Scheduling
@SIGCOMM'16

Wednesday 8:30am

Isaac Keslassy

Technion / VMware

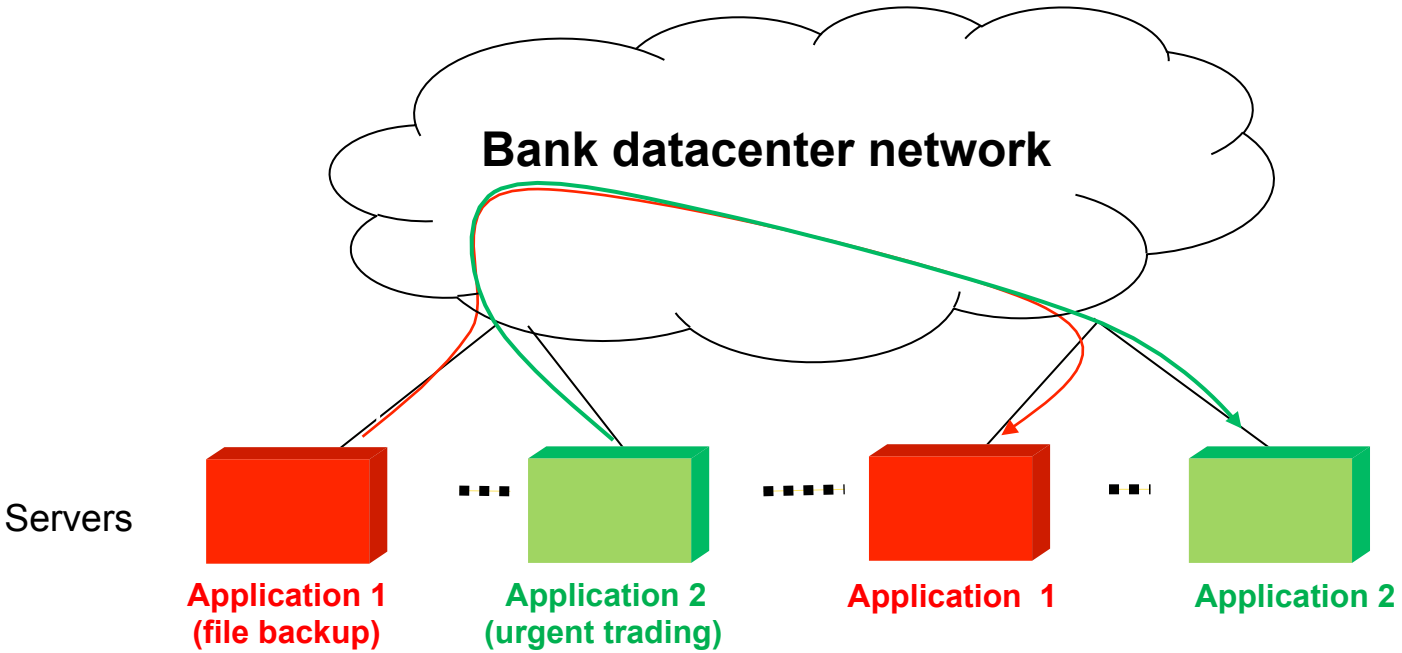
vmware®

What this talk is about

- What is scheduling?
- Research challenges?

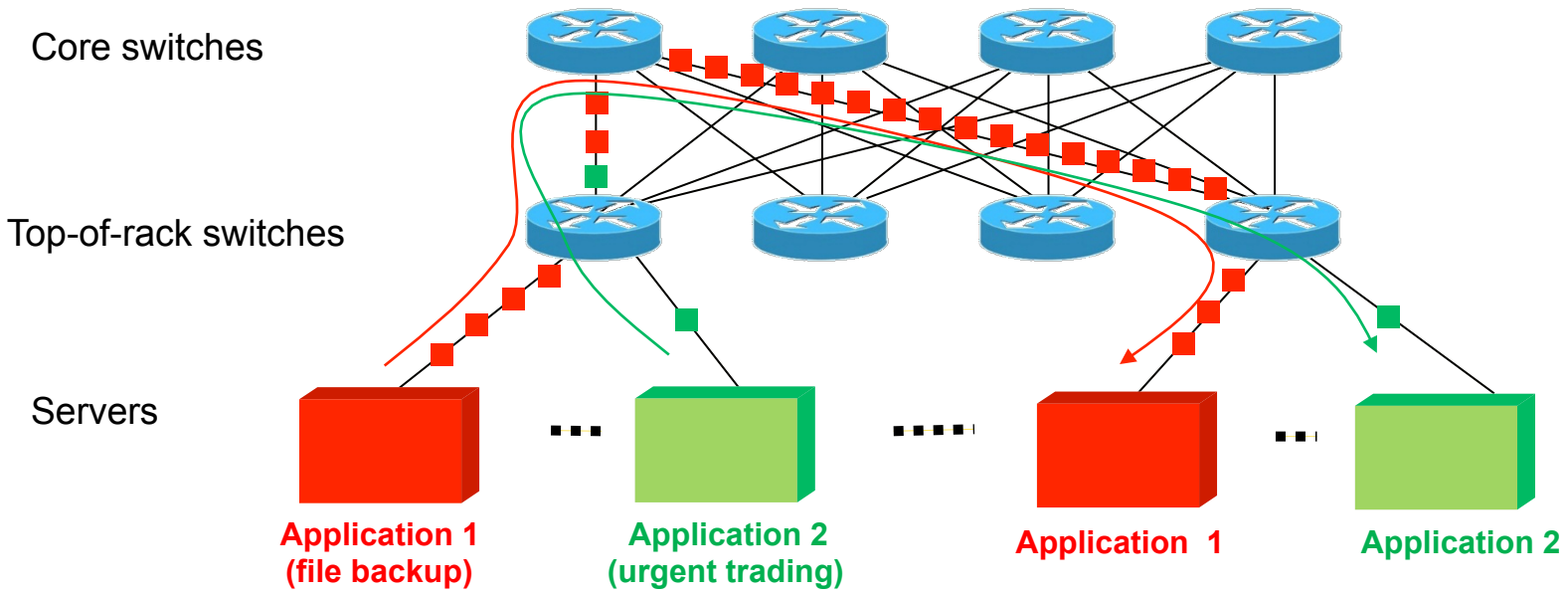
Datacenter networks

- Datacenters increasingly form the backbone of our day-to-day lives: banks, commerce, science, etc.
- They host many applications with **heterogeneous** bandwidth and latency requirements



What is scheduling?

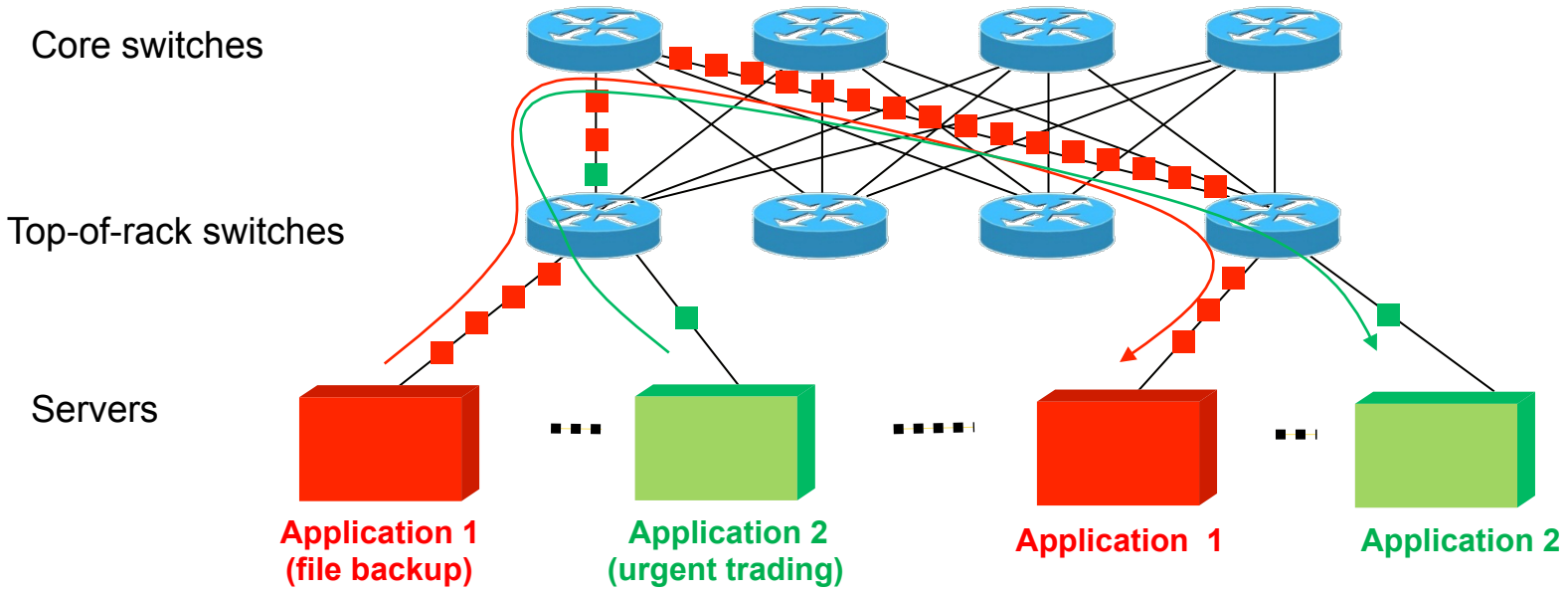
- Scheduling is about determining what packets to release and when
- It is done both in the **servers** (establishing packet rate) and in the **switches** (fairness between application flows)



Scheduling

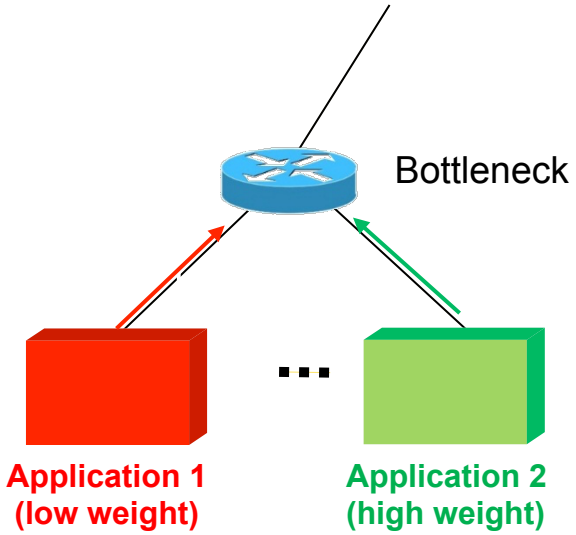
- **Challenge:** applications have heterogeneous requirements: low latency, high bandwidth, etc.

- **Our goal: world peace**
- Scheduling is about **fairness** in datacenters: exploring the design space to fit the requirements of each application – and avoiding the dreaded phone call of unhappy customers

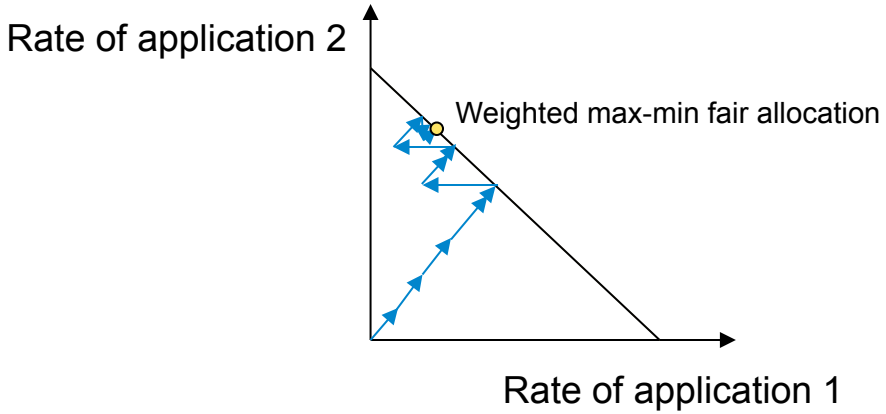


Challenge (1): Fast network fairness

- We want to converge to a network-wide weighted max-min fair allocation

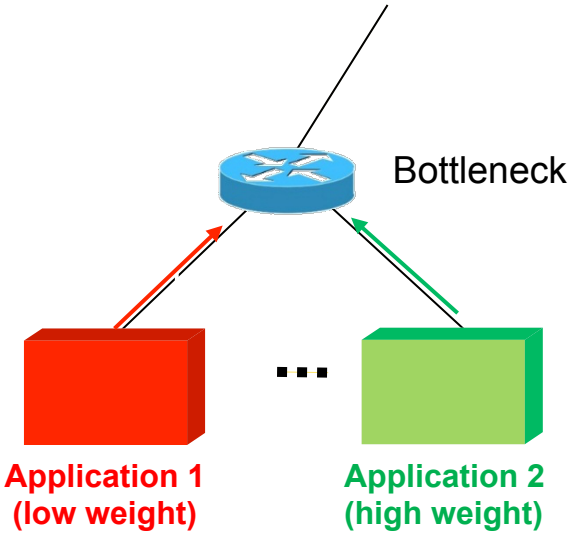


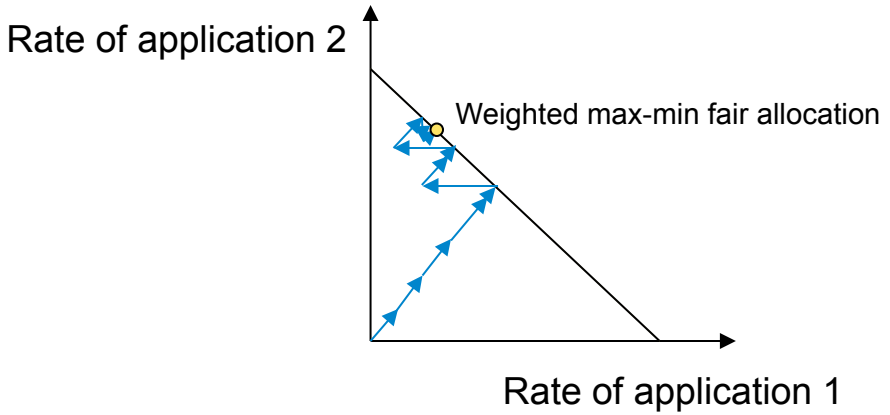
- Most approaches can be thought of as using gradient descent



Challenge (1): Fast network fairness

- Problem: **long convergence time** (~100 RTTs)
- Meanwhile:
  - Network bandwidth is not fully utilized
  - Most flows may only last a few RTTs at high link speeds





NUMFabric

Core idea: split problem into two layers:

(a) Bottom layer makes sure to always have a network-wide weighted max-min fair allocation

(b) Top layer looks for the correct weighted max-min allocation

NUMFabric: Fast and Flexible Bandwidth Allocation in Datacenters

Kanthi Nagaraj*, Dinesh Bharadia†, Hongzi Mao†, Sandeep Chinchali*, Mohammad Alizadeh†, Sachin Katti*
*Stanford University, †MIT CSAIL

ABSTRACT

We present NUMFabric, a novel transport design that provides flexible and fast bandwidth allocation control. NUMFabric is flexible: it enables operators to specify how bandwidth is allocated amongst contending flows to optimize for different service-level objectives such as weighted fairness, minimizing flow completion times, multipath resource pooling, prioritized bandwidth functions, etc. NUMFabric is also very fast: it converges to the specified allocation 2.3× faster than prior schemes. Underlying NUMFabric is a novel distributed algorithm for solving network utility maximization problems that exploits weighted fair queueing packet scheduling in the network to converge quickly. We evaluate NUMFabric using realistic datacenter topologies and highly dynamic workloads and show that it is able to provide flexibility and fast convergence in such stressful environments.

CCS Concepts

•Networks → Transport protocols; Data center networks;

Keywords

Resource allocation; Convergence; Network utility maximization; Weighted max-min; Packet scheduling

1. INTRODUCTION

Bandwidth allocation in networks has historically been at the mercy of TCP. TCP's model of allocation assumes that bandwidth should be shared equally among contending flows. However, for an increasing number of networks such as datacenters and private WANs, such an allocation is not a good fit and is often adversarial to the operator's intent. Consequently, there has been a flurry of recent work on transport designs, especially for datacenters, that target different bandwidth allocation objectives. Many aim to minimize per-packet latency [2, 54] or flow completion time [23, 3], while others target multi-tenant bandwidth allocation [58, 7, 55, 26], while still others focus on sophisticated objectives like resource pooling [56], policy-based bandwidth allocation [35], or coflow scheduling [15, 17, 14]. In effect, each design supports one point in the bandwidth allocation policy design space, but operators ideally want a transport that can be tuned for different points in the design space depending on workload requirements.

In this paper, we present NUMFabric, a novel transport fabric that enables operators to flexibly specify bandwidth allocation policies, and then achieves these policies in the network using simple, distributed mechanisms at the switches and end-hosts. NUMFabric's design is based on the classic Network Utility Maximization (NUM) [33] framework which allows per-flow resource allocation preferences to be expressed using *utility functions*. Utility functions encode the benefit derived by a flow for different bandwidth allocations, and can be chosen by the operator to achieve different bandwidth and fairness objectives. In §2, we show how an operator can translate high level policies such as varying notions of fairness, minimizing flow completion times, resource pooling a la MPTCP [56] and bandwidth functions [35] into utility functions at end-hosts. NUMFabric then realizes the bandwidth allocation that maximizes the sum of the utility functions in a completely distributed fashion.

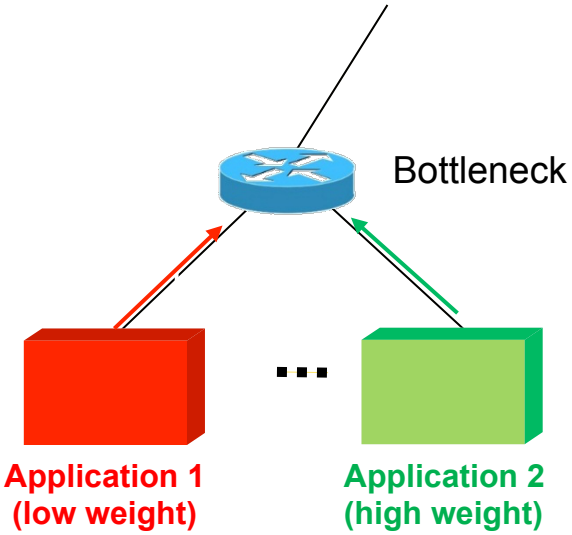
Network utility maximization of course is not new. There is a long line of work [61] on designing distributed algorithms based on gradient descent for NUM (§3). However, these algorithms are slow to converge to the optimal rate. For datacenter workloads, where a majority of the flows may last only a few RTTs due to the high link speeds, the convergence time of these algorithms is often much larger than the lifetime of the majority of flows, and hence no guarantees on resource allocation can be made. Moreover, gradient-descent algorithms are difficult to tune since they have a "step-size" parameter that needs to be tuned for each workload and resource allocation objective. In practice, given the scale of datacenters and the variety of objectives, getting the tuning right is a formidable task.

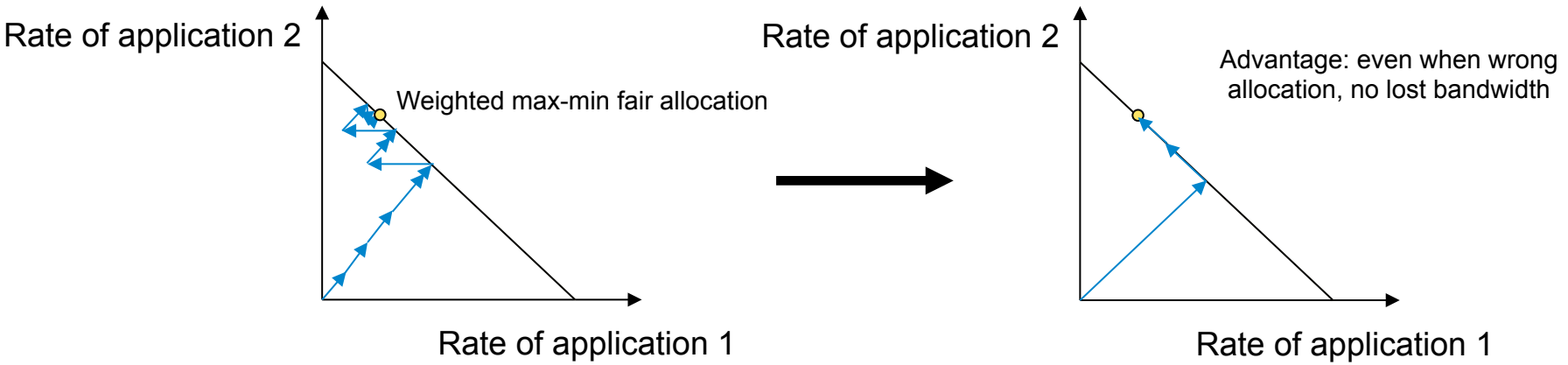
Our main technical contribution is a transport design for solving the NUM problem that converges significantly faster than prior work and is much more robust. The key insight

SIGCOMM '16, August 22 - 26, 2016, Florianopolis , Brazil

ISBN 978-1-4503-4193-6/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2934872.2934890

NUMFabric


Bottleneck
Application 1
(low weight)
Application 2
(high weight)


Rate of application 2
Weighted max-min fair allocation
Rate of application 1
Rate of application 2
Advantage: even when wrong
allocation, no lost bandwidth
Rate of application 1

Challenge (2): Deadlines

- Near-real-time machine-learning applications (e.g. web search, navigation, maybe self-driving cars?)
- How can these flows with deadlines coexist with flows without deadlines?
- **Challenge:** Strict priority to flows with deadlines can result in starvation of other flows

Karuna

Core idea: trade off:
(a) max deadline meet rate for deadline flows, and
(b) min flow completion time (FCT) for non-deadline flows.

Scheduling Mix-flows in Commodity Datacenters with Karuna

Li Chen, Kai Chen, Wei Bai, Mohammad Alizadeh(MIT)
SING Group, CSE Department
Hong Kong University of Science and Technology

ABSTRACT

Cloud applications generate a mix of flows with and without deadlines. Scheduling such *mix-flows* is a key challenge; our experiments show that trivially combining existing schemes for deadline/non-deadline flows is problematic. For example, prioritizing deadline flows hurts flow completion time (FCT) for non-deadline flows, with minor improvement for deadline miss rate.

We present Karuna, a first systematic solution for scheduling mix-flows. Our key insight is that deadline flows should meet their deadlines while *minimally* impacting the FCT of non-deadline flows. To achieve this goal, we design a novel Minimal-impact Congestion control Protocol (MCP) that handles deadline flows with as little bandwidth as possible. For non-deadline flows, we extend an existing FCT minimization scheme to schedule flows with known and unknown sizes. Karuna requires no switch modifications and is backward compatible with legacy TCP/IP stacks. Our testbed experiments and simulations show that Karuna effectively schedules mix-flows, for example, reducing the 95th percentile FCT of non-deadline flows by up to 47.78% at high load compared to pFabric, while maintaining low (<5.8%) deadline miss rate.

CCS Concepts

•**Networks** → **Transport protocols;**

Keywords

Datacenter network, Deadline, Flow scheduling

1. INTRODUCTION

User-facing datacenter applications (web search, social networks, retail, recommendation systems, etc.) often have stringent latency requirements, and generate a diverse mix of short and long flows with strict deadlines [3, 22, 38, 39]. Flows that fail to finish within their deadlines are excluded from the results, hurting user experience, wasting network bandwidth, and incurring provider revenue loss [39]. Yet, today's datacenter transport protocols such as TCP, given their Internet origins, are oblivious to flow deadlines and perform poorly. For example, it has been shown that a substantial fraction (from 7% to over 25%) of flow deadlines are not met using TCP in a study of multiple production DCNs [39].

Meanwhile, flows of other applications have different performance requirements; for example, parallel computing applications, VM migration, and data backups impose no specific deadline on flows but generally desire shorter completion time. Consequently, a key question is: how to schedule such a mix of flows with and without deadlines? To handle the mixture, a good scheduling solution should *simultaneously*:

- Maximize deadline meet rate for deadline flows.
- Minimize average flow completion time (FCT) for non-deadline flows.
- Be practical and readily-deployable with commodity hardware in today's DCNs.

While there are many recent DCN flow scheduling solutions [3–5, 22, 30, 38, 39], they largely ignore the mix-flow scheduling problem and cannot meet all of the above goals. For example, PDQ [22] and PIAS [5] do not consider the mix-flow scenario, while pFabric [4] simply prioritizes deadline flows over non-deadline traffic, which is problematic (§2). Furthermore, many of these solutions [4, 22, 30, 39] require non-trivial switch modifications or complex arbitration control planes, making them hard to deploy in practice.

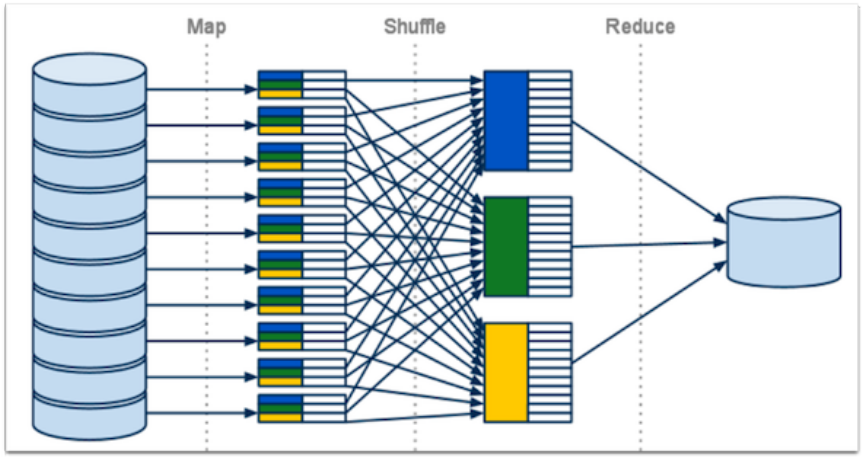
We observe that the main reason prior solutions such as pFabric [4], or more generally, EDF-based (Earliest Deadline First) scheduling schemes, suffer in the mix-flow scenario is that they complete deadline flows *too* aggressively, thus hurting non-deadline flows. For example, since pFabric strictly prioritizes deadline flows, they aggressively take *all* available bandwidth and (unnecessarily) complete far before their deadlines, at the expense of increasing FCT for non-deadline flows. The impact on non-deadline flows worsens with more deadline traffic, but is severe even when a small fraction (e.g., 5%) of all traffic has deadlines (see §2.2).

SIGCOMM '16, August 22–26, 2016, Florianopolis, Brazil

ISBN 978-1-4503-4193-6/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2934872.2934888

Challenge (3): Coflows

- An application consists of more than one flow
- **Coflow**: group of flows of an application in a communication stage (e.g. MapReduce shuffle stage)
  - The communication stage finishes when **all** the flows complete
- There are many results on scheduling coflows
- **Challenge**: how to identify coflows?



CODA

Key ideas:

(a) Clustering-based method to identify coflows

(b) Error-tolerant coflow scheduler

CODA: Toward Automatically Identifying and Scheduling COflows in the DArk

Hong Zhang[1] Li Chen[1] Bairen Yi[1] Kai Chen[1] Mosharaf Chowdhury[2] Yanhui Geng[3]
[1]SING Group, Hong Kong University of Science and Technology
[2]University of Michigan [3]Huawei
{hzhangan,lchenad,biy,kaichen}@cse.ust.hk, mosharaf@umich.edu, geng.yanhui@huawei.com

ABSTRACT

Leveraging application-level requirements using coflows has recently been shown to improve application-level communication performance in data-parallel clusters. However, existing coflow-based solutions rely on modifying applications to extract coflows, making them inapplicable to many practical scenarios.

In this paper, we present CODA, a first attempt at automatically identifying and scheduling coflows *without* any application modifications. We employ an incremental clustering algorithm to perform fast, *application-transparent coflow identification* and complement it by proposing an *error-tolerant coflow scheduler* to mitigate occasional identification errors. Testbed experiments and large-scale simulations with production workloads show that CODA can identify coflows with over 90% accuracy, and its scheduler is robust to inaccuracies, enabling communication stages to complete 2.4× (5.1×) faster on average (95-th percentile) compared to per-flow mechanisms. Overall, CODA's performance is comparable to that of solutions requiring application modifications.

CCS Concepts

•**Networks** → **Cloud computing;**

Keywords

Coflow; data-intensive applications; datacenter networks

1 Introduction

A growing body of recent work [21, 23, 24, 30, 38, 68] has demonstrated that leveraging application-level information using coflows [22] can significantly improve the communication performance of distributed data-parallel applications.[1] Unlike the traditional flow abstraction, a coflow captures a collection of flows between two groups of machines in successive computation stages, where the communication stage finishes only after *all* the flows have completed [23, 26]. A typical example of coflow is the shuffle between the mappers and the reducers in MapReduce [28]. By taking a holistic, application-level view, coflows avoid stragglers and yield benefits in terms of scheduling [21, 23, 24, 30], routing [68], and placement [38].

However, extracting these benefits in practice hinges on one major assumption: *all* distributed data-parallel applications in a shared cluster – be it a platform-as-a-service (PaaS) environment or a shared private cluster – have been modified to *correctly* use the *same* coflow API.

Unfortunately, enforcing this requirement is infeasible in many cases. As a first-hand exercise, we have attempted to update Apache Hadoop 2.7 [58] and Apache Spark 1.6 [65] to use Aalo's coflow API [24] and faced multiple roadblocks in three broad categories (§5): the need for intrusive refactoring, mismatch between blocking and non-blocking I/O APIs, and involvement of third-party communication libraries.

Given that users on a shared cluster run a wide variety of data analytics tools for SQL queries [3, 4, 7, 15, 41, 63], log analysis [2, 28, 65], machine learning [33, 43, 48], graph processing [34, 44, 46], approximation queries [7, 12], stream processing [9, 11, 13, 50, 66], or interactive analytics [7, 65], updating one application at a time is impractical. To make things worse, most coflow-based solutions propose their own API [23, 24, 30]. Porting applications back and forth between environments and keeping them up-to-date with evolving libraries is error-prone and infeasible [54, 57].

Therefore, we ponder a fundamental question: *can we automatically identify and schedule coflows without manually updating any data-parallel applications?* It translates to three key design goals:

- **Application-Transparent Coflow Identification** We must be able to identify coflows without modifying applications.

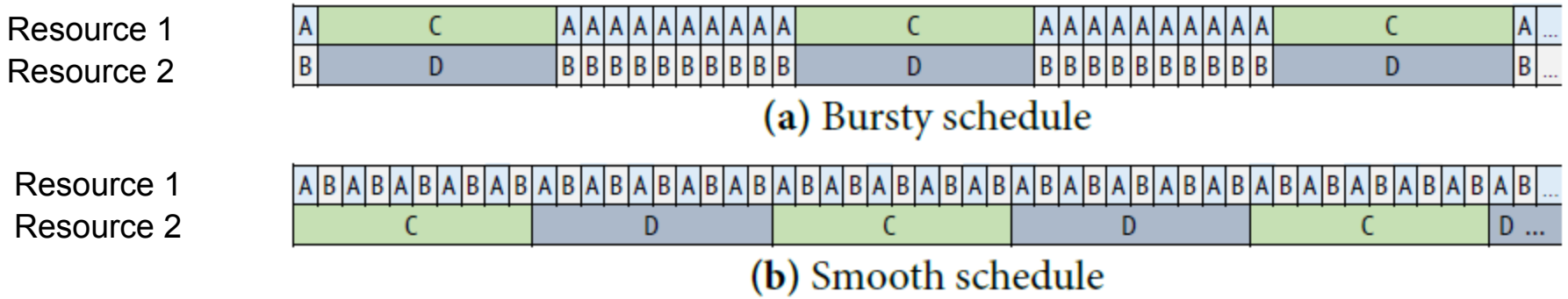
SIGCOMM '16, August 22–26, 2016, Florianopolis, Brazil

DOI: http://dx.doi.org/10.1145/2934872.2934880

[1]We use the terms application and framework interchangeably in this paper. Users can submit multiple jobs to each framework.

Challenge (4): Processing Allocation

- Different applications do not only fight for network resources, but also for **processing resources** (threads) in the same server
- **Challenge:** When using classical fair scheduling, heavy applications may use all resources at the same time, temporarily starving light applications
- Example with 4 applications (A,B,C,D):



2DFQ

Key ideas:

(a) Separate light applications from heavy applications

(b) In case of doubt whether an application is heavy or light, assume it is heavy

2DFQ: Two-Dimensional Fair Queueing for Multi-Tenant Cloud Services

Jonathan Mace[1], Peter Bodik[2], Madanlal Musuvathi[2], Rodrigo Fonseca[1], Krishnan Varadarajan[2]

[1]Brown University, [2]Microsoft

ABSTRACT

In many important cloud services, different tenants execute their requests in the thread pool of the same process, requiring fair sharing of resources. However, using fair queue schedulers to provide fairness in this context is difficult because of high execution concurrency, and because request costs are unknown and have high variance. Using fair schedulers like WFQ and WF^{2}Q in such settings leads to *bursty schedules*, where large requests block small ones for long periods of time. In this paper, we propose Two-Dimensional Fair Queueing (2DFQ), which spreads requests of different costs across different threads and minimizes the impact of tenants with unpredictable requests. In evaluation on production workloads from Azure Storage, a large-scale cloud system at Microsoft, we show that 2DFQ reduces the burstiness of service by 1-2 orders of magnitude. On workloads where many large requests compete with small ones, 2DFQ improves 99th percentile latencies by up to 2 orders of magnitude.

CCS Concepts

•**Networks** → **Cloud computing**; *Packet scheduling*; •**Computer systems organization** → *Availability*;

Keywords

Fair Request Scheduling; Multi-Tenant Systems

1. INTRODUCTION

Many important distributed systems and cloud services execute requests of multiple tenants simultaneously. These include storage, configuration management, database, queueing, and co-ordination services, such as Azure Storage [9], Amazon Dynamo [16], HDFS [53], ZooKeeper [36], and many more. In this context, it is crucial to provide resource isolation to ensure that a single tenant cannot get more than its fair share of resources, to prevent aggressive tenants or unpredictable workloads from causing starvation, high latencies, or reduced throughput for others. Systems in the past have suffered cascading failures [19, 27], slowdown [14, 20, 27, 28, 33], and even cluster-wide outages [14, 19, 27] due to aggressive tenants and insufficient resource isolation.

However, it is difficult to provide isolation in these systems because multiple tenants execute *within the same process*. Consider the HDFS NameNode process, which maintains metadata related to locations of blocks in HDFS. Users invoke various APIs on the NameNode to create, rename, or delete files, create or list directories, or look up file block locations. As in most shared systems, requests to the NameNode wait in an admission queue and are processed in FIFO order by a set of worker threads. In this setting tenant requests contend for resources, such as CPU, disks, or even locks, from *within* the shared process. As a result, traditional resource management mechanisms in the operating system and hypervisor are unsuitable for providing resource isolation because of a mismatch in the management granularity.

In many domains, resource isolation is implemented using a fair queue scheduler, which provides alternating service to competing tenants and achieves a fair allocation of resources over time. Fair schedulers such as Weighted Fair Queueing [46], which were originally studied in the context of packet scheduling, can be applied to shared processes since the setting is similar: multiple tenants submit flows of short-lived requests that are queued and eventually processed by a server of limited capacity. However, in shared processes there are three additional challenges that must be addressed:

- **Resource concurrency:** Thread pools in shared processes execute many requests concurrently, often tens or even hundreds, whereas packet schedulers are only designed for sequential execution of requests (*i.e.* on a network link);
- **Large cost variance:** Request costs vary by at least 4 orders of magnitude across different tenants and API types, from sub-millisecond to many seconds. By contrast, network packets only vary in length by up to 1.5 orders of magnitude (between 40 and 1500 bytes). Unlike CPU thread schedulers, requests are not preemptible by the application;
- **Unknown and unpredictable resource costs:** The execution time and resource requirements of a request are not known at schedule time, are difficult to estimate up front, and vary substantially based on API type, arguments, and transient system state (*e.g.*, caches). By contrast, the length of each network packet is known a priori and many packet schedulers rely on this information.

SIGCOMM '16, August 22 - 26, 2016, Florianopolis , Brazil

 ISBN 978-1-4503-4193-6/16/08 ... $15.00

DOI: http://dx.doi.org/10.1145/2934872.2934878

Wednesday, August 24, 2016

8:30am - 10:10am Session 4 - Scheduling

Session Chair: Sergey Gorinsky *(IMDEA Networks Institute)*

2DFQ: Two-Dimensional Fair Queueing for Multi-Tenant Cloud Services Paper
Jonathan Mace *(Brown University)*, Peter Bodik *(Microsoft)*, Madanlal Musuvathi *(Microsoft)*, Rodrigo Fonseca *(Brown University)*, Krishnan Varadarajan *(Microsoft)*

CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark Paper
Hong Zhang *(Hong Kong University of Science and Technology)*, Li Chen *(Hong Kong University of Science and Technology)*, Bairen Yi *(Hong Kong University of Science and Technology)*, Kai Chen *(Hong Kong University of Science and Technology)*, Mosharaf Chowdhury *(University of Michigan)*, Yanhui Geng *(Huawei Noah's Ark Lab)*

Scheduling Mix-flows in Commodity Datacenters with Karuna Paper
Li Chen *(Hong Kong University of Science and Technology)*, Kai Chen *(Hong Kong University of Science and Technology)*, Wei Bai *(Hong Kong University of Science and Technology)*, Mohammad Alizadeh *(MIT)*

NUMFabric: Fast and Flexible Bandwidth Allocation in Datacenters Paper
Kanthi Nagaraj *(Stanford University)*, Dinesh Bharadia *(Stanford University)*, Hongzi Mao *(M.I.T)*, Sandeep Chinchali *(Stanford University)*, Mohammad Alizadeh *(M.I.T.)*, Sachin Katti *(Stanford University)*

Obrigado!