

國立陽明交通大學
網路工程研究所
碩士論文
初稿

Institute of Network Engineering
National Yang Ming Chiao Tung University
Master Thesis

多台交換機架構之網路內部合流排程

In-network multi-switch coflow scheduling

研究生：陳沂葳 (Yi-Wei Chen)

指導教授：林靖茹 (Kate Ching-Ju Lin)

中華民國 一一三年一月
November 2023

多台交換機架構之網路內部合流排程
In-network multi-switch coflow scheduling

研 究 生：陳沂葳
指導教授：林靖茹

Student: Yi-Wei Chen
Advisor: Dr. Kate Ching-Ju Lin

國立陽明交通大學
網路工程研究所
碩士論文 初稿

A Thesis Draft
Submitted to Institute of Network Engineering
College of Science
National Yang Ming Chiao Tung University
in partial Fulfilment of the Requirements
for the Degree of
Master
in
Computer Science

November 2023

Taiwan, Republic of China

中華民國 一一三年一月

多台交換機架構之網路內部合流排程

學生：陳沂葳

指導教授：林靖茹 博士

國立陽明交通大學 網路工程研究所

摘 要

資料中心網路在現今扮演至關重要的角色，而有效率的匯流排程則是確保最佳性能的關鍵。匯流排程旨在管理擁有共同應用層級目標的數據流。然而，現有的方法往往面臨與性能、準確性和可拓展性的挑戰。在本論文中，我們提出了在多台交換機架構之下的網路內部合流排程系統，以解決資料中心網路中的挑戰。我們的方法透過將即時的匯流辨識拓展至跨交換機的維度，藉以促進分散式的合作。這一系統在資料中心網路中有效率地提升了性能、可拓展性以及應用層級的效率。

關鍵詞: 匯流排程、網路內部排程、匯流辨識。

In-network multi-switch coflow scheduling

Student: Yi-Wei Chen

Advisor: Dr. Kate Ching-Ju Lin

Institute of Network Engineering
National Yang Ming Chiao Tung University

Abstract

Datacenter networks play a pivotal role in today's digital landscape, and the efficient scheduling of coflows is critical to ensuring optimal performance. Coflow scheduling aims to manage data flows that share a common application-level goal. However, existing approaches often face challenges related to performance, accuracy, and scalability. In this thesis, we present the In-Network Multi-Switch Coflow Scheduling system, addressing challenges in datacenter networks. Our approach extends real-time coflow classification across switches, fostering distributed cooperation. This system efficiently enhances performance, scalability, and application efficiency in datacenter networks.

Keyword: Coflow Scheduling, In-network Scheduling, Coflow Recognition

Table of Contents

摘要	i
Abstract	ii
Table of Contents	iii
List of Figures	v
List of Tables	vi
1 Introduction	1
2 Related Work	3
2.1 Flow Scheduling	3
2.2 Clairvoyant Coflow Scheduling	3
2.3 Non-Clairvoyant Coflow Scheduling	4
3 Background and Motivation	6
3.1 Background	6
3.2 Motivation Example	7
3.3 Challenge	8
4 Design	10
4.1 Overview	10
4.2 Coflow queue mapping	12
4.3 Mapping threshold configuration	12
4.4 Local Prioritization based on Global Coflow Size	14
5 Evaluation	16
5.1 Experimental Setup	16
5.2 Baseline	16
5.3 Metrics	17
5.3.1 Coflow estimation size	17
5.3.2 Coflow priority accuracy (average)	19

5.3.3 Coflow priority accuracy (top k)	19
6 Conclusion	24
References	25

List of Figures

3.1	Motivation example	9
3.2	Objective	9
4.1	System Overview	11
4.2	Coflow queue mapping	13
4.3	Local Prioritization based on Global Coflow Size	15
5.1	Coflow size estimation	18
5.2	Coflow priority accuracy (Average)	20
5.3	Coflow priority accuracy (Top k)	22
5.4	Coflow priority accuracy (Top k) under different coflow sizes	23

List of Tables

Chapter 1

Introduction

Over the years, there has been a surging demand for distributed computing, attributed to its cost-effectiveness and impressive scalability, especially in datacenter network environments. Within this architectural context, data flows pertaining to a singular task often possess a unified objective, such as the reduction of the completion time for the most time-consuming flow. While numerous studies [1–10] emphasize scheduling individual flows to optimize overall flow completion time, they often neglect the crucial aspect of enhancing the application-level completion time.

The concept of coflow was first introduced in [11] to underscore the significance of scheduling a set of flows with a shared application-level objective. Coflow scheduling methods [12–30] focus on prioritizing the average completion time of coflows, rather than merely considering individual flow completion times, can bolster performance from an application-centric perspective. Numerous studies have concentrated on coflow scheduling using an offline model, wherein the information regarding all incoming coflows is provided upfront. In contrast, other research endeavors have proposed methods that offer scheduling techniques without any prior knowledge of the coflows. Nonetheless, a majority of the aforementioned studies necessitate a centralized server or controller to execute coflow scheduling. This approach can readily introduce a bottleneck or emerge as a single point of vulnerability. Additionally, these works often conceptualize the switches between the data source and the destination within the datacenter network as a singular, large switch equipped with multiple ports. This implies that information regarding the distribution of coflows across various switches is presupposed from the onset.

In this paper, we introduce a pioneering approach tailored explicitly for in-network coflow scheduling in a multi-switch environment, obviating the need for a central controller. Our contributions, especially designed to cater to the intricacies of multiple switch structures, are enu-

merated as follows:

- **Localized Flow Feature Extraction and Cross-switch Classification:** To harness the inherent parallelism of a multi-switch structure, we equip individual switches to extract flow features locally. This granular data then aids in performing cross-switch classification within the control plane, optimizing the interconnected operations in a multi-switch setup.
- **Coflow Queue Mapping for Multi-switch Environments:** Recognizing the unique challenges of multi-switch infrastructures, we've mapped coflow queues across these switches. This allows for an accurate computation of the global size of coflows, which is indispensable in determining and upholding the accurate priority of each coflow.
- **Priority Aggregation and Dissemination Across Switches:** In our system, global priority data is aggregated from diverse local switches. This aggregated priority, once formulated, is relayed back to each local switch, ensuring a harmonized operational paradigm across the expansive multi-switch landscape.

Our tailored contributions advocate for a more decentralized, robust, and efficient coflow scheduling system, ideally suited for the complexities and potential of multi-switch architectures.

Chapter 2

Related Work

2.1 Flow Scheduling

Traditionally, networks have relied on the concept of flows, which represent a sequence of packets traveling from a single source to a single destination. This abstraction has traditionally served well, particularly in optimizing network designs for latency or throughput in point-to-point connections, critical for conventional applications like file transfers and web access. Numerous prior studies have concentrated on network-level optimization, targeting metrics such as flow completion time [2–5]. For instance, D3 [2] investigates the feasibility of utilizing deadline information to manage traffic introduction rates at end hosts. PDQ [3] introduces a distributed algorithm that enables a set of switches to collaboratively collect information about flow workloads and converge to stable allocation decisions. pFabric [4] devises a minimalistic data center transport scheme that achieves near-optimal flow completion times by independently encoding packet priorities and strictly buffering packets based on their assigned priority. PIFO [5] introduces a sorting mechanism for an array of PIFO elements, effectively determining the order and timing for packet scheduling.

2.2 Clairvoyant Coflow Scheduling

In certain distributed computing frameworks such as MapReduce, Dryad, and Spark, data flows within a job often share a common performance objective, such as minimizing the completion time of the slowest flow. However, these application-level requirements are frequently overlooked in the context of prior works, as discussed in the previous section, which predominantly concentrate on individual flow completion times, potentially hindering overall application

performance.

The concept of coflow, introduced by [11], was devised to address the scheduling needs of these frameworks. A coflow is defined as a group of parallel flows sharing a common application-level goal (e.g., shuffle flows in MapReduce). Coflow scheduling aims to enhance the average completion time of coflows, referred to as CCT. Typically, minimizing CCT is achieved by approximating the "shortest remaining job first" (SRJF) policy, which necessitates comprehensive knowledge of coflow parameters, such as coflow size and start times. Varys [13] develops efficient scheduling heuristics and algorithms that enable improved application-level performance and predictable completions within coflow deadlines. Rapier [16] optimizes application performance by jointly considering routing and scheduling at the coflow level, rather than treating individual flows separately. Sincronia [21] employs a simple, yet effective, greedy algorithm that periodically organizes the set of unfinished coflows, achieving nearly optimal average CCT without relying on an explicit per-flow rate allocation mechanism.

While these approaches offer robust scheduling mechanisms with coflow considerations, it's important to note that they require comprehensive prior knowledge of coflow information, which may prove impractical in some scenarios.

2.3 Non-Clairvoyant Coflow Scheduling

Gathering all the necessary coflow information in advance can be burdensome and result in significant overhead for coflow scheduling, especially given the typical short-burst nature of data in modern data center networks. Therefore, there are approaches that integrate scheduling techniques in-network or during runtime. Baraat [12] adopts a First-In-First-Out (FIFO) task scheduling approach while dynamically adjusting the network's multiplexing level to prevent head-of-line blocking. Aalo [17] operates without prior knowledge and utilizes Discretized Coflow-Aware Least-Attained Service (D-CLAS) to categorize coflows into a limited number of priority queues based on their progress in data transmission across the cluster. CODA [19] stands out as the first work to identify coflows among individual flows using machine learning

techniques. This led to the development of an error-tolerant coflow scheduler. Yosemite [?] advocates scheduling coflows based on assigned weights, where these weights express the importance or priorities of various coflows and their associated applications.

However, it's worth noting that most of these works treat the entire network structure as a single switch and assume that each switch has access to information from other switches, which may not accurately reflect real-world scenarios.

In our work, we address this limitation by implementing information exchange across multiple switches to enable real-time accumulation of coflow size, a critical component of effective coflow scheduling.

Chapter 3

Background and Motivation

3.1 Background

Coflow scheduling: A coflow, characterized as a collection of parallel flows with a shared application objective, plays a pivotal role in distributed computing. Efficiently scheduling a multitude of coflows distributed across various machines necessitates the acquisition of comprehensive coflow information. In the realm of coflow scheduling, the "shortest job first" (SJF) policy is widely recognized for its optimal performance in terms of coflow completion time. However, a notable challenge lies in the substantial overhead associated with collecting and processing data. This inherent cost often compels prior works to strike a delicate balance, striving to approximate the scheduling policy as closely as possible to SJF, while mitigating the data collection burden. Achieving an optimal equilibrium between data collection workload and coflow scheduling performance becomes imperative.

Host-Assist Scheduling: Traditional coflow scheduling solutions have predominantly relied on a centralized server to aggregate crucial coflow information, including coflow identifiers and sizes, and to orchestrate global scheduling. This centralized scheduling approach is commonly referred to as host-assist scheduling since it mandates the involvement of end hosts once the centralized server provides the requisite information. However, there are three primary drawbacks associated with host-assist scheduling.

Firstly, it imposes substantial communication overhead and computational costs. This behavior is incongruent with the characteristics of contemporary data center networks, where coflows typically exhibit relatively small sizes.

Secondly, it raises concerns about single points of failure, making the centralized server vulnerable to becoming a bottleneck in the system's operations.

Lastly, centralized schedulers necessitate end host cooperation, thus requiring modifications to end hosts to facilitate subsequent scheduling processes.

In-Network Scheduling: In-Network scheduling systems revolutionize coflow scheduling by eliminating the necessity for host cooperation. They effectively shift the scheduling responsibility from a centralized controller to programmable switches. This innovative approach circumvents the need for upfront collection of coflow information and the requirement for host modifications, thereby addressing the three drawbacks associated with host-assist scheduling, as discussed in the previous section.

3.2 Motivation Example

In-Network Scheduling, achieved through programmable switches in the data pipeline, relies on approximating scheduling policies such as SJF by gathering job information, like the mean size of a set of tasks, and grouping jobs with similar characteristics into the same coflow. While this method functions effectively within a single switch, its accuracy diminishes when incoming coflows span multiple switches within the same epoch. This is primarily due to each switch having access only to its own incoming jobs and lacking the ability to exchange information with other switches. Consequently, in scheduling policies like SJF, where the aggregated coflow size significantly impacts scheduling performance, the accuracy of coflow scheduling priorities may vary across switches, thereby influencing packet order outcomes.

Let's illustrate the concept with a practical scenario involving an In-Network scheduler where two switches, denoted as S_1 and S_2 , are operating. At a particular time, three coflows, namely C_1 , C_2 , and C_3 , with respective total sizes of 5, 5, and 6 units, are introduced. In this case, the packets belonging to C_3 should logically be scheduled after those from C_1 and C_2 because a larger coflow size assigns lower priority. Therefore, both S_1 and S_2 should follow this priority order.

However, consider a situation where C_3 is split into two flows, with one flow forwarding to S_1 and the other to S_2 . We can denote these flows as f_{ij} , where i signifies the coflow ID,




and j signifies the switch ID. With specific sizes as $f_{11} = 5$, $f_{22} = 5$, $f_{31} = 3$, and $f_{32} = 3$, the correct coflow priority is obvious from a global perspective. However, each individual switch possesses knowledge solely about its own incoming flows. For instance, S_1 is aware of f_{11} (size 5) and f_{31} (size 3), thus deducing that C_3 holds higher priority than C_1 . Likewise, in S_2 , it is privy to f_{22} (size 5) and f_{32} (size 3), leading to an identical conclusion as that of S_1 .

This outcome from both switches contradicts the globally assigned coflow priority because they erroneously classify C_3 as a smaller coflow, even though, in reality, both f_{31} and f_{32} represent two small flows within a larger coflow. This example highlights how the distribution of coflows across multiple switches, coupled with the lack of information exchange, can lead to inaccurate coflow priority assignments.

In response to the issues highlighted in the preceding example, our objective is to estimate the global coflow size by mapping the major queue of each switch and subsequently implement local prioritization based on the calculated global coflow size.

3.3 Challenge

- **How to map the coflow queues from different switches?**
- **How to determine the timing for coflow queue mapping?**
- **How to record the mapping result for each coflow queue?**

Coflow	Size
	6
	5
	5

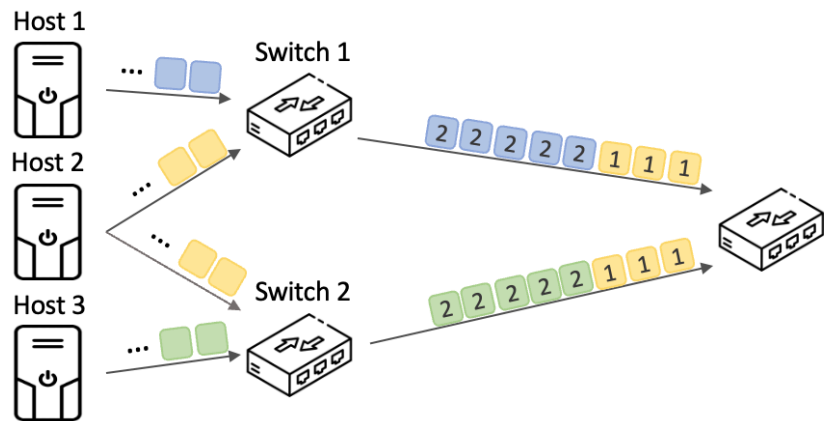





Figure 3.1: Motivation example

Coflow	Size
	6
	5
	5

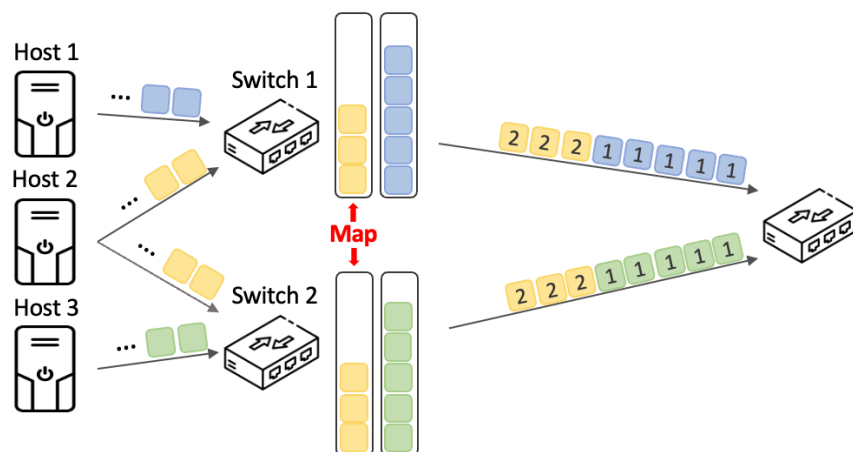


Figure 3.2: Objective

Chapter 4

Design

4.1 Overview

In response to the challenges outlined earlier, we introduce an innovative solution: the In-Network Multi-Switch Coflow Scheduling system. This fully distributed coflow scheduling system promotes cooperation among switches, allowing for local prioritization that closely approximates the global truth. Building upon the foundation laid by PICO [31], we extend the concept of real-time coflow classification beyond local boundaries to encompass cross-switch coflow queue mapping.

Fig. 4.1 provides an architectural overview of this design. The primary objective of this design is to identify a set of remote coflow queues for each local coflow queue. Furthermore, the mapping outcome serves as a representation of the distribution of each coflow across switches.

In implementing this design, we initiate the mapping process when the coflow queue reaches the designated mapping threshold. At this point, we sample flows from the coflow queue and extract flow features. These features are subsequently transmitted to remote switches. Upon arrival at the remote switch, the coflow classification process commences, ultimately returning a remote queue ID to the local switch, signifying the mapping outcome for the remote switch. This sequence is iterated across all remote switches, resulting in a comprehensive set of remote queue IDs.

It's worth noting that for each remote switch, there is at most one corresponding remote queue ID. This set of remote queue IDs is later stored in the local coflow queue's register, facilitating the local scheduling process.

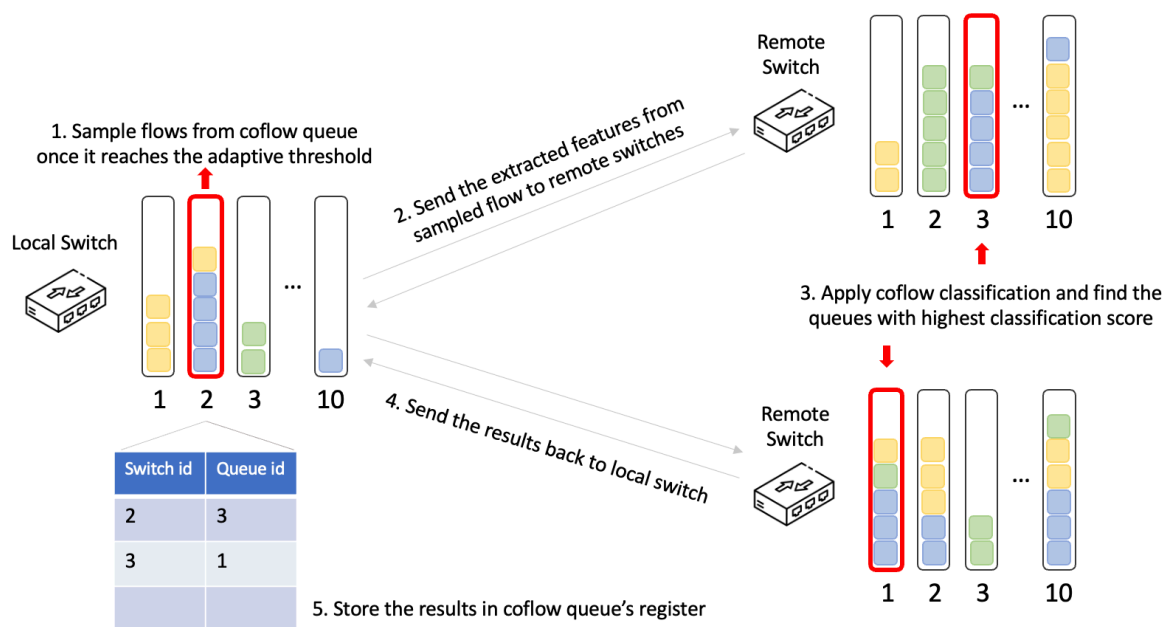


Figure 4.1: System Overview

4.2 Coflow queue mapping

In the proposed mapping mechanism, once the size of a coflow queue on a designated switch surpasses the mapping threshold, a selection procedure is implemented to sample a maximum of k flows from the coflow queue. These sampled flows are representative of the entire coflow queue and are utilized for further analysis. Essential flow features, such as mean packet size and arrival time, are then extracted from these sampled flows, drawing information from both the flow size sketch and flow record table. Subsequently, these features are transmitted to remote switches to initiate the coflow classification process.

At the remote switch, the coflow classification process is initiated [31]. Pairwise coflow detection [31] is performed for each flow in the sampled flows, comparing them with select flows from a remote coflow queue. A score is assigned to each pairwise coflow detection, reflecting the probability of the two flows belonging to the same coflow. Subsequently, these individual scores are aggregated into a single score for each coflow queue on the remote switch.

Upon completion of the classification process for all sampled flows, the coflow queue with the highest classify score is identified as the mapping coflow queue for the local coflow queue. It is crucial to acknowledge that the presence of a mapping queue is not guaranteed, as the classify score may be invalid or undefined. In such instances, it is probable that a suitable mapping queue cannot be identified for the local queue on that specific remote switch. In response, the remote switch communicates this information by sending error messages.

The identified mapping queues' IDs are then transmitted back to the local switch, treated as remote queue IDs. These remote queue IDs are subsequently stored in the local queue's register for future reference and utilization in the local scheduling process.

4.3 Mapping threshold configuration

The determination of the mapping threshold necessitates a careful balance between performance and mapping accuracy. Opting for a low mapping threshold enhances mapping accu-

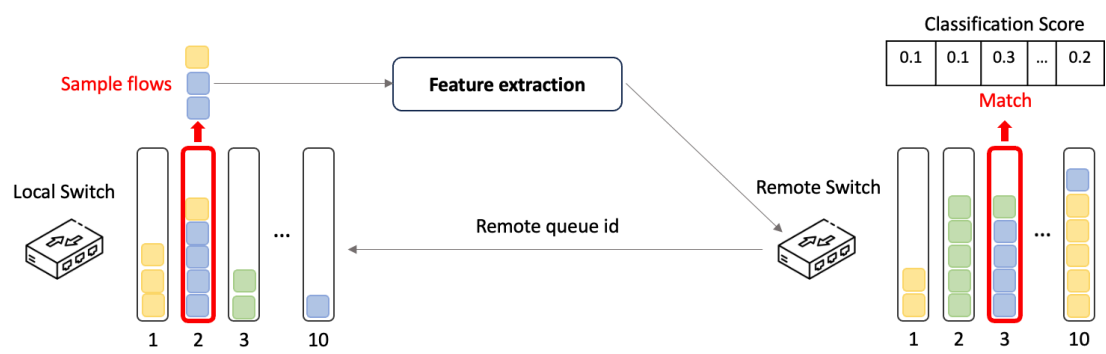


Figure 4.2: Coflow queue mapping

racy, as it readily adapts to changes in data distribution. However, this approach may introduce a trade-off in terms of performance, given the elevated overhead of information exchange between switches and the heightened frequency of coflow classification processes. Conversely, assigning a high mapping threshold can reduce the overhead associated with the coflow classification process but may result in decreased accuracy in the mapping outcome.

Our observations indicate that the mapping result tends to be more unstable when the coflow queue size is relatively small. Conversely, as the coflow queue size increases, the mapping result becomes more stable over time. Consequently, we have devised an adaptive method for selecting the mapping threshold based on the size of the coflow queue. This adaptive approach ensures that the mapping threshold remains low when dealing with small coflow queues, gradually increasing as the coflow queue size grows larger. By implementing this strategy, we provide each coflow queue with an adaptive mapping threshold that caters to its specific requirements.

4.4 Local Prioritization based on Global Coflow Size

In the process of assigning local priorities, all flows within the same coflow queue are treated as part of a unified coflow entity. By referencing the flow size sketch, a local estimation of coflow size is derived. To augment this local estimation with a global perspective, local switches access the register within each coflow queue. This coflow queue register provides a numerical pair denoting switch ID and coflow queue ID, respectively. By leveraging this information, the global coflow size can be ascertained by summing the flow sizes within its mapping queues.

Armed with the global coflow size estimation, the local switch can then proceed to calculate priorities based on this holistic view of the coflow's size. This approach ensures that local prioritization aligns with the broader context of the entire network, ultimately enhancing the scheduling and performance of coflows. The process is shown in Fig. 4.3

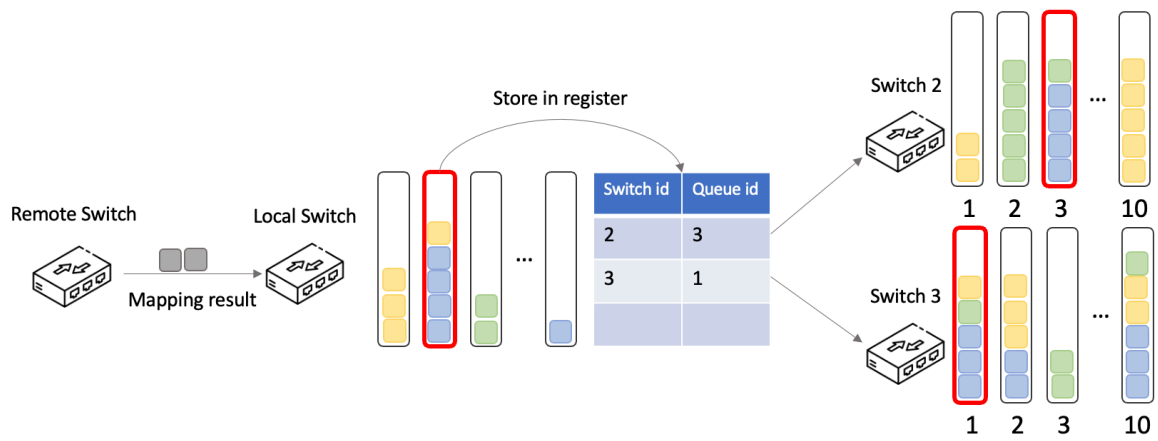


Figure 4.3: Local Prioritization based on Global Coflow Size

Chapter 5

Evaluation

5.1 Experimental Setup

Hardware setup: The simulation runs on a Ubuntu 20.04.5 server equipped with a 8-core Intel i7-7700K 4.20GHz CPU and 64GB RAM, and a GeForce GTX 1080 Ti GPU.

Dataset: We use a realistic cluster trace collected by Facebook from a 3,000 machines and 150-rack MapReduce cluster with 10:1 over-subscription as the workload of our simulations. The data-trace contains over 700,000 flows, which belong to over 500 coflows.

Pairwise detection model: DNN model with 2 hidden layers and 16 neural in each layer.

5.2 Baseline

We conduct a comparative analysis with the In-Network Coflow Scheduling approach [31]. The previous work [31] centers on real-time coflow identification, independent of supplementary information from hosts or a centralized controller. The entire coflow categorization process occurs within a programmable switch, aided by a DNN model in the control plane to facilitate pairwise coflow detection.

Our primary emphasis in this comparison lies in evaluating how the estimation of global coflow sizes can enhance local coflow prioritization in a multi-switch scenario. We posit that aggregating the global size of coflows based on information obtained from individual local switches can effectively contribute to achieving this objective.

5.3 Metrics

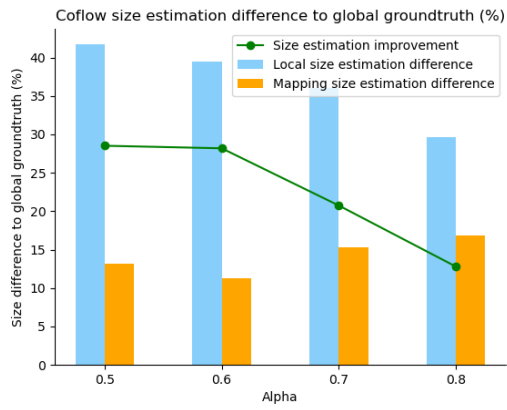
5.3.1 Coflow estimation size

Let the global ground truth be defined as the global coflow size. We evaluate the disparities between the local coflow size estimation and the global ground truth, as well as the disparities between the mapping coflow size estimation and the global ground truth.

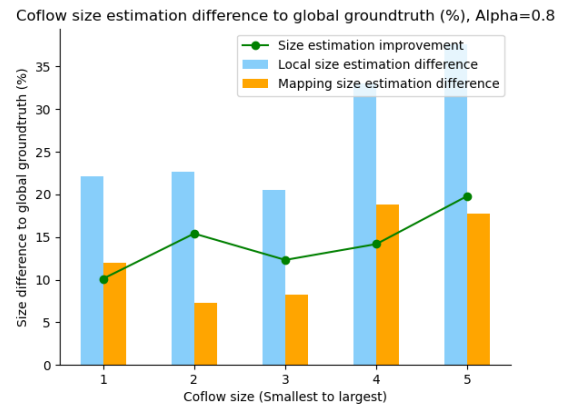
Impact of α Value We posit that each incoming coflow possesses a preferred major switch destination. The α value serves as a quantification of the coflow’s inclination to select this major switch. For instance, with $\alpha=0.5$, it implies that each flow within a coflow has a 50% likelihood of choosing its designated major switch, and a complementary 50% chance of randomly selecting from the remaining switches. Hence, as the value of α approaches 0.5, it signifies a more balanced distribution of the coflow among its major switch and minor switches. Conversely, when α approaches 1.0, it indicates a more pronounced imbalance in the distribution.

In Figure 5.1 (a), we observe a noteworthy enhancement in size estimation, peaking at nearly 30% and gradually diminishing to approximately 15% as α approaches 0.8. The diminishing improvement percentage with increasing α is attributed to a higher concentration of flows on a coflow’s major switch, resulting in a reduced size estimation disparity on that specific switch. While this implies a substantial size estimation difference on other minor switches, their impact is minor when computing the overall size estimation difference.

Impact of coflow size We partitioned the dataset into five segments based on coflow size to investigate the impact of coflow size on coflow size estimation. Given our emphasis on rectifying coflow distribution imbalance, we set α to 0.8 for this specific experiment. In Figure 5.1 (b), the size estimation improvement consistently exhibits an ascending trend with increasing coflow size. This observation aligns with the intuitive expectation that larger coflow sizes result in more pronounced disparities between local coflow size estimations and the actual global groundtruth.



(a) Impact of alpha



(b) Impact of coflow size

Figure 5.1: Coflow size estimation

5.3.2 Coflow priority accuracy (average)

Consider the global groundtruth as the definitive global coflow priority order, represented as a sequence: C_1, C_2, \dots, C_n . Priority accuracy is defined as the count of distinct pairs (C_x, C_y) with the correct relation, divided by the total count of distinct pairs (C_x, C_y) , where x and y are distinct numbers ranging from 1 to n , with $x < y$. We quantify accuracy improvement by comparing the difference between local priority accuracy and mapping priority accuracy.

Impact of α Value In Figure 5.2, a notable improvement in accuracy on the priority sequence is evident, particularly as the value of α approaches 0.8. This enhancement is attributed to the unbalanced distribution of large coflows across different switches, with a majority of flows concentrated at the major switch and the remaining distributed among minor switches. From the perspective of the minor switches, this distribution pattern may result in misclassification of a large coflow as a smaller one, leading to a significant deviation in the coflow priority order compared to the global coflow priority.

5.3.3 Coflow priority accuracy (top k)

The concept of the top k coflow priority aligns with the average coflow priority accuracy, with a specific focus on the uppermost k elements of the priority sequence. For instance, if $k = 1/2$, the attention is directed to the sequence: $C_{\frac{1}{2}}, \dots, C_n$. In this context, the primary concern is the correctness of elements within this sequence, without emphasizing their specific order. This assessment aims to evaluate our system's capability to accurately identify and prioritize large coflows, thereby preventing misclassification that could impede coflow scheduling.

Impact of α Value By observing Figure 5.3, we note a trend similar to Figure 5.2. In both graphs, there is a noticeable increase in accuracy improvement as α approaches 0.8. This observation substantiates our method's efficacy, not only in delivering a more accurate coflow priority order but also in facilitating a superior selection of large coflows.

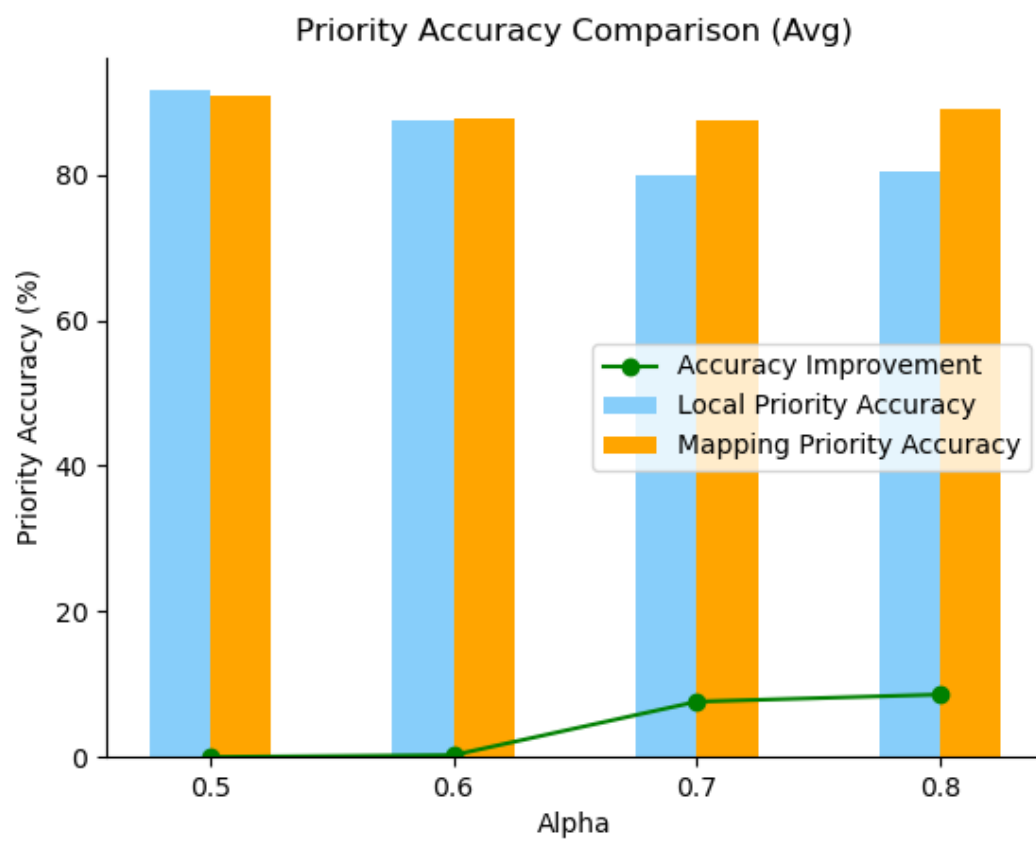


Figure 5.2: Coflow priority accuracy (Average)

Impact of coflow size and k Subsequently, we analyzed the influence of coflow size, examining Figure 5.4 (a) through Figure 5.4 (d), in conjunction with varying values of k for top k analysis. We extended the range of k from 0.1 to 0.5, symbolizing the size of the window within which we concentrate on the coflow priority sequence.

The observed trend in these graphs indicates a marginal increase in priority accuracy as k expands, regardless of coflow size. This phenomenon may be attributed to the larger window size, offering a heightened probability of encompassing the correct coflows, even without strict adherence to their precise order.

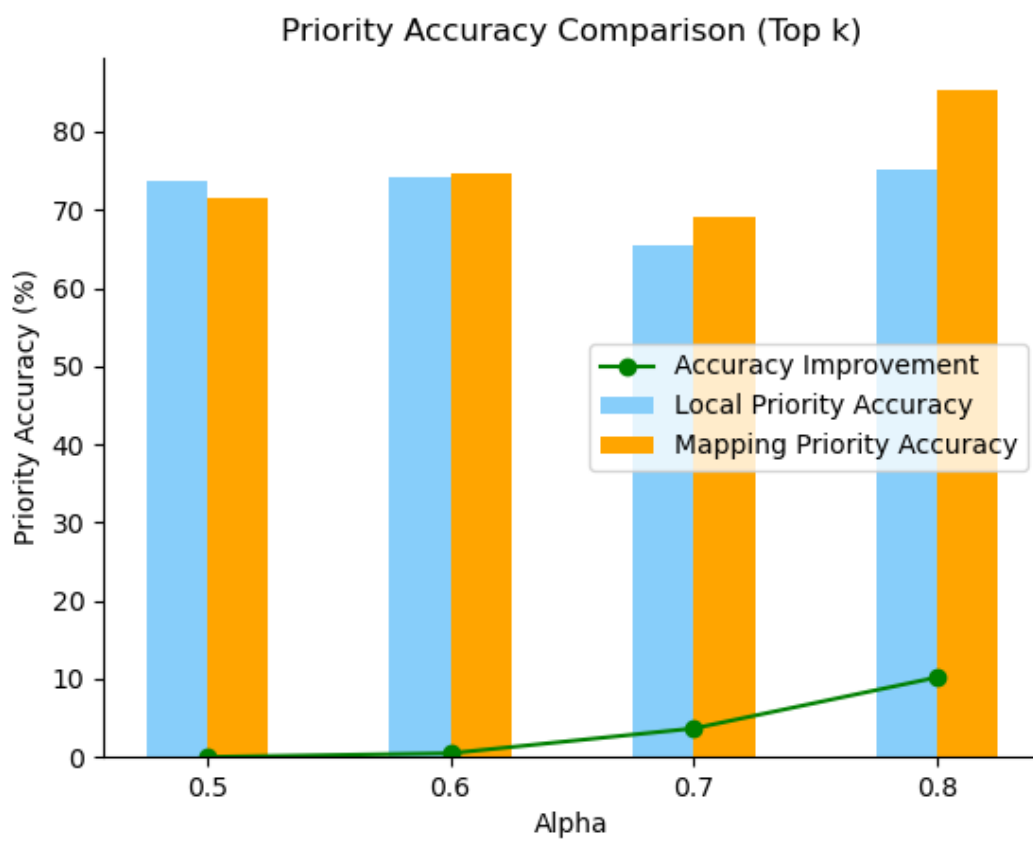
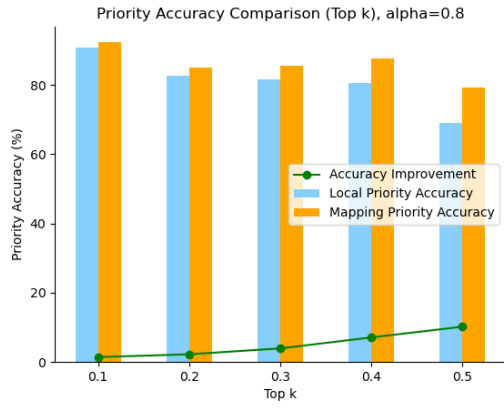
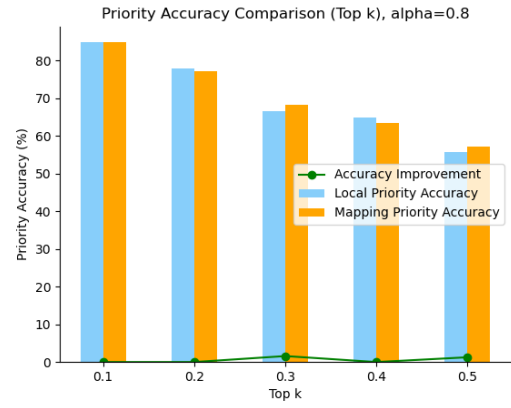


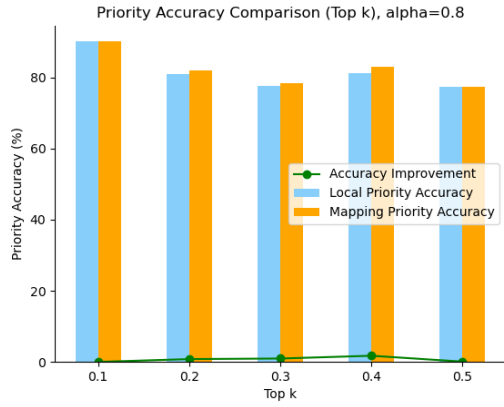
Figure 5.3: Coflow priority accuracy (Top k)



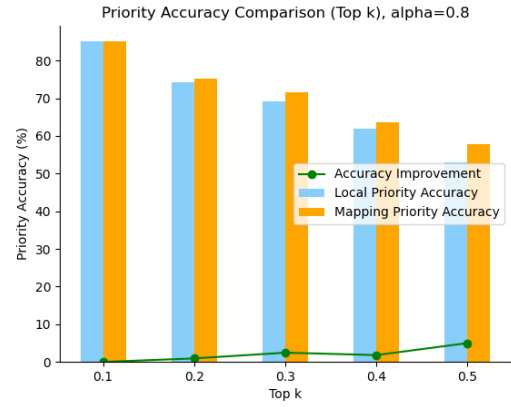
(a) Coflow sizes=1 (Smallest)



(b) Coflow sizes=2



(c) Coflow sizes=3



(d) Coflow sizes=4 (Largest)

Figure 5.4: Coflow priority accuracy (Top k) under different coflow sizes

Chapter 6

Conclusion

In this thesis, we introduced a novel and efficient approach to coflow scheduling in datacenter networks. Our In-Network Multi-Switch Coflow Scheduling system addresses the challenges associated with coflow scheduling by enabling collaboration between switches to approximate global coflow prioritization. We extended real-time coflow classification to encompass cross-switch coflow queue mapping, resulting in a distributed and cooperative scheduling solution.

Our approach introduces adaptability in mapping threshold configuration, ensuring that mapping accuracy and performance are carefully balanced. By dynamically adjusting the mapping threshold based on the size of the coflow queue, we provide each coflow with an appropriate threshold to cater to its specific needs.

Furthermore, our system employs local prioritization based on global coflow size, enhancing the accuracy and fairness of coflow scheduling. This approach considers the entire network context by deriving a global estimation of coflow size from local information, thus enabling more effective prioritization.

In summary, our In-Network Multi-Switch Coflow Scheduling system offers a practical and efficient solution to the challenges of coflow scheduling in datacenter networks. By incorporating distributed cooperation, adaptive mapping thresholds, and global-local prioritization, we enhance the performance and scalability of datacenter networks, ultimately contributing to improved network efficiency and application performance.

References

- [1] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, “Managing data transfers in computer clusters with orchestra,” *ACM SIGCOMM*, 2011.
- [2] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, “Better never than late: Meeting deadlines in datacenter networks,” *ACM SIGCOMM*, 2011.
- [3] C.-Y. Hong, M. Caesar, and P. B. Godfrey, “Finishing flows quickly with preemptive scheduling,” in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2012.
- [4] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, “Pfabric: Minimal near-optimal datacenter transport,” *ACM SIGCOMM*, 2013.
- [5] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown, “Programmable packet scheduling at line rate,” in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.
- [6] Z. Yu, C. Hu, J. Wu, X. Sun, V. Braverman, M. Chowdhury, Z. Liu, and X. Jin, “Programmable packet scheduling with a single queue,” in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, p. 179–193.
- [7] P. Gao, A. Dalleggio, Y. Xu, and H. J. Chao, “Gearbox: A hierarchical packet scheduler for approximate weighted fair queuing,” in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.
- [8] T. Yang, J. Li, Y. Zhao, K. Yang, H. Wang, J. Jiang, Y. Zhang, and N. Zhang, “Qcluster: Clustering packets for flow scheduling,” in *Proceedings of the ACM Web Conference 2022*, 2022.
- [9] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks,” 2010.

- [10] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, “Detail: Reducing the flow completion time tail in datacenter networks,” *SIGCOMM Comput. Commun. Rev.*, 2012.
- [11] M. Chowdhury and I. Stoica, “Coflow: A networking abstraction for cluster applications,” in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012.
- [12] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, “Decentralized task-aware scheduling for data center networks,” *ACM SIGCOMM*, 2014.
- [13] M. Chowdhury, Y. Zhong, and I. Stoica, “Efficient coflow scheduling with varys,” *ACM SIGCOMM*, 2014.
- [14] Z. Qiu, C. Stein, and Y. Zhong, “Minimizing the total weighted completion time of coflows in datacenter networks,” in *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, 2015.
- [15] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, “Network-aware scheduling for data-parallel jobs: Plan when you can,” *ACM SIGCOMM*, 2015.
- [16] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, “Rapier: Integrating routing and scheduling for coflow-aware data center networks,” in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015.
- [17] M. Chowdhury and I. Stoica, “Efficient coflow scheduling without prior knowledge,” *ACM SIGCOMM*, 2015.
- [18] Y. Li, S. H.-C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. C. M. Lau, “Efficient online coflow routing and scheduling,” in *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2016.
- [19] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, “Coda: Toward automatically identifying and scheduling coflows in the dark,” in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

- [20] H. Zhang, X. Shi, X. Yin, and Z. Wang, “Yosemite: Efficient scheduling of weighted coflows in data centers,” in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, 2017.
- [21] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, “Sincronia: Near-optimal network design for coflows,” in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018.
- [22] B. Li, G. Zuo, W. Bai, and L. Zhang, “1pipe: Scalable total order communication in data center networks,” in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021.
- [23] C. Tian, Y. Wang, B. Tian, Y. Zhao, Y. Zhou, C. Wang, H. Guan, W. Dou, and G. Chen, “Pushbox: Making use of every bit of time to accelerate completion of data-parallel jobs,” *IEEE Transactions on Parallel and Distributed Systems*, 2022.
- [24] A. Jajoo, Y. C. Hu, and X. Lin, “A case for sampling-based learning techniques in coflow scheduling,” *IEEE/ACM Transactions on Networking*, 2022.
- [25] X. Sunny, X. Huang, S. Sun, and T. S. E. Ng, “Sunflow: Efficient optical circuit scheduling for coflows,” *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016.
- [26] A. Jajoo, R. Gandhi, and Y. C. Hu, “Graviton: Twisting space and time to speed up coflows,” in *Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing*, 2016.
- [27] A. Jajoo, Y. C. Hu, and X. Lin, “Your coflow has many flows: Sampling them for fun and speed,” in *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, 2019.
- [28] H. Jahanjou, E. Kantor, and R. Rajaraman, “Asymptotically optimal approximation algorithms for coflow scheduling,” in *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, 2017.

- [29] J. W. Park, A. Tumanov, A. Jiang, M. A. Kozuch, and G. R. Ganger, “3sigma: Distribution-based cluster scheduling for runtime uncertainty,” in *Proceedings of the Thirteenth EuroSys Conference*, 2018.
- [30] S. Khuller and M. Purohit, “Brief announcement: Improved approximation algorithms for scheduling co-flows,” in *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, 2016.
- [31] J. Du and K. C.-J. Lin, “Distributed in-network coflow scheduling,” in *2022 IEEE 30th International Conference on Network Protocols (ICNP)*, 2022.