POLITECNICO DI TORINO

Master Degree course in Communications and Computer Networks Engineering

Master Degree Thesis

Priority-driven load balance for low-latency datacenter networks

Supervisors
Prof. Paolo Giaccone
Prof. Andrea Bianco

Candidate
Alessandro Cornacchia

Sessione di boh

Acknowledgements

Thanks

Abstract

Major cloud computing providers agree on denoting the flow completion time (FCT) as the primary goal to achieve in the design of a data center network (DCN). This is motivated by the fact that latency affects the performance of most cloud computer applications, such as web search, video streaming and social networking, with a direct impact on quality of experience as perceived by end users, hence ultimately on revenues.

Many existing approaches in literature rely on prioritization mechanisms at flow or packet level and schedulers in the network to reduce the average FCT, albeit for some applications also tail latency is important. To this purpose, the Shortest Remaining Processing Time (SRPT) scheduling policy is proven to be optimal when the job size is known in advance. Unfortunately this information is rarely available, therefore some proposals emulate the complementary scheme that give service first to flows that have transmitted less (Least Attained Service), exploiting a finite number of priority levels. All flows start with the highest priority and are progressively demoted to lower priorities as they receive service. The effectiveness of these algorithms, is largely improved augmenting the number of priorities. However, DCNs are usually realized with low cost commodity devices, where only few queues per port - typically two - are vacant.

The contribution of this work is to investigate the feasibility and evaluate the performance of a design which exploits multi-pathing - offered by DCN Fat-Tree topologies - to intelligently route flows across the switching fabric depending on their priority, so as to better segregate latency demanding flows. In other words, the key observation is that load balancing and prioritization can be jointly analyzed, in what multiple outgoing links together provide more priority queues than a port alone. Instead of assigning flow priorities with a scope limited to single interfaces, the problem can be addressed at data center level considering the set of links towards the flow destination as a whole. Unlike aforementioned techniques, longest flows are demoted across all priority queues of all exit interfaces, thus implying that routing choices are directly based on priority.

First, an analytical queueing model is provided for the setting of optimal parameters. Then, it will be shown through queueing model experiments that the proposed strategy indeed it is helpful under the few priority queue regime, but that long flows can be excessively penalized. Finally, extensive large scale packet level simulations in a real data center topology are conducted with Omnet++ discrete-event simulator, in order to validate the results obtained with previous steps.

Contents

List of Figures

Chapter 1

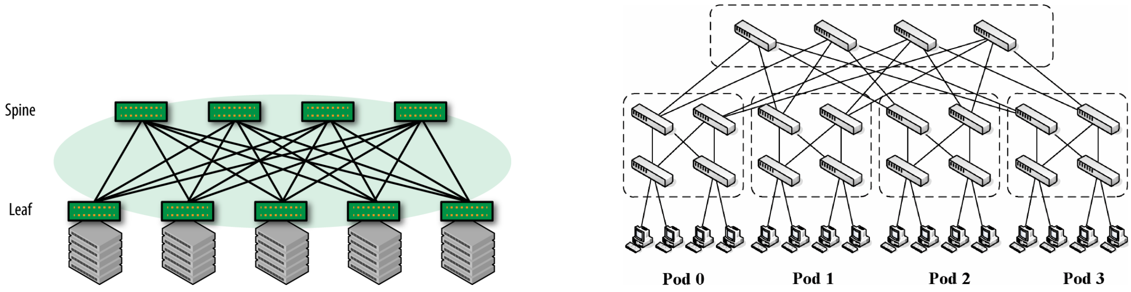
Datacenter networks

In the last decades, with the spreading of cloud services accessible to anyone, computing has undergone a remarkable evolution. Encouraged by the astonishing growth of virtualization technologies in the IT industry, as well as the availability of storage and chips at ever-more modicum prices, over-the-top (OTT) players like Google, Amazon and Microsoft, have been building datacenter hotspots all around the world. Most of the applications any sector of modern society relies on, such as commercial and financial services, Web search, scientific computing, on-demand video streaming, recommendation systems, not to mention social networking and online gaming, more often than not run inside one of their datacenters' infrastructure. Indeed, from small enterprises to large corporations, it has been a common cost-saving strategy to offload, up to a certain extent, the deployment and operation of their own information systems to third parties, the cloud service providers. After all, it is well-known in engineering, how much resource concentration enables better design and ease of mantainance.

Among the major challenges posed by cloud computing paradigm, certainly there is the design of a huge communication network, hosting hundreds of thousands of servers, that is required to simultaneously provide high throughput and low-latency, while guaranteeing uninterruptible service continuity and lending itself to relentless expansion. The peculiar requirements of the datacenter environment and the lack of flexibility of traditional TCP/IP stack, have pushed the development and adoption of unconventional approaches, such as centralized control and network softwarization (SDN), laying the premises for a wider transformation process in the network industry. In this sense, datacenters can be credited to having represented natural incubators for the evolution of telecommunication networks happening during the last few years. As a matter of fact, virtualization technologies and control plane programmability soon would have become disruptive innovations in networking ecosystem, that started to experience the same changes the IT world faced years before. As a result of this process, nowadays, ISPs are converting their infrastructure towards the same solutions, re-architecting PoPs as small-scaled modern datacenter, with massive employment of virtualization as regards network functions and devices (NFV). Similarly, the next generation mobile network, 5G, is going to base both its core and edge functions on the same paradigm, as revealed by its standardization and by the investments in multi-access edge computing (MEC).

1.1 Interconnection network design

The typical structure of a datacenter, as shown in Fig.**??**, is comprised of many *racks*, interconnected among each other thanks to a common network infrastructure. A rack is nothing else than a group of servers physically co-located in a common cabinet, attached to the same *Top of Rack* switch (ToR) and thus separated by a single hop. In achieving the goal of hosting an enormous amount of servers, the principal bottleneck often results in being the interconnection fabric, usually referred to as *Data Center Network* (DCN). Ideally, it should act as a huge switch, able to provide maximum-rate communication among servers, that is their NIC's access capacity. At high level, there are fundamentally two ways for practically realizing such infrastructure. The first choice is to rely on complex specialized solutions, like InfiniBand, or high-performance IP devices with many ports, that successfully provide bandwidth for thousands of nodes, however incurring in high deployment and management costs. Conversely, the second possibility is to build the network infrastructure by simply leveraging on commodity off-the-shelf switches that are cheaper, already on economy of scale and fully compatible with existing hardware and operating systems, just as large distributed clusters are made of general purpose cheap computers. This is usually the design pursued by principal cloud providers, as recently disclosed by themselves. A major drawback of the latter strategy, is the difficulty in providing full-rate communication among hosts in different racks. In other words, depending on traffic patterns and especially network topology, it is very hard to place enough inter-rack capacity - usually referred to as *bisection bandwidth* - to satisfy the collective demand of all the racks. For this reason, the topology design plays a fundamental role for the feasibility of scalable and cost-beneficial large datacenters. It is worth noticing that sometimes, building a DCN with full bisection bandwidth is unnecessary and cost prohibitive, therefore it is commonly accepted some degree of *oversubscription*, meaning that the ratio between total intra-rack and bisection bandwidth is greater than 1. This choice is acceptable when applications are mostly rack local and statistical multiplexing is beneficial. In general, a non-oversubscribed network allows greater flexibility in resource allocation and deployment of applications across racks, leading to higher utilization which is also important for big cloud providers.



(a) Leaf-Spine

(b) 3-layers Fat-Tree

Figure 1.1: interconnection networks.

Two examples of topology design are provided in Fig. 1.1. The *Leaf-Spine* (Fig. 1.1a), is a typical hierarchical architecture, where the interconnection between leaf and spines switches form a bipartite graph. Despite being popular in campus networks, scaling to thousands of servers would require having spines with lots of high-capacity ports . A more flexible solution dates back to the early telephone network, when Charles Clos had to solve a similar problem and came up with its proposal for multi-stage switching fabrics. Indeed, the *Fat-Tree* (Fig. 1.1b), representing the most widely adopted topology, is a folded Clos network. The main plus of this configuration is the possibility to host as many servers as desired, using only switches with a fixed number of port k, eventually providing full bisection bandwidth, just by recursively adding new layers, or stages. Fig. 1.1b shows an example of 3-layer Fat-Tree with 4 ports switches. Essentially, it is a recursive design, in which a l-layer network is built by connecting k blocks, called *Point of Delivery* (POD). Note that in terms of PODs, a Fat-Tree is actually a Leaf-Spine with $k/2$ spines and k POD leafs. Each POD has the same structure of a (l-1)-layer network, unless for the fact that $k/2$ ports from those nodes that were the spines of layer l-1 have been used for interconnecting the POD to the new spines of the l-layer architecture. To put it differently, the l-layer DCN can become a POD for the (l+1)-layer DCN by disconnecting $k/2$ PODs from the spines to and connecting them to a new row of spines (core switches). The elementary POD is the 2-layer architecture, that can host at most $k/2)^2$ servers. Thus, a datacenter with l stages can support up to $k^l/2^{l-1}$ machines, and this is the reason behind popularity of Fat-Tree, other than its compatibility with main requirements for datacenters, such as path diversity, which is important for fault-tolerance and traffic engineering. Throughout this work, in particular, it will be investigated a flow scheduling solution that exploits this multi-path property.

1.2 Traffic engineering

Large data centers represent a challenging environment for network designers. They host many thousands of servers running a myriad of applications belonging to tenants with heterogeneous QoS demands on a very physically circumscribed network. Data travel for no more than few hundreds meters with negligible propagation delays, through links of huge capacities up to 40Gbps. In such a context, the traditional TCP/IP protocol stack alone operates inefficiently, hence custom transport protocols, traffic control and traffic management techniques have been deployed. In this section will be first pointed out common objectives for network operations, then briefly reviewed the traffic characteristics and the subsequent issues they pose, in order to lay down the fundamentals of this work and justify the choices for traffic generation undertaken in its simulative part.

1.2.1 Traffic control objective: FCT

There is unanimous consensus among data center providers in putting effort to minimize the *Flow Completion Time* (FCT) metric, that is the all-up delay since when data are requested by a client application to the time they are at its disposal. The reason behind this interest is that latency directly impacts the quality of experience (QoE) as perceived by end users and ultimately operator revenues. For interactive Web applications and online services, the responsiveness determines the number of users on the long run. Also, low-latency communication also enables flexible inter-rack deployment of micro-services across the network. Therefore, a common goal is to find efficient traffic control algorithms to handle latency sensitive flows, while maximizing network utilization which is also important for cost savings. Usually, different techniques are compared by measuring the *average* flow completion time.

1.2.2 Traffic properties

Several studies has been conducted in literature to characterize the main properties of traffic in data centers. Traffic characteristics are highly dependent on applications, that determine flow sizes, flow arrival patterns and the requirements from a network perspective. Common applications running in datacenters and for which there exist traffic studies are Web search, data mining (e.g. Hadoop) and cache services. For Web search queries and the corresponding responses, for example, few packets are enough, so they generally comprise short flows. Instead, other services such as data mining and batch computing tasks, may transfer large amount of data. Additionally, long-lived background connections of large size are continuously present for VM migration, backups, consistency updates and data replication. A common scenario observed by different studies is that the majority of flows are short, but overall they do not significantly contribute to the total traffic, which is mainly carried by few large flows. Measurements from a large cloud provider production datacenter reveal that 80% of flows are are less than 10KB and almost all flows (99%) are less than 100MB. However, more than 90% of transfered bytes are in the 1% of flows greater than 100MB. Facebook [12] shared its traffic statistics and reported the median flow size to be 3KB for Web Search and 100KB for data mining with Hadoop, while the tail flow size 10MB and 100MB respectively. Similar trends are claimed by other authors [banson, pfabric, dctcp], although flow sizes are slightly different depending on datacenter services.
Therefore, the first remark is that a variety of flows with different sizes coexists on the same DCN, the majority of which lasts a couple of packets only.

Frequently, different flow sizes happen to be associated with different QoS requirements, whose knowledge drives in the design of transport protocols and traffic control algorithms and meet application demands. Short flows are typically query traffic, very sensitive to latency, stemming from the *Partition/Aggregate* pattern which is widespread in distributed computing, from social networking content composition to retail and reccomendations. According to this paradigm, a task requested at higher layer to an *Aggregator*, is broken into simpler units that are dispatched along a tree-like logic to lower level aggregators, that may further break the request into smaller pieces, until they're finally handled by worker nodes. The responses of the workers, when ready, are then conveyed back along the same reversed tree logic to aggregators that put together the results. Key in this process is that it must complete within strict deadlines, on the order of 10-100ms, that are determined in order to satisfy the worst-case latency tolerated by the Service Level Agreement (SLA) with customers, or tenants in cloud computing jargon. Ideally, application developers should not be concerned of network delays and should be entitled to employ the time before deadlines to improve final results thus end user satisfaction, without resorting to the implementation of complex ad-hoc solutions to compensate for network inefficiency. On the other hand, long flows are mainly comprised of update flows that carry fresh data for parallel computing jobs (e.g. MapReduce), or transfers for data replication across servers, also located in different facilities in the globe. They are throughput-oriented flows, demanding considerable bandwidth, but they are not sensitive to delays. Section **??** will highlight some issues deriving from the mix latency critical flows arising from the partition/aggregate workload with background long-lived connections.

Finally, one may also be interested in understanding the communication patterns to tailor traffic engineering choices and perform capacity planning. For instance, knowing the degree of traffic rack locality would allow better awareness in deciding the oversubscription ratio. To this extent, it is difficult to draw general conclusions and prior studies have given contradictory results: some work in literature (cite traffic-in-the-wild) reports a marked locality, whereas some other observes completely different patterns with traffic not at all rack local. The reason lies in how applications are deployed across servers and clusters. In the Facebook data center each machine in assigned a precise role and machines with a same role are grouped in the same rack. Since Web Servers machines talk primary to Cache Followers machines, there is substantial intra-rack communication for this service. Conversely, Hadoop traffic stays 75% of the times in the same rack. What is true in general is that flow arrival rates are quite high, as many as thousands flows per second per server. Combined with the fact that the majority of them is short and that their destinations is often randomized at application level to avoid hot-spots, it follows that the traffic matrix of a datacenter network has been revealed to be very fluctuating, unstable and difficult to predict.

1.2.3 Challenges in traffic control

The traffic characteristics illustrated so far, produce undesired pattern and effects on data center networks, which standard TCP transport suffers specifically. They are *queue buildup* and *incast* problems.

1. *Queue buildup.* On the basis of well-known congestion control mechanisms, traditional TCP sources increase their window until they experience either a timeout or a packet drop, resulting in the usual sawtooth pattern. This causes switch queues to grow, mainly due to long connections which have time to inflate enough their window. In the context of traffic depicted in the previous section, this is especially problematic because long flows sharing the same queues with short flows harvest much of the buffer space, causing short flows either to queue behind them and increase their latency, or to experience more penalizing packet drops. The queue buildup impairment is even more severe in data center networks, since commodity switches are generally cheap shared buffer architectures, where packets from different ports are stored logically in the same memory space. Thus, high utilization of a single port could degrade the performance of flows traversing a different interface of the same device.

2. *Incast.* The incast problem arises from the Partition/Aggregate pattern common

in data centers applications. After the aggregators assign portions of the same task to different workers, the flows containing the responses tend to be cluster at the same time on the same switch ports, on the way back towards aggregators. Such a flow synchronization, in conjunction with the queue buildup phenomenon, causes synchronized drops even if the flows are short, as in the case of responses to Web queries. Packet drops, therefore timeouts, are not acceptable for deadline constrained flows, typical of the Partition/Aggregate pattern.

1.2.4 Datacenter TCP(DCTCP)

Many techniques for traffic control are possible in order to counteract the previous impairments: they include transmission rate control, traffic shaping, routing and load balancing, scheduling and priority schemes. In the following of this work it will be explored a solution which combines prioritization and routing. However, a milestone for transport protocol design in data centers has been Datacenter TCP (DCTCP). It actually consists in a very concise modification to TCP which leverages Explicit Congestion Notifications (ECN) from the network to properly modulate the window size of the sources.

DCTCP

The basic idea behind DCTCP is that queues in the network should be kept as empty as possible to avoid large backlogs that increase latency and do not leave enough headroom to absorb burst arrivals, occurring for example due to incast. Instead of pushing the window to grow until a packet drop is detected, the DCTCP transmitter slows down proactively depending on the level of congestion on the bottleneck link. Network switches mark every packet with a congestion signal as soon as the queue where the packet is stored exceeds a given occupancy K. The TCP receiver then conveys back the congestion signaling to the TCP transmitter, setting the ECN-Echo bit to 1 in its next ACKs. Finally, the TCP sender mantains an estimate α of the fraction of marked packet on an interval of roughly one RTT and modulates the window as:

$$\text{cwnd} = \text{cwnd} \times (1 - \alpha/2)$$

This way the window size is gently reduced upon mild congestion — note that only in case α=1 it is cut in half as in standard TCP — still ensuring high throughput, but reducing its aggressiveness.
It has been proven that DCTCP effectively succeed in lowering the amplitude of queue oscillations to $O(\sqrt{\text{BDP}})$ from the $O(\text{BDP})$ of TCP, being BDP the bandwidth-delay product, while not losing throughput for a proper setting of the marking threshold K. Note that the only requirement from the network is to configure switches with an AQM scheme to mark packets. In practice this can be achieved configuring RED, already available in most devices, so that it marks based on the instantaneous queue length and with a unique high and low threshold equal to K.

Chapter 2

Reducing FCT with flow prioritization

To the purpose of minimizing the average flow completion time (FCT), a common strategy is to treat short flows with tight latency constraints differently from the others. This can be managed with prioritization and scheduling algorithms, for which there already exist numerous theoretical studies. In the following of this chapter will be reviewed the most significant of them, with care to their application to datacenter networks.

2.1 Theoretical scheduling background

At high level, scheduling policies can be classified into two categories, depending whether or not flow properties — such as size and deadline — are known a priori.

2.1.1 Flow-aware disciplines

Flow-aware disciplines are those techniques that assumes the job characteristics are precisely known before initiation. While this is often the case for many systems in other industries like manufacturing, it is not always so in communication networks. Sometimes the flow size is known exactly, for example when a server is requested a static object (e.g. a static Web page, a file transfer) or it can be roughly estimated, but generally speaking this information is either not available or it involves undesired modifications of the application layer.
The optimal approach to minimize the job completion time for an offline system is the *Shortest Job First* (SJF) discipline that consists in serving jobs in decreasing order with respect to their size. However this policy is not suitable for dynamic contexts where new jobs can arrive at any time instant. For such scenarios has been adopted a preemptive version of SJF, known as *Shortest Remaining Processing Time* (SRPT), which chooses first the job with shortest time left to its completion, or equivalently in the context of flow scheduling, the smallest amount of bytes left. SRPT has been proven for long to be the optimal policy for minimizing the average response time (i.e. FCT) in a single server system, regardless of the serving time and inter-arrival distribution [13].

2.1.2 Flow-agnostic disciplines

In absence of precise knowledge about the flow length, a very effective policy is the dual approach to SRPT, the so called *Least Attained Service* (LAS) or *Foreground-Background* (FB). LAS is a preemptive scheduler which gives service first to the flow that has transmitted less bytes, serving in processor sharing when there are ties among flows. In other words, a job retains alone the processor until it is has received the same service of another jobs in the system or it is preempted by a newly arrived shorter job. If no fresh jobs arrive, those in the systems prosecute sharing the processor. The main insight of LAS is to exploit the increasing knowledge about the flow size gained during its service. In fact, the LAS scheduler becomes more and more confident that a given flow is a large one, as further of its bytes are transmitted.

LAS is optimal among the flow-agnostic disciplines when the hazard rate of the flow size distribution is a decreasing function [1]. The hazard rate is defined as the ratio of the probability density function $f_X(x)$ to the survival function:

$$h(x) = \frac{f_X(x)}{1 - F_X(x)}$$

and it is a function that represents the instantaneous failure rate of a quantity. For instance, it can be seen as the likelihood that the flow size X ends at value x given that x bytes have already been observed. As a general rule, heavy-tailed distributions exhibit a decreasing $h(x)$, whereas $h(x)$ is increasing for light-tailed distributions and constant for the exponential distribution. Thus, LAS works at best under job size distributions that present high variability, which is a case of interest since they well model datacenter's traffic where a majority of short flows coexists with few very large flows.

At this point, it is interesting to better understand how LAS compares to other disciplines, such as PS, not only on average but in depth with respect to flow sizes, in order to quantify its impact on small and large jobs. Also, it would be useful to bound the sub-optimality of LAS with respect to the SRPT flow-aware scheduler. To this purpose, the authors of [11] compared different distributions with increasing coefficients of variation, under LAS, PS and SRPT disciplines. The *coefficient of variation* C, is a single number measure of the variability of a distribution and it is expressed as the standard deviation σ normalized to the mean of the distribution μ:

$$C = \frac{\sigma}{\mu}$$

The comparison was based on the *average conditional completion time*, that is the average completion time of flows belonging to a given size. Formally, denoting as T the random variable associated to the completion time and X as the random variable associated to the flow size, the average conditional completion time is defined as $\mathbb{E}[T|X = x]$. This is a rather practical quantity to tell how the FCT vary with flow sizes and to highlight unfairnesses brought by the scheduling policy.

The distributions taken into account were the negative exponential distribution, whose probability density function rapidly converges to zero, and a set of Bounded Pareto distributions with varying C, as example of heavy-tailed distributions. Their probability density functions are given by:

$$f(x)_{Exp} = u(x)\mu e^{-\mu x}, \qquad \mu \geq 0$$
$$f(x)_{BP} = \frac{\alpha k^{\alpha}}{1-(k/p)^{\alpha}} x^{-(\alpha+1)}, \qquad k \leq x \leq p, 0 \leq \alpha \leq 2$$

For the Pareto distributions, different setting of their parameters correspond to different C values, but generally it holds $C \gg 1$ which means essentially high variability. For the negative exponential distribution C is constant and equal to 1.

- **LAS vs PS.** They found that for Pareto distribution with $C \geq 6$, LAS outperforms PS above the 99th percentile of the flow size distribution, that is the conditional completion time $\mathbb{E}[T|X = x]$ is better under LAS than under PS for all but the largest 0.3% flows. Additionally, even the penalty of the longest flows is within a factor 2. Notice that it is reasonable to expect some slowdown for very long flows due to starvation, especially when an elephant flow meet a longer one, which is queued behind it. Instead, for exponential distribution, flows with size above the 80th percentile are severely penalized using LAS, but overall the average FCT is still better.
  The final message to be derived here is that LAS is always beneficial for the average flow completion time in respect of PS for distributions with $C \geq 1$.

- **LAS vs SRPT.** This comparison is of interest because it shows quantitatively the performance gap with the optimal policy. Indeed, it was provided an analytical expression for the worst case penalty of LAS with respect to SRPT in terms of mean conditional completion time, valid for every continuous time finite mean and finite variance distribution. In particular, when applied to Pareto, LAS is very close to SRPT for all job sizes, with a penalty of 1.25 under all load conditions. This result is of remarkable importance as essentially states that for heavy-tailed distributions with high variability there is no significant gain in knowing the flow size beforehand, but good performances can be achieved with the LAS policy, which is simpler to implement.

In short, the ideal scheduler is SRPT if detailed flow information are available at transport upon flow initiation, alternatively LAS is the best choice provided that flow sizes present high variability. Any practical design proposed in literature and revised in the following sections aim to approximate any of these targets.

2.2 State-of-the-art solutions in practical networks

Theoretical results, both for flow-aware and flow-agnostic scheduling disciplines, refer to a scenario with a single link and do not account at all for the implementation complexity. A major restraint of LAS and SRPT that limits their practical applicability are the fine-grained decisions they adopt for flow scheduling. To determine the next job to serve, it is required to mantain per-flow state information and perform comparisons among them at every packet transmission. These operations can be very complicated to implement at high line rates with thousands of simultaneous flows, as typically happens in DCNs. (Sec. **??**). Moreover, in a data center network many servers are connected through a multi-stage switching fabric (1.1), which implies that, actually, from source to destination multiple links are traversed, hence optimizing local decisions does not ensure global optimality. As a trivial example, consider a situation with three server s_1, s_2 and s_3 and three flows $f_1 : s_1 \longrightarrow s_2$, $f_2 : s_1 \longrightarrow s_3$ and $f_3 : s_2 \longrightarrow s_3$. With this setting f_1 competes against f_2 in the access interface of s_1, while f_2 competes with f_3 in the egress towards s_3. If a local scheduler in the outgoing interface of s_1 decide to serve f_2 before f_1, but a second independent scheduler in the last interface ahead of s_3 prioritize f_3 over f_2, the choice of the former scheduler would be vanished by the conflicting choice of the latter, because f_2 would delay f_1 despite being queued back f_3. Therefore, the optimal scheduling pattern could be achieved by means of a global network view.

The state-of-the-art proposals that address these issues are pFabric [3] and PIAS [4]. They have been taken as the reference model of this work, which will try to extend their fundamental insights.

2.2.1 pFabric: targeting ideal schedulers

Alizadeh et al. provide a general representation about the problem of scheduling flows over the datacenter fabric. Specifically, they abstract the DCN as a giant switch interconnecting all the servers, as shown in Fig. 2.1. In this representation all the links are assumed to be unidirectional, so that leftward there are the servers' access NICs and rightward the ToR egress interfaces towards the same servers.

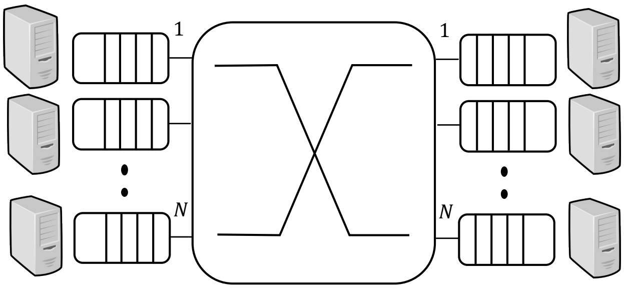


Figure 2.1: Datacenter fabric abstraction as a giant switch

Ideal flow scheduling in DCN

Assuming that the fabric can sustain maximum throughput and that flows compete against each other only at ingress and egress interfaces, the optimal solution to minimize the flow completion time can be found solving a NP-hard problem known as *sum-multicoloring*. Nevertheless, a simpler greedy algorithm exists and it has been proven to be very close to the optimum. Authors referred to this algorithm as the *Ideal* algorithm and used it as a baseline for the evaluation of pFabric. Briefly, it consists in a maximal flow scheduling, where at every new flow arrival or depart, flows are sorted in ascending

order of data remaining up to their completion and served with this order. Flows are not served until there is another flow with less data remaining traversing the same ingress or egress port. Despite being a simplified model, still the *Ideal* algorithm is a valid benchmark for any design that targets flow completion time minimization, since it is a sort of best case latency at a desired level of load and over an ideal interconnection network.
At this point, pFabric showcased that it is possible to achieve nearly ideal performances in a distributed way, treating each interface on its own with local scheduling decisions and simultaneously rely on minimal transport strategies. In particular, their work can be summarized in the following key aspects.

1. *Knowledge of flow requirements.* It is assumed to know at transport layer flow sizes or flow deadlines for deadline-constrained traffic.

2. *Prioritization.* Schedulers in the network are priority schedulers. Depending on the scheduling objective, the packets encode with a priority a different metric, on which the scheduler choices are based on. For example, to approximate SRPT on every single link, the priority may indicate the amount of bytes still to transmit for a given flow. Packets are dequeued and dropped according to their priority.

3. *Simple rate-control.* Rate control at end hosts is minimal and aims just to avoid persistent congestion. In fact, contrary to DCTCP (Sec. 1.2.4), even if queue sizes grow, only the performance of few long flows is significantly impacted, since it is used a detailed prioritization mechanism both for packet transmission and drops. Some shrinkage to standard TCP have been described to realize such minimalistic transport.

Approximate optimality with priority queues

As already mentioned, implementing such a prioritization scheme on available commodity devices is very challenging and for sure would require hardware modifications. The interested reader could find a possible digital design in the paper of pFabric. The same source, however, provided also a straightforward solution readily deployable with current switches. The idea is to coarse the granularity of the scheduler by adopting only a finite number of priority levels. Then, packets with different priorities are enqueued in corresponding priority queues (PQs) and handled with traditional network schedulers (SP, WRR, WFQ, etc.). In pFabric it is employed a Strict Priority (SP) scheduler, but in general other choices are not precluded [5]. In short, LAS and SRPT schedulers, as well as other disciplines, can be emulated by tagging packets with a priority label and leveraging separated queues in network interfaces. The typical number of priority queues available in datacenter switches ranges between 2 and 8, albeit it often occurs that some of them are reserved — more details in Sec. 2.2.2. Of course, increasing the number of PQs results in better approximation of the ideal scheduling, that implicitly would correspond to having an infinite number of PQs. In fact, the ideal scheduler directly compares each other all priorities of the packets in the buffer.

Finally, the FCT gain obtained with this system largely depends on the way flows are clustered in priority levels. This underlies the need of a careful tuning of a set of thresholds which split packets among the priorities available. A dedicated section of this work illustrates some criterion to choose them (§??).

2.2.2 PIAS: the reference model

The next step with respect to pFabric as well as the starting point of this work is PIAS: Practical Information-Agnostic Scheduler [4]. This proposal is the first which addresses the scheduling problem without the assumption of an exact prior knowledge about the flow size. Instead, PIAS tries to mimic a LAS scheduler exploiting the sole knowledge of the distribution of flow sizes rather than their precise values. Surprisingly, when compared to pFabric it delivers very similar performances for short flows, that are the more critical ones (the gap is within 4.9%). At the same time, however, PIAS preserves ease of deployment with the current switch hardware and the standard distributed congestion-control algorithms. Summarizing, the two main design principles of PIAS are:

1. *Flow-agnostic.* It requires only the knowledge of DC-wide flow size distribution, which can be easily estimated once and updated dynamically. Authors do not account for any heterogeneity of the distribution across different racks [12], in what the problem would become quite difficult ot treat with analytical methods. Also dynamic changes of the distribution along time have been ignored, but the architecture is flexible enough to adapt to this case.

2. *Low complexity.* It should be compatible with legacy TCP/IP protocol stack and readily deployable without touching the hardware of existing devices.

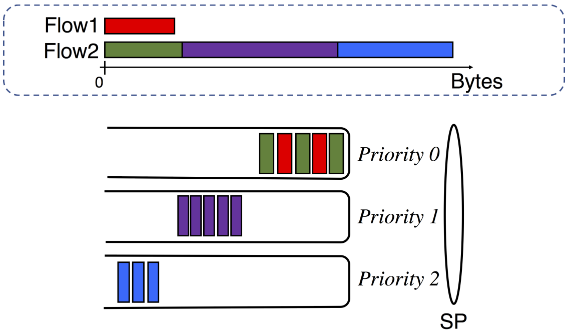


Figure 2.2: PIAS overview

PIAS embraces a *Multi Level Feedback Queue* (MLFQ) mechanism to resemble the LAS policy. MLFQ essentially apportion flows in a finite number of priority queues as proposed in pFabric, but in absence of flow size information flows are dynamically moved across priority queues. In more details, each packet of a given flow is mapped to a priority level, inversely proportional to the bytes it has already sent since its beginning. All flows starts with the highest priority, then longest flows are progressively demoted to lower priorities. Packets of same priority are buffered in the corresponding PQ according to a First-In First-Out (FIFO) order, while packets belonging to different priority queues are scheduled in Strict Priority (SP). In this way, packets belonging to short flows are always prioritized over those belonging to long flows, mimicking LAS but without incurring in the complexity of its implementation, which would require many comparisons. Figure 2.2 clarifies the demotion mechanism. $Flow_1$ is a mice flow and it is entirely served at the highest priority, whereas $Flow_2$ is an elephant flow and it is gradually de-prioritized up to the last PQ. Notice that differently from pFabric, all flows traverse highest priorities in their initial lifetime, since their size is initially obscure. Hence, longer jobs constitute a small impairment for short ones, that motivates the gap with pFabric.

A key point for this kind of systems is how to choose the demotion thresholds. In fact, especially when few queues are available, an unbalanced threshold setting may lead to severe performance degradation. On one hand, thresholds too small cause premature flow demotion and delays for short flows that get mixed with long flows, on the other hand if the thresholds are too large, medium and elephant flows overstay in high priorities, resulting again in worse FCT for delay sensitive flows. The next section will present simple heuristics and a more refined optimization to find the set of thresholds.

2.3 The problem of demotion thresholds

The same authors of PIAS proposed a queueing model to mathematically describe the dynamics of the system. It has to be remarked that the queueing model captures only the average flow completion time on a single interface. Therefore, it is assumed that the flow size distribution is homogeneous over the datacenter fabric so that bottleneck links observe all the same distribution. With this assumption, the average FCT over the whole fabric is just a linear rescaling of the average FCT on a single link, therefore the performance index of any set of thresholds obtained with the model is meaningful for the whole DCN as well.

In the following of this section, first it is formalized the queueing model along with its parameters, then it is formulated a non-linear minimization problem that can be used to optimize demotion thresholds. Finally, two other trivial heuristics for threshold assignment are reviewed.

2.3.1 Stochastic queueing model

The system, shown in Figure 2.3, is thought as a tandem of N subsequent M/M/1 queues. The customers are flows of size X extracted from a given distribution with cumulative function $F(x)$ and arriving according to a Poisson process of intensity λ. Queues are lazy and able to serve at most a maximum amount of bytes for each flow, that depends on the demotion thresholds. When customers enter the system they are initially served by the first queue, then either they leave the system if all of their bytes have been served, or they are demoted to subsequent queue, up to queue N.

Let be $Q_p(1 \leq p \leq N)$ the N priority queues. Denote with α_i the threshold after which a flow changes its priority from $i-1$ to i, where higher priorities correspond to lower indexes and $i \in [0, N]$. The upper threshold is always set to $\alpha_N = \infty$ so that all the

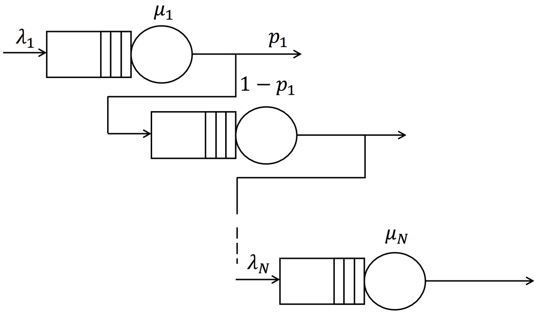


Figure 2.3: Queueing model

largest flows stay in the last queue, similarly $\alpha_0 = 0$ for simplicity. Each queue Q_i can serve at most $\alpha_i - \alpha_{i-1}$ bytes of any customer. In the real MLFQ system (Sec. 2.2.2), all queues of a given port are orchestrated by a Strict Priority (SP) scheduler, however this would make the queues dependent on each other complicating the analysis. To avoid the issue, the queues are treated as if they are independent and their priority hierarchy is taken into account by adjusting their drain rate μ_i. In fact, if μ is the overall link capacity, subsequent queues in the tandem are assigned a drain rate μ_i equal to the residual bandwidth left after servicing previous queues. In practice, denote with ρ_i the load insisting on Q_i, then

$$\mu_i = \mu \prod_{j=1}^{i-1} (1 - \rho_j) \tag{2.1}$$

Next, for the sake of conciseness let's indicate with $\theta_i \triangleq F(\alpha_i) - F(\alpha_{i-1})$ the probability that a new flow has size in $[\alpha_{i-1}, \alpha_i)$ and abbreviate with the notation $\bar{F}(x) = 1 - F(x)$ the survival function of X. The target is to derive the subsequent arrival rates λ_i. After service in queue i, a customer leaves the system with a probability p_i, that depends on flow size distribution. In fact, customers who leaves the systems after the i-th queue are the flows with size in $[\alpha_{i-1}, \alpha_i)$, among those with size in $[\alpha_{i-1}, \infty)$. Therefore, p_i can be expressed renormalizing the probability θ_i as:

$$p_i = \theta_i \,/\, \bar{F}(\alpha_{i-1})$$

Consequently, from the definition of θ_i

$$1 - p_i = \bar{F}(\alpha_i) \,/\, \bar{F}(\alpha_{i-1})$$

If the system is stable, the arrival rate of queue i is nothing else than the output rate of queue $i-1$, weighted by the probability of remaining in the system. Therefore:

$$\lambda_i = \lambda_{i-1}(1 - p_{i-1})$$

By iterative substitution it is possible to get a general expression for every λ_i that depends only on the flow generation intensity λ and the arrival rate at the previous queue λ_{i-1}.

Indeed:

$$\begin{aligned}
&\lambda_1 = \lambda && 1 - p_1 = \bar{F}(\alpha_1)/\bar{F}(\alpha_0) = \bar{F}(\alpha_1)\\
&\lambda_2 = \lambda_1(1 - p_1) = \lambda\bar{F}(\alpha_1) && 1 - p_2 = \bar{F}(\alpha_2)/\bar{F}(\alpha_1)\\
&\lambda_3 = \lambda_2(1 - p_2) = \lambda\bar{F}(\alpha_2) && 1 - p_3 = \bar{F}(\alpha_3)/\bar{F}(\alpha_2)\\
&\dots && \quad\dots
\end{aligned}$$

In general:

$$\lambda_i = \lambda\bar{F}(\alpha_{i-1})$$

Finally, this rate refers to flow arrivals, whereas thresholds α_i are expressed either in bytes or in packets, because they serve for demotion. It is simple to obtain the byte arrival rate by scaling λ_i (flows/sec) of the average traffic size produced by these flows on queue i. Remember that due to the demotion mechanisms, customers of each queue are truncated versions of the original flows, whose size ranges in $(0, \alpha_i - \alpha_{i-1})$. and that, subsequent priority queues observe a flow distribution truncated above α_{i-1}.
It is needed to compute $\mathbb{E}[L_i]$, the average length of customers served in queue i. It holds:

$$\mathbb{E}[L_i] = \underbrace{\int_{\alpha_{i-1}}^{\alpha_i} (x - \alpha_{i-1}) f(x) dx}_{\text{(i)}} + \underbrace{(\alpha_i - \alpha_{i-1}) \int_{\alpha_i}^{\infty} f(x) dx}_{\text{(ii)}} \tag{2.2}$$

(i) Traffic generated by flows with sizes in $[\alpha_{i-1}, \alpha_i)$

(ii) Traffic generated by flows larger than α_i

Define the truncated probability density function seen by queue i as:

$$f_i(x) = f(x) \,/\, \bar{F}(\alpha_{i-1})$$

It holds:

$$\lambda_i = \lambda\,\bar{F}(\alpha_{i-1}) \frac{\mathbb{E}[L_i]}{\bar{F}(\alpha_{i-1})} = \lambda\,\mathbb{E}[L_i] \tag{2.3}$$

Summarizing, it was possible to write down all arrival rates λ_i and all draining rate μ_i as function of the flow size distribution and the set of thresholds α_i. A summary of all quantities that have been defined is provided in Table ??. These parameters are sufficient to express the average sojourn time on a single link, so to characterize the performance of PIAS. The average sojourn time on queue i which comprises the average waiting and serving time is given by the well-known formula for M/M/1 queues:

$$\mathbb{E}[T_i] = \frac{1}{\mu_i - \lambda_i} \tag{2.4}$$

The M/M/1 model holds for every subsequent queue. This follows from Burke's theorem [6], that states that the outgoing process of an M/M/1 queue is a Poisson process. Thus, cascaded queues still observe a Markovian arrival process.

Variable	Description
Q_i	Priority queue i
N	Number of priorities i
X	Flow size
$F(x)$	Flow size c.d.f.
$f(x)$	Flow size p.d.f.
λ_i	Packet arrival rate at PQ i
μ_i	Drain rate of PQ i
ρ_i	Average load on Q_i
L_i	Customer size at PQ i
T_i	Waiting time at PQ i
α_i	Demotion threshold from Q_{i-1} to Q_i
p_i	Probability that a flow leaves after Q_i

Table 2.1: Variables of the model.

Finally, the total average sojourn time in the tandem of N queues is just a linear combination of $\mathbb{E}[T_i]$, where the coefficient that weight the individual sojourn times at any priority queue are the probabilities that a flow shall traverse the same PQ.

$$\mathcal{T} = \sum_{i=1}^{N} \theta_i \sum_{j=i}^{N} \mathbb{E}[T_j] \tag{2.5}$$

Thus, given the flow size distribution it is possible to easily compute the system performance according to the model yet presented.

Optimal thresholds

Plainly, it follows that the *optimal* thresholds α_i can be derived solving the non-linear minimization of $\mathcal{T}$. Notice that they are easily obtainable once all θ_i are known as

$$\alpha_i = F^{-1}(\sum_{j=1}^{i} \theta_j)$$

Hence, it is possible to solve in θ_i.

$$\begin{aligned} \min_{\{\theta_i\}} \quad & \mathcal{T} = \sum_{i=1}^{N} \theta_i \sum_{j=i}^{N} T_j \\ \text{subject to} \quad & \theta_i \geq 0 \qquad\qquad \forall i \in [1, N] \\ & \sum_{i=1}^{N} \theta_i = 1 \end{aligned} \tag{2.6}$$

2.3.2 Greedy threshold assignment

In order to verify the actual gain obtained with optimal demotion levels, it is practical to compare it against simpler greedy strategies for thresholds assignment. Another reason motivating such alternatives is that when the number of priority levels increases ($N{>}3$) the solution to the previous optimization is very time consuming, since the problem is non-linear and non-convex. In chapter 4 it is provided a breakdown which shows the CPU time spent for computing the optimal demotion thresholds through a meta-heuristic algorithm. Having a significant slow-down for the solution to converge deteriorates the reactiveness of the MLFQ scheduler in the case of a sudden change in the flow size distribution. As a consequence, the system would operate for long using thresholds mismatched with respect to the flow distribution.

Two intuitive approach are considered for fast threshold computation: Equal Split (ES) and Load Split (LS).

Equal Split (ES-N)

The *Equal-Split* method slices the flow size distribution uniformly in N percentiles. The resulting thresholds are the corresponding quantiles:

$$\alpha_i = F^{-1}\Big(\frac{i}{N}\Big), \qquad i = 1, .., N-1$$

It is easy to observe that this criterion may be largely sub-optimal depending on the shape of the flow size distribution. In the case of very sharped distributions short flows are demoted too early to lower priorities, while for heavy-tailed distributions, where high percentiles correspond to very long flows, latency sensitive flows remains mixed for long with elephant throughput-oriented streams.

Load Split

Load Split is a technique where the thresholds are chosen in a way to control the amount of traffic fed in every priority queue. The problem is easy to understand in the simple case of $N{=}2$ priority queues and a single threshold α. Let $G(y)$ be the amount of traffic generated by flows with size less than y:

$$G(y) = \int_0^y xf(x)dx$$

The traffic on high priority queue Q_0 is derived as the sum of bytes generated by flows whose size is smaller than the threshold α, and the bytes transmitted by flows larger than α:

$$\mathbb{E}[L_0] = \int_0^\alpha xf(x)dx + \int_\alpha^\infty \alpha f(x)dx = G(\alpha) + \alpha\bar{F}(\alpha)$$

while the traffic on low priority queue is

$$\mathbb{E}[L_1] = \int_\alpha^\infty (x-\alpha)f(x)dx = \mathbb{E}[X] - \mathbb{E}[L_0]$$

Then, the traffic on the high priority queue can be controlled solving the following *load-balance* equation:

$$\mathbb{E}[L_0] = \gamma\mathbb{E}[X] \tag{2.7}$$

A proper choice of γ apportions to the high priority queue a fraction of the average total traffic $\mathbb{E}[X] = \mathbb{E}[L_0] + \mathbb{E}[L_1]$. For example, if a perfect load balancing between the two queues is desired then it is set $\gamma = 1/2$. Extending these equations to the general case of any number of priority queues is rather simple. The traffic on the i-th queue has the same expression of Eq(2.2). It can be rewritten shortly with the notation just introduced:

$$\mathbb{E}[L_i] = G(\alpha_i) - G(\alpha_{i-1}) + \alpha_i \bar{F}(\alpha_i) - \alpha_{i-1}\bar{F}(\alpha_{i-1})$$

At this point, similarly to the simple case of two queues, it is enough to solve the following *load-balance* equations iteratively for all $i = 1, .., N-1$:

$$\begin{cases} \mathbb{E}[L_i] = \gamma_i \, \mathbb{E}[X], & i = 1, .., N-1 \\ \sum_i \gamma_i = 1, & \gamma_i \in [0{,}1] \end{cases} \tag{2.8}$$

In fact, also in this case it holds $\sum_{i=1}^{N} \mathbb{E}[L_i] = \mathbb{E}[X]$.

Proof. *In the expansion of* $\sum_{i=1}^{N} \mathbb{E}[L_i]$ *at each step new terms simplify old terms. Since* $\alpha_0 = 0$ *and* $\bar{F}(\alpha_N) = 0$*, it gives* $\int_0^\infty x f(x) dx = \mathbb{E}[X]$.

$$\begin{aligned} \sum_{i=1}^{N} \mathbb{E}[L_i] = \cdots &+ \cancel{G(\alpha_i)} - G(\alpha_{i-1}) + \cancel{\alpha_i \bar{F}(\alpha_i)} - \alpha_{i-1}\bar{F}(\alpha_{i-1}) + G(\alpha_{i+1}) - \cancel{G(\alpha_i)} \\ &+ \alpha_{i+1}\bar{F}(\alpha_{i+1}) - \cancel{\alpha_i \bar{F}(\alpha_i)} + \cdots = G(\alpha_N) - G(\alpha_0) + \cancelto{0}{\alpha_N \bar{F}(\alpha_N)} - \cancelto{0}{\alpha_0 \bar{F}(\alpha_0)} \\ &= \mathbb{E}[X]. \end{aligned}$$

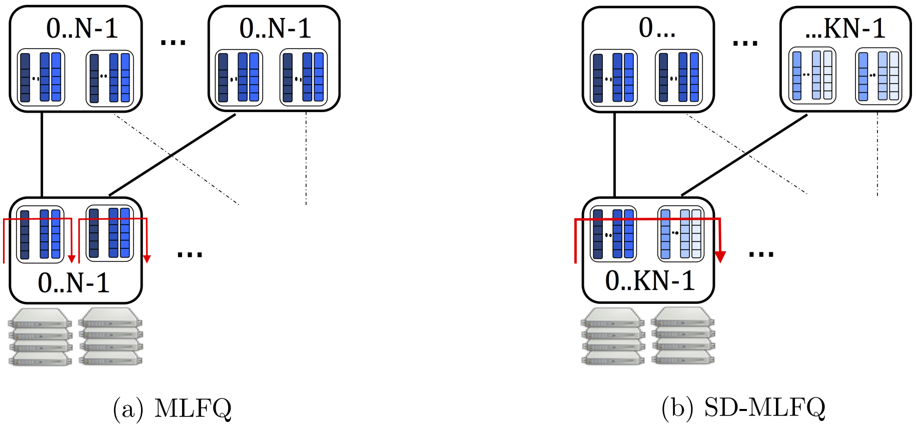
$\blacksquare$

2.4 A priority scheme with spatial-diversity

2.4.1 MLFQ with Spatial-Diversity

In this chapter have been presented two attempts to approximate the optimal LAS and SRPT schedulers leveraging multiple priority queues, that are already present in all datacenter equipments. However, one limitation of this approach remains the scarce availability of priority queues in commodity switches. A large number of priority levels is demanded to better approximate the reference scheduling disciplines, in which $N \to \infty$ (Sec. 2.1.2). Unfortunately, devices in modern DCNs are usually equipped with no more than 8 priority queues per port, whose majority are reserved for other purposes, like isolating different types of traffic. Indeed, several transport protocols may coexist in the same network, without necessarily being designed to be fair among them. Example of such transports are RDMA [9], DCTCP [2], standard TCP and UDP. For this reason, it is plausible to assume to have at most $N = 2$ priority queues as a realistic case.
Upon this understanding, the main proposal of this work is to evaluate the possibility to exploit the high degree of spatial diversity typically offered by DCN topologies to improve flow scheduling. Large DCN topologies are multilayer recursive Clos networks which offer a variety of equal-cost paths between racks. Precisely, in the simplest 2-layer Fat-Tree there are a number of paths proportional to the number of spines K (Sec. 1.1). The key

observation is that much like priority queues, different paths yet are a way for separating flows. In other words, the mechanism of priority demotion which shift the long flows across priority queues of a single interface, could be potentially applied at higher level as routing strategy over the switching fabric. Elephant flows would be physically segregated from mice flows, as they would be moved to different paths inside the switching fabric. Despite its simplicity, relevant works exploring this solution in the field of datacenter networks seem to be lacking. In more details, this contribution aims to integrate the spatial diversity property with the MLFQ scheduler. Consider a Leaf-Spine topology and focus on a single ToR switch. The interfaces from such a ToR towards all spines can be seen jointly as a unique big interface with K times more priority queues than a port alone. The idea is better clarified in Fig.2.4. The basic MLFQ (Fig.2.4a) focuses on the links individually and thanks to demotion moves flows, during their lifetime, across priority queues. Therefore, every interface handle the same priorities and flows are load balanced on different links independently from the prioritization mechanism. A standard technique for load balancing at flow-level is ECMP, which derives the next hop from the transport-layer tuple {IP ADDRESSES, PORTS, PROTOCOL ID}. Instead, this novel approach takes advantage of spatial diversity to extend the number of demotion levels beyond the limitation imposed by the PQs on a single interface. Interfaces from any ToR to the connected spines are virtually aggregated to offer a wider range of demotion levels. One new aspect of Spatially-Diverse MLFQ (SD-MLFQ) is that a demotion could imply to shift a flow from one path to another of equal-cost. In that light, the routing — thus load balancing — of flows on the Fat-Tree does depend on the priority. The spines can be seen as if they have a priority and a flow is moved both across queues and spines, effectively allowing a global resource exploitation for the demotion scheme. In other words, the routing policy over the fabric is not blind to the prioritization machinery, but smartly driven by the latter.



(a) MLFQ (b) SD-MLFQ

Figure 2.4: Extension for spatial diversity.

A clear benefit of this scheme is that finer granularity in priorities is achieved even with few queues per port, at the price of a very limited implementation complexity.

2.4.2 A queueing model extension for spatial-diversity

PIAS model with added dimension for spatial diversity. Proportional and equal splits for spatial diversiy

Chapter 3

Analysis of spatial-diversity with a numerical simulator

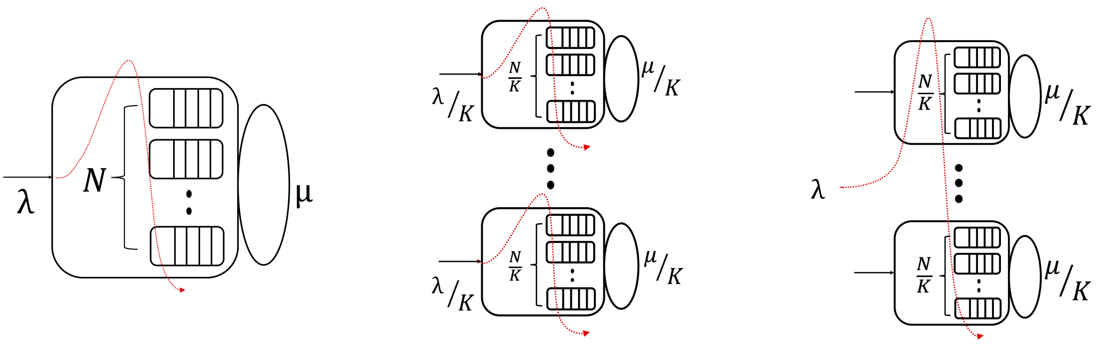
The previous chapter introduced the idea of exploiting spatial-diversity to provide more priority levels for flow classification. It was discussed why in principle such technique could yield flow completion time gains, especially when few priority queues are available. It was explained that adopting a spatial-diversity demotion has strong implication on the traffic load balancing, since flow routing becomes priority-dependent. A few questions arises soon: how to choose the thresholds to distribute the load on the topology? What are the relationships between the priority granularity in single interfaces and spatial diversity? Is the best choice to use all spines for incremental priorities when the topology is big? The goal of this chapter is to validate these intuitive hints with numerical results and to shed the light on the benefits and the restraints of the proposed algorithm. In particular, it will go through an exhaustive analysis of the system by delivering plenty of numerical experiments obtained with a custom flow simulator implemented in Python. This simulator does not capture any of the complex dynamics inherent to a real packet network, it does not have any protocol stack implemented neither at traffic sources modules nor in the switching modules. Rather, it is a job-oriented queueing simulator that does not care about packets but only runs flow arrivals and serve them in queuing modules. Its purpose is to provide a clean baseline numerical analysis not plagued by possible side effects due to network misconfiguration.

3.1 Spatial-diversity framework

3.1.1 Problem abstraction

To tackle spatial-diversity in the initial phase and get a first glance about its effectiveness, three queuing systems are compared. They are shown in Fig.3.1.

This is a general abstraction of the real system in a datacenter topology. At its heart, actually, a Leaf-and-Spine network with K spines can be described with a scheme of K parallel servers. These servers can be equivalently mapped to the *up-send* interfaces that connects a ToR switch to all the spines, as well as to the *down-send* interfaces that from



(a) Super-Server (b) Independent servers (c) Spatial-diversity

Figure 3.1: Three queuing system comparison.

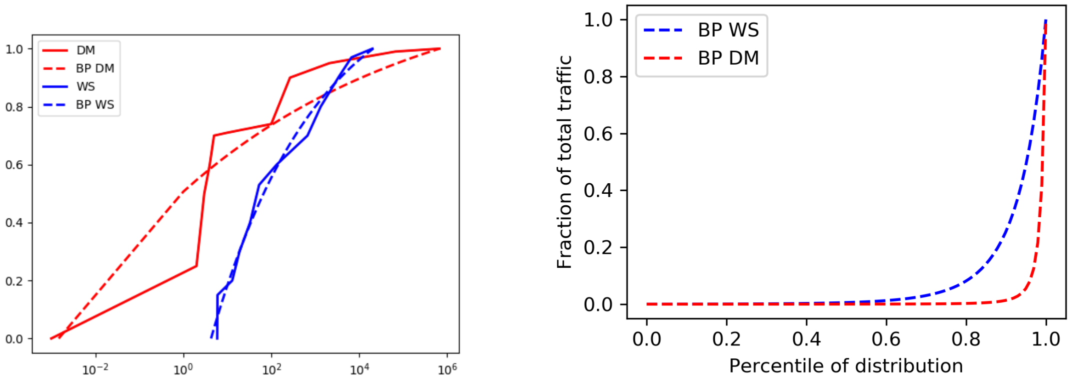
different spines forward the traffic to a single ToR. Ingress and egress ports that connect end hosts to the datacenter network are ignored at this stage. This is in contrast with the data center abstraction of pFabric as a giant switch presented previously (Figure 2.1), where the ingress and egress queues were the bottleneck links where to deploy best scheduling strategies. Also PIAS, optimizing its thresholds on the bottleneck link embraced a similar model. Somehow, differently from the state-of-the-art solutions that focus only on the bottleneck link assuming that with no oversubscription and ideal load balancing the network is able to sustain maximum throughput with negligible delays, the spatial-diversity approach shifts the attention on the queues inside the fabric itself. The integration with ingress and egress queues will be managed afterwards.

From now on and for the rest of this section, the interfaces abstracted in the simplified queuing model will be referred to as servers. The three systems are compared on the same conditions of total arrival rate λ, processing power μ and priority levels N, but with different partitioning of the same priority levels on a number K of servers and, more importantly, with different demotion trajectories. Specifically, all systems use priority demotion first introduced by PIAS (§2.2.2). Red arrows indicate the trajectory across all available priority queues followed by the longest flow that experience the maximum number of demotions.

The first case (Fig. 3.1a) is to have a single high-capacity server that handles all priorities. This of course is the absolute best case where resources are fully concentrated, that doesn't exist in real systems and that provides the smallest delay. It would be equivalent to realize an entire data center interconnection network with a single device of astonishing bandwidth. The second case (Fig. 3.1b) is the legacy way to handle priority demotion, where all servers are treated independently in parallel. Flows are evenly load balanced on the available servers and moved across priorities of the same server during their lifetime. In this case the number of demotion levels are limited to $\frac{N}{K}$. Finally, the third case (Fig. 3.1c) is the novel object under investigation, where all the flows are initially sent to the same server, then demoted on the N globally available priority queues. In this case subsequent servers are configured to handle lower and lower priorities, thus part of the flows are re-routed as a consequence of the spatial-diversity. In the following, also the servers and the links may be labeled as "high priority" or "low priority" for brevity, meaning that they handle high priority traffic or low priority traffic, respectively.

3.1.2 Workloads

The performances of the three systems are compared using two empirical flow size distributions that have been derived from production data centers (Figure 3.2). Flow size distribution is shortly termed *workload*. The first workload has been estimated instrumenting thousands of servers in a datacenter hosting a Web search [2] application, while the other refers to data mining tasks [8]. As expected, these distributions have a mix of short and long flows and both present the high-variability property typical of data center traffic (§1.2.2). Figure 3.2a shows with solid lines the cumulative density function of the two empirical workloads, along with two dashed Bounded Pareto distributions, whose parameters have been fitted to the corresponding empirical points. The bounded Pareto distribution is a truncated version of the Pareto distribution over the finite support $[u, t]$ and it is well-suited to model heavy-tail characteristics. It has three parameter: the lower extreme of its support u, the upper extreme t and the shape parameter α that controls the weight of its tail. The analytical expression of its cdf $F(x)$ in the interval $[u, t]$ is:



(a) CDF

(b) MWF

Figure 3.2: Workload properties

$$F(x) = \frac{1 - \left(\frac{u}{x}\right)^{\alpha}}{1 - \left(\frac{u}{t}\right)^{\alpha}}, \qquad 0 \leq \alpha \leq 2 \tag{3.1}$$

This distribution has been chosen to be used in the analysis due to some graceful properties. First of all, it is relatively easy to control its variability by a proper tuning of its parameter α. Values of α close to 2 accentuate the heavy-tail property, while smaller values of α tend to regularize a bit its behavior. Second, being definite on a limited support it can be adapted to any minimum and maximum flow size in the datacenter. Last, its mean and its variance — which depend on α — are finite, thus the problem of finding the shape parameter for any fixed first and second moment can be smoothly treated numerically. Specifically, for the bounded Pareto, the mean and the variance have the following expression:

$$\mathbb{E}[X] = \frac{\alpha}{(1-\alpha)(t^\alpha - u^\alpha)}(u^\alpha t - t^\alpha u)$$

$$\sigma_X^2 = \frac{\alpha}{(2-\alpha)(t^\alpha - u^\alpha)}(u^\alpha t^2 - t^\alpha u^2)$$

The best fit to the empirical distributions has been obtained with a simple Maximum Likelihood Estimator (MLE). The resulting parameters are reported for both the workloads. Let X be the flow size random variable as usual, and write in short notation $BP(u, t, \alpha)$ the bounded Pareto.

$$\begin{aligned} X_{WS} &\sim BP(3,\ 29000,\ 0.125) \\ X_{DM} &\sim BP(0.1,\ 100000,\ 0.26) \end{aligned}$$

The measurement unit for the extremes of the support in this case is kilobytes. The fitting error is higher for the data mining workload than for the web search. For low percentiles this is difficult to avoid because a very crude sampling is provided. In fact, on a total of 11 empirical points, 4 of them are for values above the 90th percentile. Instead, for high percentiles a better fitting likely could be obtained by weighting more the tail of the distribution. However, since the simulations will be performed with the empirical traces, this point is neglected.

The rightmost plot (Fig. 3.2b) completes the picture by showing the Mass-Weighted Function $M_w(x)$ [7]. This can be seen as the probability that a byte picked at random belongs to a flow below a given percentile and it is used to characterize the variability of a distribution. Its name comes from its definition, where job sizes are weighted by their probability mass:

$$M_w(x) = \frac{\int_0^x xf(x)dx}{\mathbb{E}[X]}$$

It holds:

$$\int_0^x xf(x)dx \leq \mathbb{E}[X]$$

In other words, it is just the average normalized traffic injected by flows shorter than x. The figure has on the abscissa the percentile rather than the corresponding job size, to allow the comparison between workloads with different supports on the same axis. If y is a given percentile, it is evaluated $M_w(F^{-1}(y))$.

In summary, both distributions exhibit high variability. In the web search case the largest 4% of flows carry half of the total traffic, the data mining is even more skewed: 70% of the flows are less than 8 packets only, but almost the entire load is sustained by a ridiculous percentage of flows of about 100MB of size. This suggests that the more challenging distribution to schedule is the web search, consequently it is the one that will deserve most of the attention. In fact, recall that an ideal flow-agnostic LAS scheduler guarantees lower and lower delays as the variability of the distribution increases, both on average and at high percentiles (§2.1.2). The theory is confirmed pretty straightforwardly by the simulation results presented next. Moreover, the web search distribution is also a lot easier to simulate, since the very long tails of the data mining workload require protracted time-consuming simulations before being precisely reproduced. Long flows occur sporadically, however they give the main contribution to generate a desired load on the system.

3.1.3 Optimal traffic load balancing

In previous discussions, it was already realized that the spatial diversity framework introduces strong implications on how the load is distributed on the switching fabric, in particular on the bisection bandwidth. To clarify the concept let's come back to the abstract model considered so far and let's analyze the simpler example of spatial diversity, where only two M/M/1 servers are deployed in parallel, each of them with only a single priority queue. With this setup all flow demotions correspond to shifting a flow from one server to the other. Therefore, the assignment of the demotion threshold can be equivalently seen as a load balance problem. If the threshold is too small, flows are early rerouted on the low priority path and the capacity of the high priority link is essentially wasted. For the real data center implementation this would mean a 50% throughput reduction of the switching fabric. On the other end, the traditional *perfect* load balance which uniformly splits the traffic on available servers may correspond to a bad demotion threshold for the FCT minimization. Depending on the shape of the flow size distribution, latency constrained traffic would remain unnecessarily mixed with non-demanding latency flows. Certainly, the same issues arise the same augmenting the dimensionality of the topology and the priority queues. A careful tuning of load balance is a new trade-off that was nonexistent in the legacy MLFQ framework, where traffic was demoted to different priority queues but still in the same interface (i.e. link). If that is the case, a threshold setting that gives an unbalanced load allocation in the available priority queues affects only the delays but not the maximum throughput that the network can sustain. The introduction of spatial diversity adds a level of complexity to the system.

Two related problems need to be solved: finding the set of thresholds that triggers a new flow route from one server to another and finding for each server a set of thresholds that mark the demotion boundaries among PQs. Let's denote them as *load-balance thresholds* and *sub-thresholds* respectively. In section 2.4.2 was presented an extension to the PIAS model that embeds spatial-diversity. This formulation would in principle address both problems. In fact, the optimal solution would never include load balance thresholds that overload some servers, because that would overkill the delays which the problem tries to minimize. At the same time, a unique formulation would choose a set of sub-thresholds targeted on the optimal load balance thresholds, providing a joint solution. Despite being a clean analytical formulation, its complexity seems prohibitive. The basic model without spatial diversity yet was non-convex and presented products and ratios of unknowns (Sec. 2.3.1). Nonetheless, it is still tractable since the number of variables is typically bounded to the PQs available in commodity switches. Instead, in the complete model the number of variables scales with the product $K \times N$, where K is usually a large number for large-scale data centers. An attempt for the web search workload and $K \times N = 16$ was pursued with two well-known meta-heuristics solvers, specifically PSO [10, 14] and Basin-Hoppin [15]. The time scale at which we could get a solution was at least days. Even finding an optimal solution once, it is not feasible to apply it to any real datacenter scenario where likely this solution must be computed repeatedly as statistics change. To the purpose of investigating the spatial diversity framework, it has been preferred to handle the two problems individually. The approach that has been carried out is decoupled in two sequential steps.

1. **Optimize load-balance thresholds.** First it is solved an optimization problem with the goal of finding the best load partitioning among K servers. Servers are always assumed to have a single priority (N=1), because for the time being subthresholds are ignored. Subsequent servers are assigned increasing priorities, so that the number of demotions equals the number of servers and there are no duplicated priorities. At the end of this phase a set of K-1 load balance thresholds is delivered.

2. **Greedy subthresholds**. The load-balance thresholds are provided as input of the second phase. They split the support of the flow size in K disjoint intervals covering the whole support. The i-th interval contains the sizes of those flows that end their service in server of priority i. On each interval is computed a set of sub-thresholds with a greedy algorithm like ES-N or LS-N (Sec.2.3.2).

For the first phase turns out to be useful the stochastic queuing model of the legacy MLFQ system (Sec.2.3.1-Fig.2.2). After all, since each server is equipped with a single priority per port, the load balance problem can be abstracted with exactly the same model. The only exception is that the priority queues are physically distributed to different servers, instead of being part of the same interface. They are independent on each other and the strict priority scheduler is no more involved, as they work in parallel without coordination (Fig.3.1c). Practically, the only modification is to the queues capacities μ_i. Remember that in the aforementioned model the strict priority order was described by attenuating the serving rates $\mu_i = \mu \prod_i (1 - \rho_i)$ for increasing i. Because of servers work in parallel without scheduling, this expression is not needed anymore and the draining rates just coincide:

$$\mu_i = \mu, \qquad i = 1, .., K$$

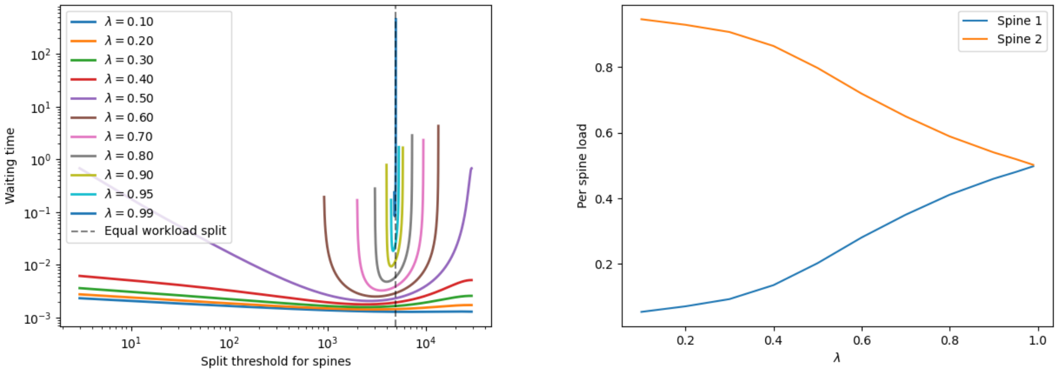
For the sake of simplicity, the K parallel M/M/1 servers were implemented with service rate $\mu = \mathbb{E}[X]$. In this way the average load fed in the system

$$\rho = \frac{\lambda}{\mu}\,\mathbb{E}[X]$$

is given only by the flow arrival intensity λ.

Load balance on 2 parallel servers

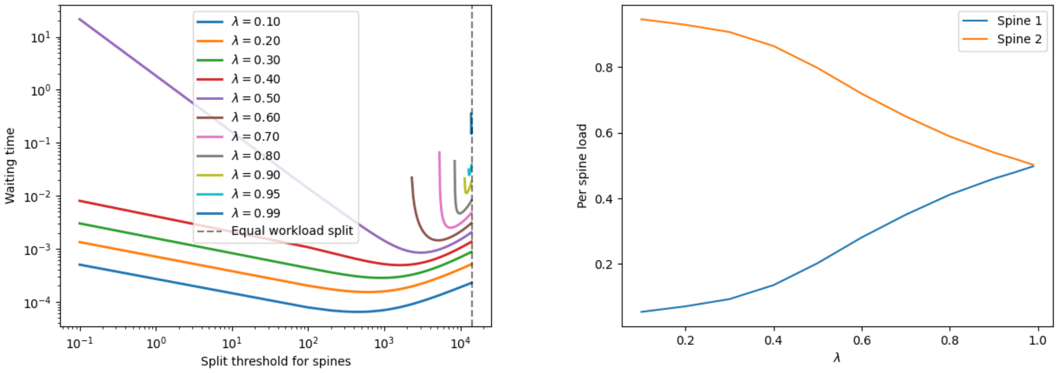
In the basic case of K=2 parallel servers, it is possible to plot the average sojourn time when varying the load balance threshold. Figures 3.3a and 3.4a show the aspect of the cost functions just described for the web search and the data mining workloads, respectively. All the axes are in log-scale and each curve represents a different normalized traffic



(a) Optimal load balance threshold

(b) Load distribution per-server

Figure 3.3: Web search workload. Simple case of K=2.



(a) Optimal load balance threshold

(b) Load distribution per-server

Figure 3.4: Data mining workload. Simple case of K=2.

λ. The vertical lines correspond the split that gives perfect load balance, apportioning half of the traffic on the high priority server and half on the low priority one. This split may be interchangeably referred to as *perfect split* or *proportionate split*. One phenomena is visible for both workloads, confirming previous intuitions. Imagine to connect the dots corresponding to the minima of the cost functions. The optimal load balance threshold does not coincide, broadly speaking, with the proportionate split threshold. Depending on the traffic level at which the system is operated and the workload, the optimal threshold triggers an earlier or later demotion with respect to the perfect split case. Equivalently, the jobs are distributed unfairly between the two servers (Figures 3.3b-3.4b). For data mining, the imaginary line would be always in the leftmost side with respect to the proportionate split. That is, apart from loads close to saturation, the high priority server is kept as jobless as possible and the majority of work is sent to the low priority server. This result is coherent with the shape of the data mining workload,

that presented huge variability. Plenty of short flows carrying few bytes are kept on a separated link from medium sized flows and a couple of long flows. This is not a surprise, since the problem formulation weighted more the average sojourn times of short flows. Slightly different trend is observed for web search. Moving from low to high values of λ, the optimal threshold is initially greater than the perfect split axis, then switches to its left and finally the two coincide. TODO Why??? For both cases, the objective function becomes much more extremely curled and steep around the perfect split axis when the load approaches the saturation value 1. As already remarked, this is inevitable in order to fully exploit all the available capacity offered by the two parallel servers. At so high load any other traffic split would overload one of the two links and strongly deteriorate the average completion time of the flows going through it. Consistently, the cost function grows very rapidly in the neighborhood of the perfect split threshold value.
Finally, the two workloads have different sensitivities to the load balance threshold optimization. In particular, the web search achieves appreciable FCT gains starting from medium loads only ($\lambda > 0.5$) with (TODO provide some numbers), whereas at low loads there is no practical difference with the proportionate split. On the contrary, the data mining distribution gives theoretically a (TODO percentage) lower waiting time even for λ=0.1. This again reflects the extreme variability of the workload, that is comprised by a disproportionate number of short flows. The perfect split would keep a lot of mice flows mixed with low priority flows of bigger size for protracted service time. Instead, the optimal threshold reroute few flows but lot of traffic on the low priority server. In fact, for λ=0.1 the absolute value of the optimal threshold is an order of magnitude lower than the proportionate split threshold, however only a small fraction of flows fits into this gap.

Summarizing, the above discussion confirms that the web search traffic is harder to cope with. This is the reason why in the following some results are shown exclusively for this workload.

Validity of the model

Analysis of the model scalability

Finding sub-thresholds

3.2 The service discipline: FIFO or PS?

3.2.1 Quantitative comparison

FCT comparison: FIFO vs PS

3.2.2 Flow synchronization

Up to which point is spatial diversity meaningful? Flow synchronization problem. Burst arrivals.

Chapter 4

Evaluation in a Datacenter Network

4.1 title

4.1.1 title

4.1.2 title

4.2 title

4.2.1 title

4.2.2 title

Bibliography

[1] Samuli Aalto, Urtzi Ayesta, and Rhonda Righter. On the gittins index in the m/g/1 queue. *Queueing Systems*, 63(1-4):437–458, 2009.

[2] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 63–74, New York, NY, USA, 2010. Association for Computing Machinery.

[3] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick Mckeown, Balaji Prabhakar, and Scott Shenker. pfabric. *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM - SIGCOMM 13*, 2013.

[4] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. Pias: Practical information-agnostic flow scheduling for commodity data centers. *IEEE/ACM Transactions on Networking*, 25(4):1954–1967, 2017.

[5] Wei Bai, Li Chen, Kai Chen, and Haitao Wu. Enabling ecn in multi-service multi-queue data centers. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, page 537–549, USA, 2016. USENIX Association.

[6] Paul J. Burke. The output of a queuing system. *Operations Research*, 4(6):699–704, 1956.

[7] Mark E. Crovella. Performance evaluation with heavy tailed distributions. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–10, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[8] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, page 51–62, New York, NY, USA, 2009. Association for Computing Machinery.

[9] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 202–215, New York, NY, USA, 2016. Association for Computing Machinery.

[10] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[11] Idris A. Rai, Guillaume Urvoy-Keller, and Ernst W. Biersack. Analysis of las scheduling for job size distributions with high variance. *Proceedings of the 2003 ACM*

SIGMETRICS international conference on Measurement and modeling of computer systems - SIGMETRICS 03, 2003.

[12] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social networks (datacenter) network. *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication - SIGCOMM 15*, 2015.

[13] Linus Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

[14] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pages 69–73, 1998.

[15] David J Wales and Jonathan PK Doye. Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116, 1997.