented with readily-available weighted ECMP. Given a threshold assignment Ω and a workload distribution $F(x)$, the average traffic load on every spine can be computed analytically, from which we derive the ECMP weights.

IV. Implementation in programmable networks

MP-MLFQ can be implemented with the readily available components found in production data centers, and transparently to end-hosts applications.

A centralized control-plane can collect statistics about the generated flows, derive the corresponding workload distribution, and compute the demotion thresholds, which are sent to the switches to configure the MP-MLFQ schedulers. In parallel, we envision three alternative data-plane solutions to measure flow sizes and implement priority-based demotion.

- **End host's kernel**: eBPF programs at end-hosts provide a universal way of implementing MP-MLFQ without any specialized hardware, and transparently to the running VMs. eBPF *maps* can be effectively used to communicate flow statistics as well as receive threshold parameters from the threshold controller. eBPF-based vTAP interfaces or XDP+VirtIO-net can be used to hook the eBPF programs to inter VM traffic [22], [23]. Overlay routing or traditional priority-based mechanisms (e.g., IEEE 802.1p [24]) can then be used to force flows to select particular paths in the network based on their priorities.
- **SmartNIC**: eBPF program execution can also be moved to programmable data-path SmartNICs, such as Netronome Agilio CX 2x40GbE [25], which provide eBPF hardware offload support through the Netronome Flow Processor (NFP) driver. Thanks to NFP, users can effortlessly program these NICs without necessitating an in-depth understanding of the NIC hardware. Due to resource constraints aboard the NIC [26], one would need to resort to approximate data structures for packet counting, such as counting sketches [27], to obtain flow sizes.
- **Programmable switch**: in the absence of SmartNICs, programmable switches can be used to offload the same task (e.g., P4-based). Indeed, sketch-like data structures have been already implemented in commercial programmable switches [28] and permit to monitoring flow sizes at line rate. Adaptive routing can then be easily implemented within the switch's data path to provide priority based routing.

Quantifying the impact of early (or late) flow demotion on the FCT due to the approximation introduced by counting sketches is left outside the scope of this paper.

V. Performance evaluation

A. Experimental setup

We implemented a numerical simulator in Python (code available on GitHub [29]) that models a single ToR switch with multiple uplink ports. This simplified scenario models a traffic matrix in which all the traffic sources are located in the servers connected to the same ToR switch and are sending traffic to the servers connected to other ToR switches. Thanks to the full bijection bandwidth of the considered DCN, given the considered traffic matrix, the downlink ports from the spine switches to the ToR are not congested and thus the effect of modeling the queueing effects on the spines switches is negligible. The ToR switch is modeled with K independent SP flow schedulers, one for each uplink port (i.e., the number of spine switches is also K). Each scheduler manages N PQs. For better scalability of the simulation, we consider a flow-based fluid approximation of the system. Flows are modeled as

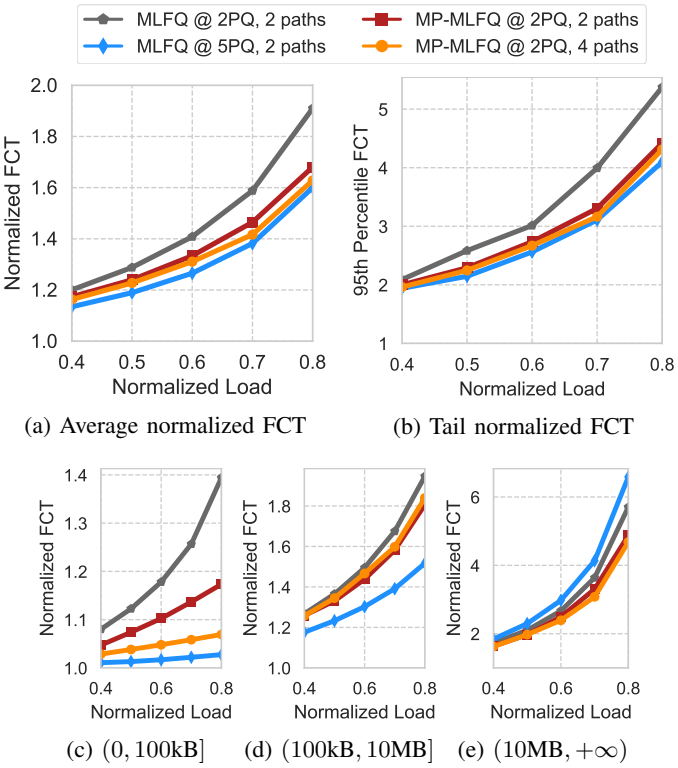
jobs of a given size, thus the simulator does not support packet-by-packet operations. All flows inside each PQ are served in parallel with a Processor-Sharing service discipline, i.e., the available service rate is equally subdivided among the flows in the same priority queue. This approximates a packet-by-packet sharing of each queue. The service is preempted when *(i)* a new flow arrives, at which point the priority queue to serve and the equal bandwidth share is re-computed, or when *(ii)* one of the flows reaches a demotion threshold.

The simulator supports different numbers of priority levels N and parallel uplink ports K. When K is set equal to 1, the simulator behaves as a traditional individual MLFQ scheduler. When $K > 1$ the simulator runs K parallel SP schedulers, whose capacity is set μ/K. Thus, the overall capacity of the data center is constant and equal to μ. Flows are generated according to a Poisson process with constant rate λ, and the simulator supports MLFQ and MP-MLFQ disciplines. For MLFQ, the flow arrival rate λ is equally split among the K schedulers and each serves flows arriving at rate λ/K. This emulates standard flow-level load-balancing techniques such as ECMP. For MP-MLFQ the flow arrival rate is λ, and all flows are initially sent to the output port corresponding to the first spine switch, hosting p_0, coherently with the discussion in Sec. III. Flows are served at rate μ/K until they are demoted to lower priorities, in which case they are sent to the subsequent $K-1$ schedulers.

Performance metrics. Given a network topology in the simulator with total capacity μ, we define $\text{FCT}_{\text{opt}}(x)$ as the FCT achieved by a flow of length x in a completely empty network, i.e., at zero load. We then measure $\text{FCT}(x)$ as the FCT of any flow of length x in the presence of other flows. Given a total of N flows we compare performance based on a normalized FCT metric defined as $\text{nFCT} = \text{FCT}(x)/\text{FCT}_{\text{opt}}(x)$. The nFCT represents the relative degradation of the flow completion time with respect to the ideal one, and allows a quick performance comparison under varying topology sizes (i.e., different link speeds μ/K) on the same graph.

Workloads. We employed a widely adopted flow length distribution [3], [4], [7], [9], derived from a WebSearch workload in a production data center. The workload has heavy-tailed flow sizes: while more than 80% flows are less than 1MB in size, around 90% of the total average traffic is contained in flows larger than 1MB. Since the available empirical CDF only contains 12 data points, we fitted a Pareto distribution and generated flows according to the fitted distribution in our simulations.

Threshold assignment. Deriving the optimal set of threshold Ω for MLFQ schemes is in general an open problem, although analytical solutions exist for some classes of distributions in the case of two priority levels [17]. In this paper, we used a threshold computation strategy, derived from PIAS [9], which solves an optimization problem based on a tandem of M/M/1 queues. Our problem formulation is an extension of the work in [9], for the sake of space we omit additional details. We numerically solved the problem through particle swarm optimization. The value of the thresholds we derived can be found in our code repository [29].



(a) Average normalized FCT (b) Tail normalized FCT

(c) $(0, 100\text{kB}]$ (d) $(100\text{kB}, 10\text{MB}]$ (e) $(10\text{MB}, +\infty)$

Fig. 5: Performance of the MP-MLFQ scheduling scheme under a Web Search [4] workload for K parallel paths (i.e., uplink ports) and N physical PQs per uplink port.

B. Comparison to other scheduling schemes

We compare the performance of the MP-MLFQ scheduler against the MLFQ scheme for different number K of spine switches, representing the degree of spatial diversity in the DCN. For both schemes we used their optimized thresholds.

Figs. 5a-5b show the average and 99th percentile nFCT when varying the offered load. Every point is obtained by averaging over 10^6 simulated flows. The results highlight two kind of scenarios. First, for a fixed number $N = 2$ of physical PQs per-port, we observe that MP-MLFQ always outperforms MLFQ, e.g., by around a factor $1.15\times$ at high loads, both in the average and tail completion time. This suggests that, given a DC network fabric with few queues available, our approach is always advantageous over MLFQ for the considered realistic workload. Second, we can compare MP-MLFQ and MLFQ for the same total number of *logical* priority levels. For MP-MLFQ we consider the orange curve corresponding to 4 parallel uplink paths and 2 physical PQs per port, which logically realize 5 priority levels, as explained in Sec. III. For MLFQ we consider the blue curve with 5 physical PQs (remember that in the schemes without spatial diversity every logical priority corresponds to a physical PQ on a ToR port). Notice that the nFCT of MLFQ is independent by the number of paths, as traffic is load balanced uniformly. Our results demonstrate that MP-MLFQ closely matches the nFCT

given by MLFQ, while requiring only two physical PQs per port. Thus, we can approximate a many-PQs scheduler with significantly less resources, by exploiting the spatial diversity.

In Figs. 5c-5e we break down our analysis by dividing the flows into three classes: mice, medium and elephant flows with sizes in $(0, 100\text{kB}]$, $(100\text{kB}, 10\text{MB}]$ and $(10\text{MB}, +\infty)$, respectively. These classes align with those defined in previous related work [7], [9]. We observe that MP-MLFQ leads to a more pronounced improvement in the average nFCT for mice end elephant flows. For mice flows, this is because MP-MLFQ is able to provide a finer granularity in the priority levels, and demote earlier the medium flows to lower priorities, thus avoiding the interference with mice flows that instead occur in MLFQ. For elephant flows, the improvement is due to the fact that majority of the elephant flows are load balanced (after step ② in Sec. III) into the same path used by mice flows. Since mice flows carry a small amount of traffic, elephant flows do not starve in the low priority queues at the SP scheduler. In contrast, for MLFQ all the elephant flows are served after both the *mice and medium* flows, thus leading to higher completion times.

VI. Conclusions

We have considered a leaf-spine architecture for a data center network. Previous scheduling approaches to minimize the FCT without a priori knowledge of the traffic flow sizes are based on Multi-Level Feedback Queueing (MLFQ) approach, whose performance improves by increasing the number of priority queues available in the switch ports.

Motivated by the limited number of available queues in practical implementations, we have proposed the Big-Spine switch abstraction, in which all the spine switches cooperate to offer a pool of logical priority queues that can be exploited to maximize the overall number of priority levels. On top of this abstraction, we have architected a novel scheduler, denoted as MP-MLFQ, and highlighted three alternative designs for programmable data-planes to realize MP-MLFQ transparently to user applications, avoiding any reconfiguration effort to end-host VMs and OS.

Our proposal is shown by simulations to reduce both the average and the 95th percentile FCT compared to MLFQ-based schemes. We expect in the near future to integrate it in a testbed including programmable NICs and P4-based switches, and quantify in a real-world setting the MP-MLFQ performance under the aforementioned alternative design choices.

Acknowledgments

This work was supported by the European Union's NextGenerationEU instrument, under the Italian National Recovery and Resilience Plan (NRRP), M4C2 Investment 1.3, "Telecommunications of the Future" (PE00000001), program "RESTART".

References

[1] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM CCR*, 2015.

[2] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM SIGCOM*, 2010.

[3] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," *ACM SIGCOMM CCR*, 2011.

[4] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *ACM SIGCOMM CCR*, 2009.

[5] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM CCR*, 2012.

[6] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: synthesizing existing transport strategies for data center networks," in *ACM SIGCOMM*, 2014.

[7] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal near-optimal datacenter transport," in *ACM SIGCOMM CCR*, 2013.

[8] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik, "Scaling distributed machine learning with in-network aggregation," in *NSDI*. USENIX Association, 2021.

[9] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *NSDI*. USENIX Association, 2015.

[10] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *ACM SIGCOMM*, 2018.

[11] H. Lim, J. Kim, I. Cho, K. Jang, W. Bai, and D. Han, "Flexpass: A case for flexible credit-based transport for datacenter networks," in *ACM EuroSys*, 2023.

[12] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "RDMA over commodity Ethernet at scale," in *ACM SIGCOMM*, 2016.

[13] L. Schrage, "A proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, 1968.

[14] N. Gautam, *Analysis of queues: methods and applications*. CRC press, 2012.

[15] V. Dukic, S. A. Jyothi, B. Karlas, M. Owaida, C. Zhang, and A. Singla, "Is advance knowledge of flow sizes a plausible assumption?" in *NSDI*, 2019.

[16] M. Nuyens and A. Wierman, "The foreground–background queue: a survey," *Elsevier Performance evaluation*, 2008.

[17] K. Avrachenkov, P. Brown, and N. Osipova, "Optimal choice of threshold in two level processor sharing," *Annals of Operations Research*, 2009.

[18] I. A. Rai, G. Urvoy-Keller, and E. W. Biersack, "Analysis of LAS scheduling for job size distributions with high variance," in *ACM SIGMETRICS*, 2003.

[19] Y. Lu, G. Chen, L. Luo, K. Tan, Y. Xiong, X. Wang, and E. Chen, "One more queue is enough: minimizing flow completion time with explicit priority notification," in *IEEE INFOCOM*, 2017.

[20] A. G. Alcoz, A. Dietmüller, and L. Vanbever, "SP-PIFO: Approximating Push-In First-Out behaviors using Strict-Priority queues," in *NSDI*, 2020.

[21] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *IEEE INFOCOM*, 2013.

[22] J. Hong, S. Jeong, J.-H. Yoo, and J. W.-K. Hong, "Design and implementation of eBPF-based virtual TAP for inter-VM traffic monitoring," in *14th International Conference on Network and Service Management (CNSM)*. IEEE, 2018.

[23] J. Wang. (2017) Accelerating VM networking through XDP. [Online]. Available: https://events19.linuxfoundation.cn/wp-content/uploads/2017/11/Accelerating-VM-Networking-through-XDP_Jason-Wang.pdf

[24] IEEE 802.1p standard. [Online]. Available: https://www.ieee802.org/

[25] Netronome. (2024) Agilio CX 2x40GbE. [Online]. Available: https://netronome.com/agilio-smartnics/

[26] S. Miano, G. Lettieri, G. Antichi, and G. Procissi, "Accelerating network analytics with an on-nic streaming engine," *Computer Networks*, 2024.

[27] S. Miano, X. Chen, R. B. Basat, and G. Antichi, "Fast in-kernel traffic sketching in eBPF," *SIGCOMM Comput. Commun. Rev.*, vol. 53, no. 1, 2023.

[28] H. Namkung, Z. Liu, D. Kim, V. Sekar, and P. Steenkiste, "Sketchovsky: Enabling ensembles of sketches on programmable switches," in *NSDI 23*. USENIX Association, 2023.

[29] A. Cornacchia and G. Sviridov. MP-MLFQ code repository. [Online]. Available: https://github.com/alessandrocornacchia/MP-MLFQ