Energy-aware Coflow Scheduling for Sustainable Workload Management

Sadiya Ahmad * Flavio Esposito * Estefanía Coronado †
* Department of Computer Science, Saint Louis University, USA; Email: {first.last}@slu.edu
† I2CAT Foundation, Barcelona, Spain; Email: estefania.coronado@i2cat.net

***Abstract*—Handling High-Performance Computing (HPC) workflows often requires the orchestration of a collection of parallel flows. Traditional techniques to optimize flow-level metrics do not perform well in optimizing such collections because the network is usually agnostic to application requirements. A Coflow is a recently proposed abstraction that created new opportunities in network scheduling for datacenter networks. However, recent work on coflow scheduling has focused on merely two objectives: decreasing communication time of data-intensive jobs and guaranteeing predictable communication time. In this paper, we take a step further and propose some initial results towards the design of heuristics that optimize also the energy consumption of a data center that hosts HPC jobs. To this aim, we built and released an energy-aware coflow scheduling simulator to the community that helps analyze the tradeoff between energy efficiency and coflow completion time. We also propose two scheduling algorithms that consider coflow completion time, CPU utilization, and energy consumption efficiency. Our initial results using the simulator clarify how each policy should be tuned to the application needs and the computational resources available.**

I. Introduction

The deployment of resource-intensive applications leads to a growing demand for high-performance computing (HPC) infrastructures, accompanied by increased energy consumption [1]. In response to this problem in communication and computation, most data centers currently implement the MapReduce model [2], [3]. MapReduce is a programming technique that facilitates concurrent processing by splitting vast volumes of data into several chunks to be processed on distributed servers. The model involves mainly two ordered processes: (i) the mappers, which take the input data, split it and build a scheduling queue list, and (ii) the reducers, which perform a specific summary operation on the assigned sub-chunks. Despite the increased performance, the data transfer between mappers and reducers is still a costly process.

Coflow scheduling builds on the MapReduce model and considers that applications are composed of a set of flows, finishing after all of them have been adequately processed. The precursors of this novel technique defined a coflow as a networking abstraction to express the communication requirements of jobs in prevalent data-parallel programming paradigms [4]. More specifically, a coflow, c, is composed of a set of individual flows, f_i, with a collective goal and whose characteristics are exploited to optimize completion time. The architecture of mappers and reducers in coflow scheduling can be modeled as a switch with m ingress ports and m egress ports. The completion time of a coflow, c, depends on the flow with the maximum ending time, $max\, end(f_i)$.

Several authors have aimed to improve coflow scheduling within the networking and the theory communities [5]. Within the vast body of literature, Varys [6] and Aalo [7] can be highlighted as two of the most significant contributions. Moreover, in many clusters, the scheduling algorithm of choice remains the Shortest Job First (SJF) due to its simplicity. However, most of the works revolve around reducing the average Coflow Completion Time (CCT). Despite being a crucial metric for cost and environmental damage minimization, none of these algorithms consider energy consumption. This energy factor has been proved to be dependent on network and reducers utilization [8]. Consequently, the efficient design of scheduling algorithms is not only able to increase performance but also to significantly reduce energy consumption.

In this context, this paper proposes some initial results towards aimed at propelling coflow scheduling algorithm that are energy-aware, that is, able to assign coflows to an infrastructure while considering energy consumption. In particular, the contribution of our work is three-fold.

- We present some initial design of a modular network simulator able to handle coflow requests based on various parameters. Unlike others, such as Mininet and GNS3, the network model of our simulator is based on high-performance computational clusters and enables throughput and energy monitoring. The code is available for the research community at [9].
- As a proof-of-concerpt, we propose two basic scheduling algorithms that consider not only CCT minimization and CPU utilization but also energy consumption efficiency.
- We compare the proposed solutions with a set of coflow scheduling strategies, showing up how even a simple approach can save up to 70% energy reduction at the price of a negligible increase in CPU time.

The remainder of this paper is structured as follows. Sec. II discusses the related work. Sec. III and Sec. IV introduce the energy-efficient coflow scheduler and the network simulator introduced in this work, respectively. Finally, Sec. V discusses the performance evaluation and Sec. VI concludes our paper.

II. Related Work

Coflow scheduling takes inspiration from the classical Network Interface Card (NIC) packet scheduling [10], where traffic shaping can be done with a single queue. In coflow scheduling, however, the main objective resides in minimizing a metric called Coflow Completion Time (CCT), using information such as the size and length of the coflows. Depending on whether the coflow provides this information, algorithms are categorized as information-aware and information-agnostic.

Several algorithms have aimed to optimize different aspects of coflow scheduling. Two significant examples are Varys and Aalo. Varys [6] presents an information-aware system that enables data-intensive frameworks to use coflows while maintaining high network utilization and guaranteeing starvation freedom to minimize CCT. This algorithm uses an agent (called master) that coordinates mappers and reducers and monitors the network usage during scheduling to perform the reducers' selection optimally. By contrast, Aalo [7] requires no prior knowledge of the coflow parameters (i.e., information-agnostic) and defines priority queues based on the amount of data already sent by the reducers in such a manner that it can handle cluster dynamics while minimizing completion times. Similarly, the authors of [11] propose sample-based online learning of the coflow size for information-agnostic scenarios. In the algorithm presented in [12] each coflow is assigned a predicate and a rank, and schedules at each round the smallest ranked eligible element. Priority scheduling is also studied in [13], which assumes that each mapper is a host that divides its processing time into epochs. Then, in each epoch, a subset of unfinished coflows are selected and ordered using a simple greedy algorithm differently at each host. Due to the performance offered by its simplicity, SJF is still widely used in data centers. It schedules the pilot flows of each coflow and uses their measured size to estimate the size of the coflow. Though scheduling pilot flows of a newly arriving coflow consumes port bandwidth, which can delay other coflows with already estimated sizes, it remains less so when compared to the multi-queue-based approaches.

In contrast to all these sound solutions, we take an energy-aware approach, noting that with the increasing demands of machine learning jobs, the energy consumption cannot be ignored.

III. Energy-aware Coflow Scheduler

A. Network Model

The data-centered network model adopted in this work is a bipartite graph with mappers at the origin of the data flows and reducers at the destination. The graph comprises M mappers and R reducers. The only constraint on this graph is that the reducers should outnumber the mappers. The structure of the network model is depicted in Fig. 1, where we can see a one-to-many relationship from mappers to reducers. This relationship defines how the jobs from a mapper can be sent over to any number of reducers. We define a job as a sequence of coflows to be scheduled in the same batch (see Figure 2).

Given a set of mappers $M = (M_1, M_2, M_3, \ldots, M_n)$ and a set of reducers $R = (R_1, R_2, R_3, \ldots R_n)$ with coflows $C_1, C_2, \ldots, C_n$, a coflow mapping can be defined as the mapping of M onto the set of R, such that each node is mapped onto all nodes of R. Formally, the mapping is a function $f\colon M \rightarrow |R|^M$.

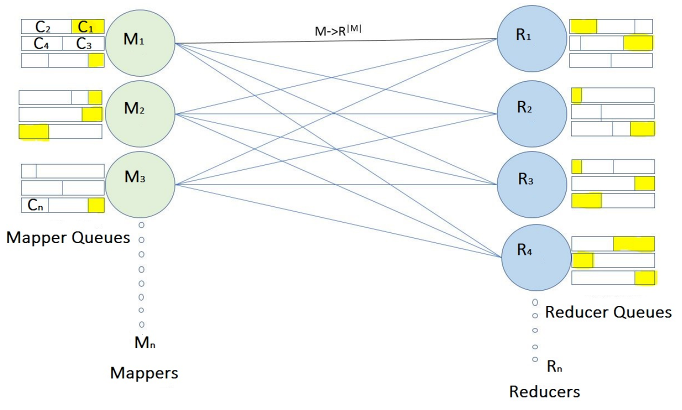

Fig. 1. Mapper-reduced bipartite graph of the adopted network model. Each mapper and reducer have a queue.

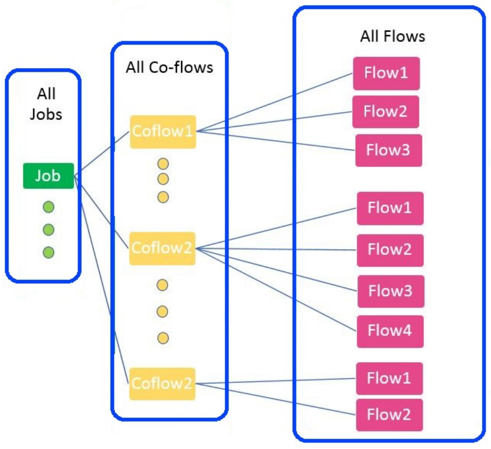

Fig. 2. Job - Coflow structure adopted in the simulator. We define a job as a collection of coflows to be scheduled.

B. Scheduler Design

The proposed scheduler is based on the MapReduce model; we assume the need to handle n user-defined jobs, following the structure presented in Figure 2. In particular, each mapper consists of three priority queues whose length can be modified. When a coflow arrives, it is assigned to a mapper in a round-robin fashion. Upon receiving all the coflows, each one is assigned to a reducer. The reducer keeps track of the coflows it receives. Each reducer has its own capacity, which is calculated at the beginning of each simulation and is predefined by the CPU it uses. Similarly, each CPU is characterized by offering a particular capacity (expressed in GigaFlops) and certain efficiency (expressed in GigaFlops/W).

Although a set of queues is modeled at the mappers, note that no scheduling decision is made at the reducers. Because of this, the number of coflows arriving at the reducers is often greater than the number of reducers $|M| > |R|$. Therefore, some coflows may remain in the mapper waiting for other coflows to finish and free up the reducers. This cycle continues until all the coflows in the mappers are assigned to the reducers. Due to resource limitations, not all produced coflows are assigned to the mappers, and the remaining coflows are reported as incomplete.

C. Proposed Energy Consumption Efficient (ECE) Methods

Taking as a reference the network model introduced in this section, we have proposed two scheduling algorithms aiming at improving the energy consumption during the coflow scheduling phase.

- **Energy Consumption Efficient (ECE)**. In this algorithm, the coflow with the greatest size, s_c, among all the mappers is assigned to the most efficient reducer, using the $argmax$ function. The most efficient reducer is the one whose CPU has the highest ratio GigaFlops/Watts, expressed as E_r. This enables an energy-efficient processing of the greatest coflow. ECE allows predicting the impact of its decision on the energy consumed by the CPUs of the reducers.
- **Energy Consumption Efficient-2**. This algorithm selects the coflow with the greatest size, s_c among all the mappers and assigns it to the reducer with the highest computational power. In other words, the method assigns the reducer whose CPU is characterized by the highest number of GigaFlops, P_r. This implementation intends to reduce computational time while saving energy during the map-reduce jobs. The increased performance of this variation of ECE highly depends on whether an efficient processor also consumes less energy.

Note that both algorithms differ in the objective sought on the reducer, r, selection (i.e., highest efficiency vs. highest computational power). Mainly differentiated by this objective, the pseudocode of the proposed ECE and ECE2 methods is presented in Algorithm 1.

Algorithm 1 Energy Consumption Efficient Algorithm

```
Require: Mappers have queues and queues have coflows
  for mapper $m$ in $M$ do
    Pick coflow $c_m$ from the queue of $m$
    Calculate $s_{c_m}$, and flow sizes of the coflow, $s_{f_{c,m}}$
    $list_s \xleftarrow{add} s_{f_{c,m}}$
  end for
  $next_c = argmax(list_s)$
  if objective == highest efficiency then
    $argmax(E_r) \xleftarrow{add} argmax(next_c)$
  else if objective == highest power then
    $argmax(P_r) \xleftarrow{add} argmax(next_c)$
  end if
```

IV. Energy-Aware Coflow Simulator Design

This section discuses the architecture of the simulator introduced in this paper, which allows the testing and validation of several coflow scheduling algorithms.

A. Data Flow Description

The simulator architecture is detailed in Figure 3. In particular, our design considers the capabilities of mappers, i.e., connection map and mapper queues, and reducers, e.g., free memory, CPU capability, etc. We implemented a set of scheduling algorithms at the reducer. In addition to the two ECE algorithms presented in the previous subsection, the simulator includes other basic scheduling policies, e.g., (i) First-Input-First-Output (FIFO), which assigns to the reducer the coflow that entered first in the mapper's queue; and (ii) Short Job First (SJF), which takes first the shortest job that is composed of coflows. The engine and the graph are generated based on the configuration of the algorithm executed.

The simulation is initialized through a framework that calls upon other software elements. More specifically, the simulator takes the input from the user regarding the particular scheduling algorithms and their configuration. This user-defined data is stored in a `Config.prop` file. From such configuration, the simulator includes a server and client application to capture the properties of each reducer, creating the bipartite network graph of mappers and reducers and the list of jobs and coflows. Finally, it runs the mapper and reducer scheduling algorithms based on the specific scheduler configured policies. The data flow of the simulator is depicted in Figure 4.

Our programming environment also provides a set of metrics to evaluate the performance of the algorithms. Such metrics include job completion time records, average waiting time, average completion time, and number of incomplete jobs.

The software was designed to allow ease of use and has been implemented in Java. We use the Factory pattern as a design principle with different approaches. Our implementation uses the data to object (dto) and design by contract (dbc) paradigms.

B. Coflow parameters

In this simulator, a coflow is modeled as an object with several attributes. Our model enables an easy evaluation of the various attributes that affect the scheduling performance. Such attributes are defined as follows:

- **Length**. The size gives it in bytes of the largest flow in the coflow.
- **Width**. It is defined by the number of parallel flows in the coflow.
- **Size**. It is calculated as the sum expressed in bytes of all the flows in the coflow.
- **Skew**. It refers to the coefficient of variation of the flows in the coflow in terms of their size.

We characterize the size of flows from sample data. Before delving into the size of the flows, we can quickly determine the width of a coflow, which is only possible if there is no time-lapse between the assignment of two flows.

V. Evaluation and Analysis

In this section, we evaluate the energy-efficient algorithms and compare their performance to the FIFO and SJF algorithms in terms of CPU time and energy consumption.

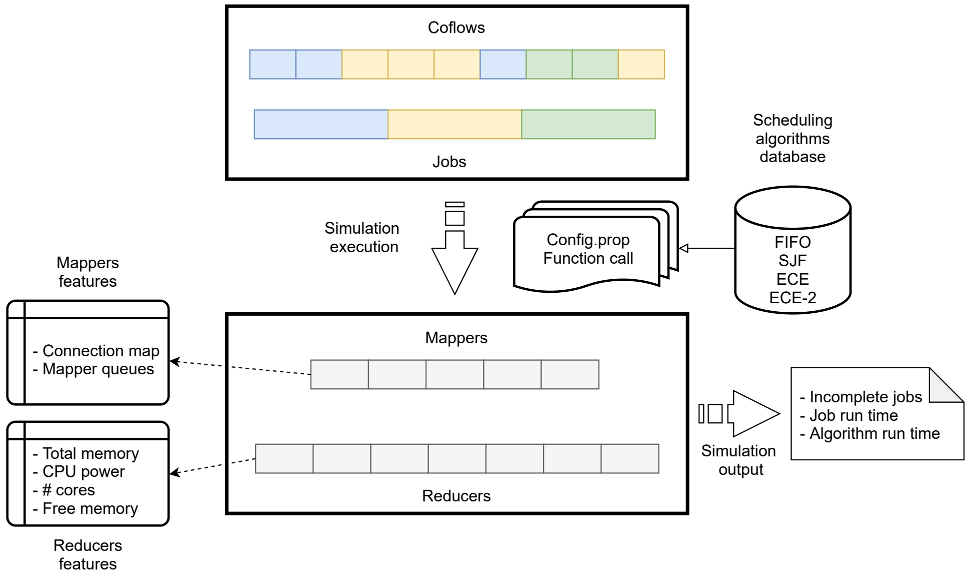

Fig. 3. High-level architecture of the programming framework.

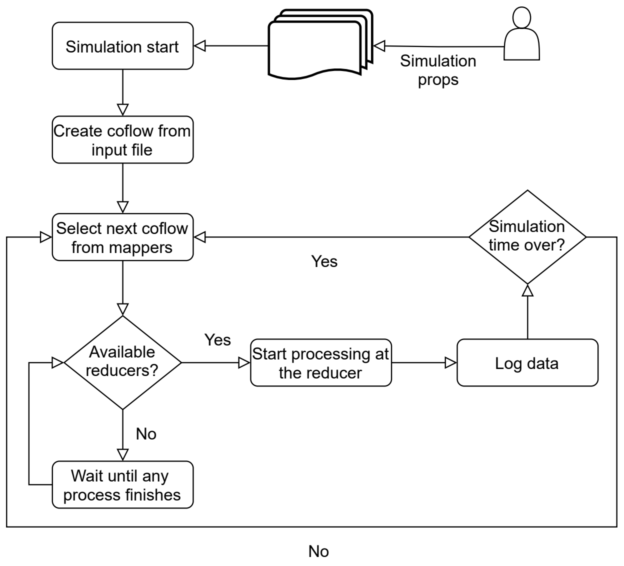
Simulation start
Simulation props
Create coflow from input file
Select next coflow from mappers
Yes
Simulation time over?
Available reducers?
Yes
Start processing at the reducer
Log data
No
Wait until any process finishes
No

Fig. 4. Flow chart of the energy-aware coflow scheduling simulator.

A. Evaluation Scenario

We take as a reference the computational power and energy consumption data from common real-world server-level processors (see Table I). We choose processors with very different performance metrics to investigate the impact of scheduling algorithms on evaluation criteria. Note that the CPU configurations vary widely between various clusters, and that the results of a scheduling algorithm may therefore differ if different server infrastructures were chosen, given its dependency on both the cluster configuration and the data to be processed.

The reference scenario studied comprises 25 coflows and 730 flows. The user-defined properties have been set to 25 coflows, two mappers and three reducers. Moreover, the mappers have been configured with three queues with a length of 5. Three reducers are randomly chosen in the experiments, but the same reducers are used for every algorithm to avoid the discrepancy in results. CPU calculations are made beforehand, and all tasks are provided with enough CPU capacity. It is also assumed that there is no idle time on any reducer between task processing. Simulation results are shown with 95% confidence intervals considering 15 runs for each algorithm.

TABLE I
CPU SPECIFICATIONS.

CPU	# Cores	Clock (GHz)	Power (W)	GFLOPS	Efficiency (GFLOPS/W)
Ampere Altra	80	3.30	715.87	2112	2.95
Intel 8280 SP	28	2.70	118.13	604.80	5.12
Intel Xeon SP 8276	28	2.20	47.83	492.80	10.30

B. Evaluation Results

Taking as a reference the scenario described before, Fig. 5 depicts the energy consumed by each algorithm (i.e., ECE, ECE2, FIFO and SJF). As we can see, FIFO and SJF are not energy-aware policies and therefore consume more energy in their scheduling tasks. FIFO is by far the worst energy management algorithm, almost tripling the energy consumption with respect to the other scheduling algorithms. The results arise from the fact that the waiting time for FIFO is greater than the one of SJF, which makes FIFO the least attractive option for coflow scheduling in data centers. However, SJF performs worse than the ECE algorithms. This is because SJF takes the smallest job under consideration instead of just the smallest coflow. By contrast, the ECE and ECE2 algorithms,

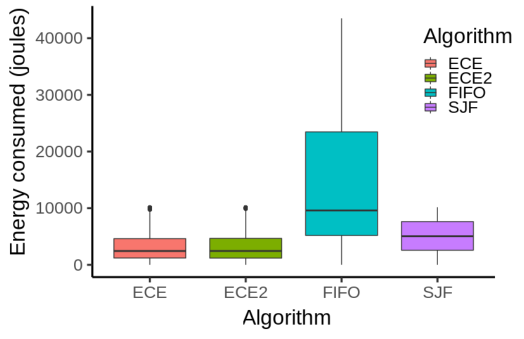

Fig. 5. Energy consumption evaluation and variation.

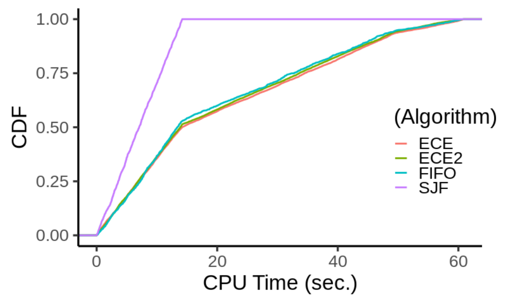

Fig. 7. Cumulative distribution function for CPU time.

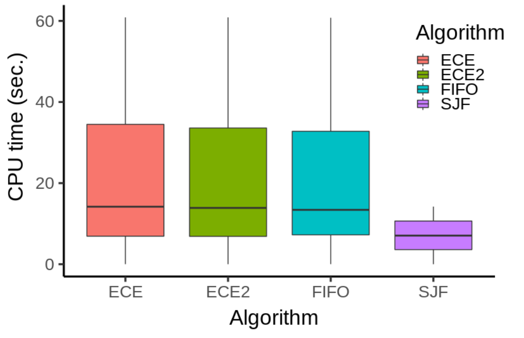

Fig. 6. Algorithm processing time comparison.

introduced to reduce the energy consumed, can improve by a factor of 50% and 70% with respect to SJF and FIFO.

The energy consumption of the ECE algorithms is lowered at the price of a negligible increase in CPU time. We show this result in Figure 6 (processing time at the reducer). As we can see, the CPU time of ECE, ECE2, and FIFO present minimal differences, while SJF provides a significantly shorter CPU time. These results are further studied by the cumulative distribution function of the CPU time in Fig. 7. In particular, it reveals that the most drastic consumption change occurs for SJF while ECE, ECE2, and FIFO show a gradual increase over time. This is because SJF sends all the coflows on the lower size spectrum to any reducer without considering their efficiency or energy consumption. Observing this change is useful when data centers use a combination of scheduling algorithms, e.g., the process could start with the SJF policy and over time be transferred to any ECE version as the benefits of using SJF get reduced.

VI. Conclusion

Power saving in data centers have been proved beneficial from the economic and the environmental standpoints. Although coflow scheduling marked a turning point for increased performance, energy-aware scheduling is still a pending aspect to be analyzed in the literature. In this paper, we have presented an initial version of a network simulator enabling performance and energy assessment of coflow scheduling algorithms. Moreover, we have proposed two energy efficient algorithms aiming to reduce the energy consumed based on efficiency and power of the CPUs involved. Finally, we have used the simulator introduced to compare our algorithms with state-of-the-art policies, showing in the initial results that with a slightly higher CPU time the algorithms are able to reduce the energy consumed by an average of 50%.

Acknowledgement

This work has been partially supported by NSF Awards CNS-1836906 and CNS-1908574.

References

[1] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. K. Steyer, E. Masanet, N. Horner, I. Azevedo, and W. Lintner, "United States Data Center Energy Usage Report," 2016, acessed on 02.06.2021. [Online]. Available: https://www.osti.gov/servlets/purl/1372902/

[2] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[3] Apache Software Foundation, "Hadoop," accessed on: 2021-06-06. [Online]. Available: https://hadoop.apache.org

[4] M. Chowdhury and I. Stoica, "Coflow: A Networking Abstraction for Cluster Applications," in *Proc. of ACM HotNets*, Redmond, WA, USA, 2012.

[5] S. Wang, J. Zhang, T. Huang, J. Liu, T. Pan, and Y. Liu, "A Survey of Coflow Scheduling Schemes for Data Center Networks," *IEEE Commun. Mag*, vol. 56, no. 6, pp. 179–185, 2018.

[6] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient Coflow Scheduling with Varys," in *Proc. of ACM SIGCOMM*, Chicago, IL, USA, 2014.

[7] M. Chowdhury and I. Stoica, "Efficient Coflow Scheduling Without Prior Knowledge," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 393–406, 2015.

[8] G. Wen, J. Hong, C. Xu, P. Balaji, S. Feng, and P. Jiang, "Energy-aware hierarchical scheduling of applications in large scale data centers," in *Proc. of IEEE CSC*, Hong Kong, China, 2011.

[9] Sadiya Ahmad, "Energy-aware coflow scheduling simulator," accessed on: 2021-06-06. [Online]. Available: https://github.com/sadiya0312/NWSim/.

[10] B. Stephens, A. Akella, and M. Swift, "Loom: Flexible and Efficient NIC Packet Scheduling," in *Proc. of USENIX NSDI*, Boston, MA, 2019.

[11] A. Jajoo, Y. C. Hu, and X. Lin, "Your Coflow has Many Flows: Sampling them for Fun and Speed," in *Proc. of USENIX ATC*, Renton, WA, USA, 2019.

[12] V. Shrivastav, "Fast, scalable, and programmable packet scheduler in hardware," in *Proc. of ACM SIGCOMM*, Beijing, China, 2019.

[13] S. Agarwal, S. Rajakrishnan, A.Narayan, R. Agarwal, D.Shmoys, and A.Vahdat, "Sincronia: Near-Optimal Network Design for Coflows," in *Proc. of ACM SIGCOMM*, Budapest, Hungary, 2018.