Sunflow: Efficient Optical Circuit Scheduling for Coflows

Xin Sunny Huang Xiaoye Steven Sun T. S. Eugene Ng

Rice University

ABSTRACT

Optical Circuit Switches (OCS) are increasingly used in cluster networks due to their data rate, energy and longevity advantages over electrical packet switches. Concurrently, an emerging crucial requirement for modern data-parallel clusters is to achieve high application-level communication efficiency when servicing structured traffic flows (a.k.a. Coflows) from distributed data processing applications. This paper presents the first OCS scheduling algorithm called Sunflow that addresses this requirement.

Preemption decisions are the key to any OCS scheduling algorithm. Sunflow makes preemption decisions at two levels. First, at the intra-Coflow level, Sunflow does not allow subflows within a Coflow to preempt each other. We prove that the performance of this strategy is within a factor-of-two to the optimal. We further demonstrate that under realistic traffic, performance of Sunflow is on average within 1.03× to optimal. Second, at the inter-Coflow level, Sunflow provides a framework for flexible preemption policies to support diverse usage scenarios. In the specific case of the shortest-Coflow-first policy, we find that Coflows on average finish just as fast in a Sunflow-scheduled optical circuit switched network as in a comparable packet switched network employing the state-of-the-art Coflow scheduler.

CCS Concepts

•Networks → Network control algorithms; Network resources allocation; Data center networks;

CoNEXT '16, December 12 - 15, 2016, Irvine, CA, USA

 ISBN 978-1-4503-4292-6/16/12…$15.00

DOI: http://dx.doi.org/10.1145/2999572.2999592

Keywords

Circuit scheduling algorithm; Optical circuit switching; Structured traffic flow; Cluster network; Data center;

1. INTRODUCTION

Optical Circuit Switches (OCS) are increasingly used in cluster networks [35, 16, 36, 15, 30, 25, 34, 26] due to several compelling advantages over electrical packet switches. First, the OCS is future data-rate proof. Because it simply transmits optical signal at the physical layer and does not process packets, it is agnostic to data rate. Thus, an OCS needs no upgrade as link speed increases. Second, OCS, a passive physical layer device, consumes an order of magnitude less power than an electrical packet switch. An OCS consumes as little as 150 mW per port while an electrical packet switch consumes 4 W per port. Third, OCS also have very large mean time to failure (up to 30 years for a commercial 3D-MEMS switch [17, 9]), and its benefits can be enjoyed for years to come. Concurrently, structured traffic flows called Coflows [10] from distributed data processing applications are becoming an increasingly dominant workload in data-parallel clusters; thus, an emerging crucial requirement for data-parallel cluster networks is to achieve high application-level communication efficiency when servicing Coflows [12, 11, 39, 38].

In this paper, we set out to explore the question: how to best service Coflows from distributed data processing jobs in an optical circuit switched network? Previous work has observed that OCS is well suited when the traffic is sparse [35]. However, many Coflows are dense with heavy many-to-many subflows. Therefore, how well circuit switching can serve Coflow traffic remains an open question.

Preemption decisions are the key to any OCS scheduling algorithm: good decisions lead to high circuit utilization, while poor decisions lead to performance problems like head-of-line blocking. We start by exploring how well state-of-the-art circuit scheduling algorithms in the data center context [35, 16, 30, 34, 26] can serve Coflow. Unfortunately, we find their performance disappointing for several reasons. First, they require ex-

cessive circuit preemptions, resulting in large latency for Coflow due to heavy switching overhead. Second, they lack the ability to explicitly handle multiple Coflows or meet individual Coflow's performance requirement.

We observe that these algorithms stem from the *all-stop* circuit switch model (§ 2.1), a conventional model commonly used from [21, 7, 28, 33] to [16, 30, 34, 26], spanning decades. In general, an *all-stop* switch requires more preemptions to reduce circuit idleness (§ 3.1.2). However, the modern optical switch is in fact less stringent than required by the *all-stop* model. Under a more accurate, yet less constrained switch model, i.e. the *not-all-stop* switch model (§ 2.1), circuits can be better utilized with fewer preemptions.

Based on these observations, we design Sunflow, the first circuit scheduling algorithm that is efficient for scheduling Coflows. Sunflow makes preemption decisions at two levels. First, at the intra-Coflow level, Sunflow does not allow subflows within a Coflow to preempt each other. This non-preemptive strategy is starvation-free because subflows within a Coflow share the same performance objective. We prove that the performance of this strategy is within a factor-of-two to the optimal, while the other circuit scheduling algorithms have no known performance guarantee. We further demonstrate that under realistic traffic, performance of Sunflow is on average within 1.03× to optimal. Second, at the inter-Coflow level, Sunflow provides a framework for flexible preemption policies to support diverse usage scenarios (e.g. policies that govern how production vs. non-production traffic is handled). In the specific case of the shortest-Coflow-first policy, we find that Coflows on average finish just as fast in a Sunflow-scheduled optical circuit switched network as in a comparable packet switched network employing Varys [12] and Aalo [11], the state-of-the-art Coflow schedulers. With Sunflow's ability to achieve Coflow performance in a circuit switched network approaching that of a packet switched network, an OCS becomes a viable component in a modern data-parallel cluster network especially when the data rate, energy, and longevity advantages of the OCS are simultaneously considered.

2. PROBLEM FORMULATION

2.1 Network Model

We start by describing the conceptual model used to study the scheduling problem. In our network model, the network fabric is abstracted as one non-blocking N-port switch with link bandwidth B. This model is attractive because of its simplicity, yet it is practical because topology designs such as Fat-tree or Clos [5, 20] enable building a network with full bisection bandwidth. We assume the switch ports are connected to Top-of-Rack (ToR) switches, and each ToR switch is connected to a group of machines.

Within this network model, we will investigate two types of switches to study the performance of circuit switched networks and packet switched networks.

Optical Circuit Switch: In the circuit switched network, the switch is an optical circuit switch. An optical switch has *port constraint*: an input (output) port can set up at most one circuit to an output (input) port at a time. In other words, no input (output) port may connect to multiple output (input) ports. Each circuit can be reconfigured with the cost of a fixed time delay δ. This means during a period of δ to set up one circuit, communication stops on the input (output) ports affected, including ports on circuits to be set up, as well as ports on circuits to be torn down. However, circuits unchanged are not impacted and may keep serving traffic. This circuit switch model, referred to as the *not-all-stop* model hereafter, is *a more accurate, yet less constrained model for the optical switch* (off-the-shelf product examples include [17, 9, 29]). However, its counterpart, i.e. the *all-stop* model, would unnecessarily assume communication stops on *all* optical circuits during reconfiguration, but it is commonly used in previous studies [15, 30, 25, 34, 26].

Flows are buffered at the sender machines. Each input port of the switch is to serve flows from sender machines to various output ports. The flows are aggregated and organized into logical virtual output queues (VOQs) associated with each input port. At any time for an input port, at most one VOQ is served, and it is served with the full link bandwidth B. The ordering of flows in a VOQ is to be determined.

Electrical Packet Switch: In the packet switched network, the switch is an electrical packet switch. Denote $b_{i,j}$ as the bandwidth allocated for the link from input port $in.i$ to output port $out.j$. At any time, multiple VOQs may be served simultaneously, but the bandwidth constraints must be satisfied with $\forall j : \sum_{i=1}^{N} b_{i,j} \leq B$ and $\forall i : \sum_{j=1}^{N} b_{i,j} \leq B$.

We focus on the pure circuit (packet) switch models, which is the foundation to different scheduling problem variants, such as scheduling on hybrid circuit/packet networks, where traffic can be filtered and offloaded to different parallel networks [35, 16, 26].

2.2 Traffic Model

In this paper, we focus on a traffic model called Coflow [10], which represents a collection of independent flows that share a common performance goal. A Coflow is defined by the endpoints and size in byte of each flow within the Coflow. A Coflow can represent any communication pattern, such as many-to-many, one-to-many, many-to-one and one-to-one, etc. The traffic demand of a Coflow can be expressed in a demand matrix D, where each element $d_{i,j} \in D$ indicates flow $f_{i,j}$ transfers $d_{i,j}$ amount of data from input

Symbol	Definition
B	link bandwidth
N	switch port count
δ	circuit switching delay
$in.i$, $out.j$	i-th input port, j-th output port
$[in.i, out.j]$	a circuit from $in.i$ to $out.j$
$\|C\|$	Number of subflows in a Coflow C
$p_{i,j}$	data processing time on $[in.i, out.j]$ in a Coflow ($p^k_{i,j}$ for k-th Coflow)
T^c_L, T^p_L	CCT lower bound in circuit or packet switching network, respectively.

Table 1: Notations

port $in.i$ to output port $out.j$. Denote $|C|$ as the number of subflows in a Coflow C, and thus $|C|$ is the number of non-zero entries in D. The network scheduler is to decide how individual flows of Coflows are serviced.

2.3 Scheduling Objective

At the intra-Coflow level, the objective is to minimize the Coflow Completion Time (CCT), which is the duration to finish all flows in a Coflow, so as to speed up application level performance. For a Coflow C, denote its arrival time as t^{Arr}, and the finish time of the flow $f_{i,j} \in C$ as $t^F_{i,j}$. Hence CCT is defined as $\max_{f_{i,j} \in C}(t^F_{i,j} - t^{\text{Arr}})$. This objective is also used by all previous work on Coflow scheduling.

In a circuit switched network, the circuit scheduling algorithm must decide *when* to set up *which circuit* to serve the Coflow traffic. Generally, a circuit may be preempted before the demand on the circuit is finished, but preemption has a penalty of δ to reestablish the circuit for the remaining demand. When the preemption penalty is zero, i.e. $\delta = 0$, the problem can be solved optimally with the classic BvN algorithm [31]. When preemption penalty $= \infty$, the problem reduces to the non-preemptive open-shop problem to minimize work span [18], where each job (input port $in.i$) requires processing time ($d_{i,j}/B$) on a specific machine (output port $out.j$). The open-shop problem is *non-preemptive*, which means a machine (output port) would always finish processing a job (flow from one input port) before moving to another job (flow from another input port). The open-shop problem is NP-hard [18].

At the inter-Coflow level, the objective will depend on the resource management policy employed by the system. Coflows may come from different applications that have different importance to the users. If the resource management policy is to give priority service to more important Coflows, then the scheduling objective is to minimize the time when more important Coflows are blocked by less important ones. On the other hand, if the resource management policy is to prioritize smaller Coflows over larger ones, then the objective is to minimize the average CCT.

2.4 CCT Lower Bounds

No efficient optimal algorithm is known for intra-Coflow circuit scheduling in the general case where $0 < \delta < \infty$. Moreover, the optimal CCT may also depend on δ. As a result, we derive CCT lower bounds for the general case.

Note that our CCT lower bound in the circuit switched network, T^c_L (Equation (4)), is more accurate for the optical switch compared with the one in a previous work [26], because their bound is based on the *all-stop* switch model, while our bound is based on the *not-all-stop* switch model.

To derive the CCT lower bounds, we begin by translating a Coflow traffic demand D from size to *data processing time* with

$$p_{i,j} = \frac{d_{i,j}}{B}, \tag{1}$$

where $p_{i,j}$ is the processing time required on the circuit $[in.i, out.j]$ incurred by the flow $f_{i,j}$ within the Coflow.

Packet-Switched Lower Bound: In the packet switched network, the CCT lower bound is the maximum time required to finish data transfer on all ports, and it is given by

$$T^p_L = \max\left(\max_i \sum_{j=1}^{N} p_{i,j}, \max_j \sum_{i=1}^{N} p_{i,j}\right) \tag{2}$$

Circuit-Switched Lower Bound: In the circuit switched network, each port must spend time to set up circuits and transfer data. Each flow incurs at least one circuit reconfiguration delay δ. Thus the *minimum* time required to transfer flow $f_{i,j}$ with data processing time $p_{i,j}$ from $in.i$ to $out.j$ is

$$t_{i,j} = \begin{cases} 0, & \text{if } p_{i,j} = 0 \\ p_{i,j} + \delta, & \text{if } p_{i,j} > 0 \end{cases} \tag{3}$$

Hence the CCT lower bound in the circuit switched network is

$$T^c_L = \max\left(\max_i \sum_{j=1}^{N} t_{i,j}, \max_j \sum_{i=1}^{N} t_{i,j}\right) \tag{4}$$

Note that T^p_L and T^c_L are the theoretical lower bounds for CCT. Thus both of them are independent of the scheduling policy. The optimal achievable CCT may be much larger than the lower bound. Nevertheless, we will compare algorithms' performance against these stringent lower bounds.

3. RELATED WORK

3.1 Intra-Coflow Scheduling

In a packet switched network, intra-Coflow scheduling for one Coflow demand can be optimally solved [31]. However, how to best schedule circuits for one Coflow demand remains an open problem.

Coflow	Circuit Switched		Packet Switched
Intra-	Preemptive	§ 3.1.1	§ 3.1
	Non-preemptive	§ 3.1.2	
Inter-	§ 3.2		§ 3.2

Table 2: Taxonomy of related work

3.1.1 Preemptive Circuit Scheduling

Researchers have seized the compelling benefits of OCS with a range of infrastructure designs in recent work [35, 16, 36, 15, 30, 25, 34]. These designs involving OCS all rely on the circuit scheduler to compute a series of circuit configurations, and their scheduling algorithms follow a common approach: 1) group requested traffic demand into *one* demand matrix, which amounts to one Coflow demand, and 2) based on the demand matrix, produce a sequence of *circuit assignments* $\{A_1, ..., A_m,...\}$, each associated with a *duration* of time $\{t_{A_1}, ..., t_{A_m},...\}$. Each A_i encodes a set of circuits. To respect the port constraints in the circuit switch, each A_i is a one-to-one matching between the input ports and output ports. The assignments are scheduled one by one so that A_m is active for t_{A_m}. These algorithms are *preemptive* in the sense that circuits in A_m can be *preempted* to allow for the next $A_{(m+1)}$. In many cases, most of the circuits in A_m are preempted, while in rare cases, circuits remain unchanged in two consecutive assignments. Further discussions on three representative algorithms using this approach are provided next. Later in this paper, we will show that Sunflow does not suffer from the problems associated with this approach.

Edmond used in [35, 16, 34] applies the maximum weighted matching algorithm [14] on the demand matrix and makes up *one* assignment from the matching, with the duration determined externally of the algorithm, which is typically fixed and on the order of hundreds of milliseconds. Edmond causes large delay for Coflow, as each assignment scheduled usually does not cover all requested demand of any specific Coflow.

TMS used in [15, 30] applies the classic BvN matrix decomposition algorithm [8]. To use BvN, TMS would 1) *pre-process* the demand matrix to the meet the input assumption of BvN and 2) *decompose* the pre-processed matrix into assignments and weights for each assignment, with durations determined proportionally by the weights. TMS also causes large delay for Coflow because the pre-processing step may heavily modify the original demand matrix, such that the scheduled circuits may poorly serve the original requested demand.

Solstice in [26] is similar to TMS, but has a better pre-processing and decomposition design, as well as a scheduling objective to cover all original requested demand within the minimum time. Particularly, Solstice may add dummy demand to the demand matrix in its pre-processing step. Solstice is shown to be more effective among the preemptive algorithms, but the number of reconfigurations is still a major cause of inefficiency.

These algorithms can be traced back to the classic crossbar switch scheduling problem, or the so-called Time Slot Assignment (TSA) problem. TSA is based on the *all-stop* switch model. Such an *all-stop* switch model has been commonly used in classic papers [21, 7, 28, 33] as well as recent papers [30, 34, 26]. They rely heavily on the techniques of matrix decomposition and schedule circuits in one assignment after another. Note that our problem differs from TSA because our problem is based on a different switch model, i.e. the *not-all-stop* model.

3.1.2 Non-preemptive Circuit Scheduling

Under the *all-stop* switch model, while many of the scheduling algorithms are preemptive, there are non-preemptive TSA designs [19, 23]. However, these designs are less favorable in general due to circuit idleness: *all* idle circuits have to wait for *any* transmitting circuits in the same assignment. Moreover, non-preemptive TSA designs are not efficient for the *not-all-stop* switch model, because non-preemptive TSA forces idle circuits to wait and occupy the ports needed to serve demand on other circuits.

Circuit scheduling for one Coflow demand is NP-hard when preemption penalty $= \infty$ (§ 2.3). In a general case where $0 < \delta < \infty$, the problem is already NP-hard [6] in a simplified two-machine case, and such simplification corresponds to a limitation in our problem that the network can only have two ports. A related problem for arbitrary δ under multiple machines is studied in the context of wavelength-division multiplexed (WDM) network [32, 13, 24], but the heuristics depend on rigid conditions (e.g. linear constraints, uniformity) on the input demand, which places impractical limitations on the Coflow demand in our problem.

Later in this paper, we will show that Sunflow provides performance guarantee in the general case with no constraints on δ, Coflow demand or number of ports.

3.2 Inter-Coflow Scheduling

To the best of our knowledge, there is no circuit scheduling algorithm designed for inter-Coflow scheduling. Existing circuit scheduling algorithms can only function on a single demand matrix. These algorithms would need to aggregate the demand from multiple Coflows as one generic demand and schedule without considering the structure of multiple Coflows. To the best of our knowledge, Sunflow is the first circuit scheduling algorithm that supports inter-Coflow scheduling.

In a packet switched network, inter-Coflow scheduling remains an open problem under discussion [12, 11, 31, 39, 27, 38]. Note that inter-Coflow scheduling in a circuit switched network where $\delta > 0$, is different from scheduling in a packet switched network that amounts to $\delta = 0$ in previous works. Particularly, *Varys* [12] and *Aalo* [11] are state-of-the-art packet-switched Coflow schedulers designed to minimize average CCT. Varys assumes all information of a Coflow (endpoints and sizes for all subflows) is known when the Coflow arrives, while

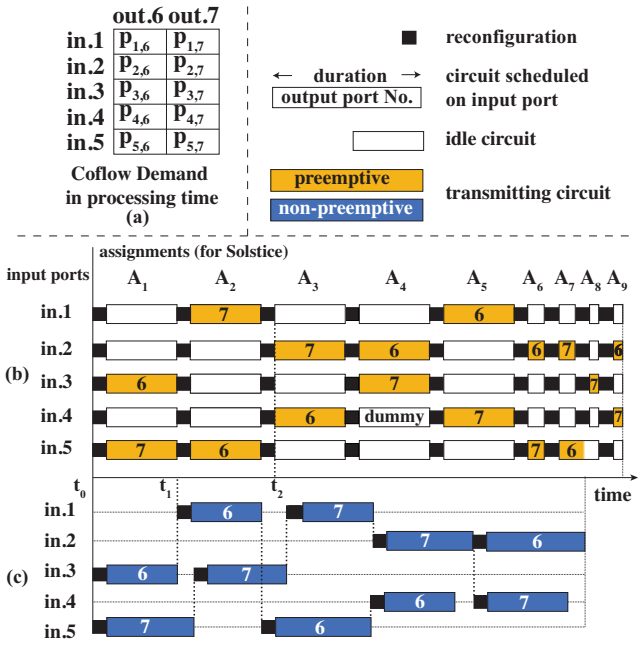

Figure 1: Intra-Coflow Circuit scheduling. (a) Coflow demand with input ports as rows, output ports as columns, and values as data processing time. (b) Preemptive scheduling by Solstice. (c) Non-preemptive scheduling by Sunflow. Circuits for input ports are laid horizontally on the time line. Circuit durations in (b) and (c) are proportional to $p_{i,j}$ in (a).

Aalo assumes known endpoints and *unknown* traffic sizes. In many cases, the Coflow information can be obtained from the application (e.g. the task scheduler can provide information for the shuffle traffic during a map-reduce job). We evaluate Sunflow with the same traffic assumption as Varys because accurate Coflow information removes the effects of traffic uncertainty. Other Coflow related operational issues, such as routing [39] and predicting Coflow volume [11] or structure [38] are out of the scope of this study.

4. SUNFLOW

4.1 Intra-Coflow Scheduling Design

Figure 1 illustrates the key difference between Sunflow and the most viable alternative scheduling algorithm, Solstice, at the intra-Coflow level. The scheduled circuits for different input ports are laid horizontally on the time line. For example, in Figure 1b, two circuits are set up at time t_2, and they are [*in*.2, *out*.7] and [*in*.4, *out*.6]. The circuit schedule should satisfy the Coflow demand in Figure 1a, which means for any input port *in.i*, the sum of the circuit durations for any output port *out.j* should be no less than $p_{i,j}$.

Solstice produces the schedule in Figure 1b. Like many of the existing circuit scheduling algorithms, Solstice schedules circuits with assignments, and in this example, it produces $\{A_1, ..., A_9\}$.

A new assignment may be scheduled when a circuit becomes idle. For example, in Figure 1b, A_1 finishes $p_{3,6}$ and [*in*.3, *out*.6] turns idle, so A_2 is scheduled, which preempts [*in*.5, *out*.7]. Note that Solstice schedules A_5, because one of the circuits in A_4 is actually created for a dummy demand added in the pre-processing step of Solstice (§ 3.1.1). Consequently, when Solstice considers the dummy demand is "finished" and certain circuit becomes "idle", Solstice schedules a new assignment A_5.

Preempted circuits with unfinished demand incur extra reconfiguration overhead to be re-established. For example, in Figure 1b, [*in*.5, *out*.7] is preempted by A_2, which incurs an extra δ to set up in A_6. Circuits unchanged in two consecutive assignments (such as [*in*.5, *out*.6] in A_7 and A_8) may stay active continuously. Unfortunately, in many cases, few circuits overlap in two consecutive assignments. Consequently, a circuit may need to pay *multiple* reconfiguration penalties to finish a Coflow demand.

In contrast, Sunflow produces a much simpler circuit schedule as shown in Figure 1c. Given a Coflow demand, reconfiguration is minimized for each circuit: a circuit with non-zero demand is set up only once and stays active until the demand is finished. Besides, circuits are interleaving with each other over time with no synchronized setup/teardown times. As shown, Sunflow achieves a smaller CCT than Solstice in the example.

Thus, a key property of Sunflow is that it is non-preemptive at the intra-Coflow level. This approach is supported by the following observations.

A Coflow's traffic volume is not suitable for preemptive intra-Coflow scheduling. The inefficiency of preemptions is less prominent when the data processing time $p_{i,j}$ is very large compared to the preemption overhead δ. However, a Coflow's traffic volume is generally smaller than previous work assumed of the demand volume because a Coflow is an application-level traffic unit, and it is not heavily aggregated. As a result, overhead associated with preemption becomes a significant factor.

Starvation is not a concern. Non-preemption at the intra-Coflow level is starvation-free, because all subflows in a Coflow share the same performance objective.

The optical switch is not all-stop. Under the *all-stop* switch model, non-preemption would be less favorable due to circuit idleness (see discussion in § 3.1.2). However, the optical switch [17, 9, 29] is not all-stop, circuits may be reconfigured individually, and therefore the side effect of non-preemption is small: while one circuit stays non-preempted and transmitting, other circuits are free to be reconfigured, so long as the port constraints are satisfied (§ 2.1). For example, in Figure 1c, the time-overlapping circuits [*in*.1, *out*.6] and [*in*.5, *out*.7] are not allowed in the *all-stop* switch model, because the reconfiguration of [*in*.1, *out*.6] will stop the transmission of [*in*.5, *out*.7]. As a result, the conven-

Algorithm 1 Sunflow

```
1: Global PRT[.]
2: procedure IntraCoflow(Coflow C)
3:     P = all (i, j, p_{i,j}) in C            ▷ Shuffle P if desired
4:     time t = 0
5:     while P not empty do
6:         for all (i, j, p_{i,j}) in P do
7:             p_{i,j} = MakeReservation(i, j, t, p_{i,j})
8:         end for
9:         Remove entries in P where p_{i,j} = 0
10:        Advance t to next circuit release time in PRT[.]
11:    end while
12: end procedure

13: procedure MakeReservation(int i, j, double t, p_{i,j})
14:    reservation length l = 0
15:    if both in.i and out.j are free at t then
16:        t_m = earliest next-reserv-time of in.i, out.j
               ▷ Needed by inter-Coflow scheduling (§ 4.2)
17:        max length l_m = t_m − t
18:        desired length l_d = δ + p_{i,j}
19:        actual length l = l_m < δ ? 0 : min(l_m, l_d)
20:        Reserve in.i, out.j during (t, t + l) in PRT[.]
21:    end if
22:    return l > 0 ? (l_d − l) : p_{i,j}   ▷ Remaining demand
23: end procedure

24: procedure InterCoflow(Coflows C)        ▷ Sorted C
25:    PRT[.] = ∅
26:    for all C in C do
27:        IntraCoflow(C)
28:    end for
29: end procedure
```

tional algorithms based on the *all-stop* model discard many optimization opportunities in an optical switch. In contrast, Sunflow exploits the additional flexibility in the optical switch and achieves higher efficiency.

4.1.1 Algorithm Details

Refer again to the example in Figure 1c. Initially, Sunflow starts at t_0 and begins considering each demand in the Coflow. Within the same Coflow, the ordering of demand to be considered can be arbitrary (Lemma 1). Without loss of generality, we assume Sunflow starts reserving circuits for $p_{3,6}$ and $p_{5,7}$. Then Sunflow advances to the first circuit release time (t_1 in Figure 1c), and makes another reservation for $p_{1,6}$. After each reservation, Sunflow updates the remaining Coflow demand. Then Sunflow advances to the next circuit release time, and continues to schedule new circuits for the remaining demand, until all demand is drained.

Sunflow uses a Port Reservation Table (PRT) data structure to record ports taken by each circuit. The PRT contains entries for all input and output ports, and circuits are scheduled with reservations associated with each port. Each reservation tells when a port is taken and released, as well as the peer port for a circuit. Scheduling a circuit implies making reservations on both the input and output port of the circuit. Figure 1c illustrates the entries for input ports in a PRT.

Algorithm 1 shows the pseudocode of Sunflow. Sunflow starts at t_0 on PRT, considers each flow demand (Line 6), and reserves circuits for the flow(s) if the port constraints are satisfied (Line 15). Note that the ordering of flows to be considered can be arbitrary, since Sunflow's optimality analysis holds for any ordering of scheduled circuits (Lemma 1). The length of a reservation accounts for both the circuit reconfiguration overhead δ and the data processing time for the flow (Line 18) to transmit at link rate B. Sunflow updates the PRT with new reservation(s), which correspondingly adds constraints on ports in the PRT. After each reservation, Sunflow updates the remaining Coflow demand by subtracting the amount of demand fulfilled by the reservation (Line 22).

Note that a flow transmits at the full link bandwidth B when its reserved circuit is set up. This is because in the circuit switched network, servicing a flow would block other flows that compete for the same input or output port. Therefore, by transmitting at B, one subflow may complete as soon as possible to allow other blocked subflows to proceed as soon as possible.

Sunflow always observes port constraints on PRT when scheduling new circuits so that existing circuit reservations on PRT are not preempted. As a result, Sunflow continues by advancing to the most recent time when a circuit is released (Line 10). At this time, the released circuit(s) remove constraints on at least one input port and at least one output port, so that Sunflow may reserve more circuits. Then Sunflow identifies potential reservations by considering the remaining demand. Sunflow keeps reserving circuits until all the demand in the Coflow is satisfied.

4.1.2 Optimality Analysis

Recall that non-preemptive intra-Coflow circuit scheduling is NP-hard (§ 3.1.2). The following lemma establishes that Sunflow achieves a CCT within a factor of 2 to the optimal CCT in a circuit switched network.

LEMMA 1. *Denote T_S as Sunflow CCT, T_O^c as the optimal CCT, both in a circuit switched network, then $T_S \le 2T_L^c$ and $T_S \le 2T_O^c$, for any B, any δ, any Coflow, and any ordering of scheduled circuits.*

PROOF. See Appendix. □

The following lemma bounds the gap between Sunflow's CCT and the optimal CCT in a packet switched network.

LEMMA 2. *Denote T_S as Sunflow CCT in a circuit switched network, T_O^p as the optimal CCT in a packet switched network, then $T_S \le 2(1+\alpha)T_L^p$ and $T_S \le 2(1+\alpha)T_O^p$, where $\alpha = \frac{\delta}{\min_{f_{i,j} \in C}(d_{i,j}/B)}$ for Coflow C.*

PROOF. See Appendix. □

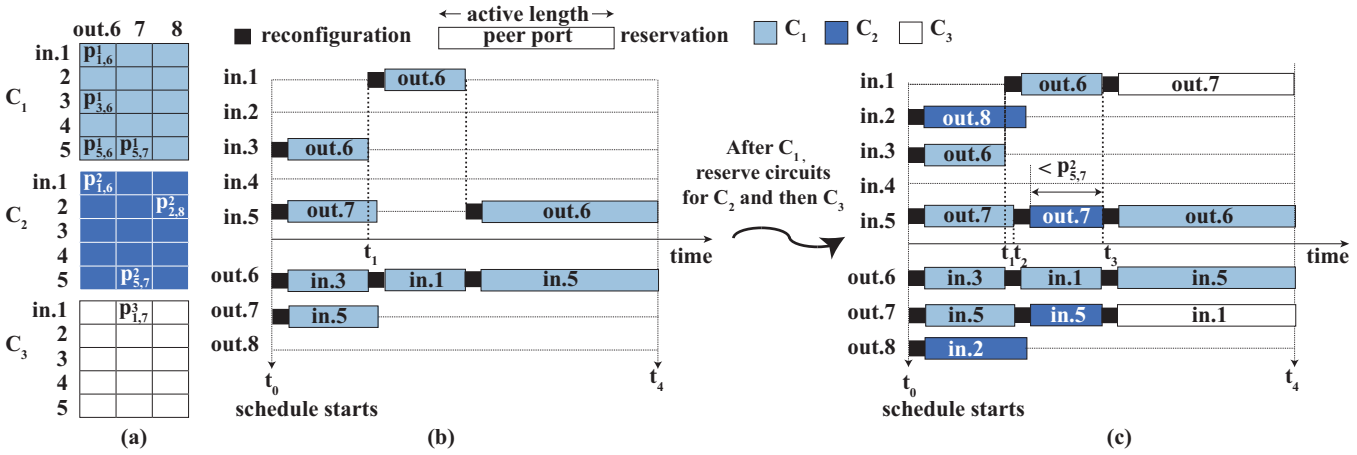

Figure 2: Inter-Coflow Circuit Scheduling. (a) Coflow demand with input ports as rows, output ports as columns, and values as data processing time. The desired order is C_1, C_2, C_3. (b) C_1 reservations. (c) Reservations for C_2 and C_3. Reservations are laid horizontally on the time line for each port. In (c), C_2 on $[in.5, out.7]$ should not block C_1 on $[in.5, out.6]$.

Edmond	TMS	Solstice	Sunflow
$O(N^3)$	$O(N^{4.5})$	$O(N^3 \log^2 N)$	$O(\|C\|^2)$

Table 3: Time complexity comparison

4.1.3 *Time Complexity*

For a single Coflow with $|C|$ subflows, Sunflow's time complexity is $O(|C|^2)$. In an extreme case where a Coflow covers all input and output ports, $|C| = N^2$. In contrast, the time complexity of other circuit schedulers solely depend on N, so that their running time could be very long even for a small Coflow. Besides, unlike these other algorithms which have no known performance bounds with respect to the optimal, Sunflow has a proven performance bound. Detailed analysis on time complexity is in the Appendix.

4.2 Inter-Coflow Scheduling Design

For inter-Coflow scheduling, when multiple Coflows compete for network resources, an important goal is to accommodate a variety of resource management policies employed by the system operator.

Flexible Management Policies. Different policies may naturally arise from different usage scenarios in data-parallel clusters. As a first example, a usage scenario may consist of privileged and regular users. The policy may require that Coflows requested by privileged users are preferred by the network over Coflows requested by regular users. As a second example, Coflows may be categorized into different classes to match the complex hierarchy for various jobs that generate Coflows. Coflows can be further subdivided into latency-sensitive versus latency-tolerant, and the network should serve them differently. As a third example, when Coflow-dependencies are introduced by multi-stage jobs [22, 37, 2, 1], the policy may require that later-staged Coflows yield to earlier-staged Coflows to avoid the potential creation of stragglers that unnecessarily prolong job runtime [11]. These example scenarios can potentially be mixed as well.

To leave as much flexibility as possible to the network operator, Sunflow provides a very simple framework that only requires the operator to translate the high-level policy into a priority ordering of Coflows. Sunflow then schedules Coflows to minimize the time when more prioritized Coflows are blocked by less prioritized ones. Sunflow also allows multiple Coflows to have the same priority, in which case these Coflows can be further sorted on arrival time and served individually, *or* these Coflows can be combined as one Coflow so that each constituent Coflow may have equal chance to be serviced. This should be a design choice made by the user. However, combining Coflows may come at the cost of a larger average CCT for the Coflows involved. In the simple case where the management policy is to serve shortest-Coflow-first, the Coflows may be ordered by their T_L^p.

To schedule circuits for Coflows, we use the intra-Coflow scheduling algorithm (IntraCoflow in Line 2 of Algorithm 1) as the building block. We apply IntraCoflow to each Coflow in the desired order (Line 26), so more prioritized Coflows complete faster without being blocked by the less prioritized ones. For example in Figure 2, reservation of $[in.5, out.7]$ for C_2 is shorter than desired so as *not* to block C_1 on $[in.5, out.6]$.

Avoiding Starvation Allowing Coflows to block the less prioritized ones may lead to starvation especially if a malicious attacker is involved. To avoid starvation, we introduce a simple and lightweight solution as follows.

We adopt a list of N fixed circuit assignments $\Phi = \{A_1, A_2, ..., A_N\}$, so that all N^2 circuits are covered in the N assignments. Given two tunable parameters

T and τ ($T \gg \tau > \delta$), we schedule circuits with re-occurring intervals of length $(T + \tau)$ over time. Each $(T + \tau)$-interval is divided into two smaller intervals of length T and τ, respectively. For the time T-interval, we apply InterCoflow to ensure the prioritized Coflows are not blocked by the less prioritized ones. For the time τ-interval, we schedule circuits in the assignment $A_k \in \Phi$. A_k is chosen in a round-robin manner so that A_k is scheduled if $A_{(k-1)}$ was scheduled in the last $(T+\tau)$-interval (k wraps around from N to 1). During the τ-interval, when a circuit is set up, subflows from all Coflows share the link bandwidth B on the circuit.

This solution ensures all Coflows receive non-zero service in every $N(T + \tau)$ interval. This is similar to the starvation-prevention design in previous work [12]. Avoiding starvation comes at the cost of reduced circuit utilization because some of the N circuits in one assignment may not have Coflow demand during τ. However, this approach is lightweight and incurs the minimal circuit switching overhead during τ.

5. EVALUATION

5.1 Settings

Workload: We use a realistic workload based on a one-hour Hive/MapReduce trace collected from a Facebook production cluster [4]. The trace contains more than 500 Coflows observed in a 150-port fabric with exact inter-arrival times. The traffic size in the original trace has been rounded to the nearest MB, as a result, many subflows in a Coflow have exactly the same size. We add $\pm 5\%$ perturbation of flow sizes to each flow to account for unequal flow sizes in real MapReduce jobs. The flow sizes are lower bounded at 1 MB (the smallest flow size in the trace) after perturbation, so that we have a factor of 4.5 upper bound of Sunflow's CCT/T_L^p ($\alpha = 1.25$, in Lemma 2) for all Coflows in the trace.

Simulator: We implement a trace-driven flow-level discrete-event simulator with various scheduling algorithms.[1] To evaluate the two levels of scheduling, we reproduce traffic from the trace in two ways. In intra-Coflow evaluation, a Coflow arrives only after the previous one is finished, so that only one Coflow is scheduled at any time and the Coflow arrival time in the original trace is ignored. In inter-Coflow evaluation, we perform detailed trace replay including arrival time.

Within each level of Coflow scheduling, we have evaluations for different settings. We evaluate B from 1 Gbps to 100 Gbps. For the circuit switched network, we evaluate δ from 100 ms to 10 μs. The default value of δ is 10 ms, which is typical of a 3D-MEMS optical switch that can scale to thousands of ports [3].

5.2 Baselines for Comparison

Lower bounds: We compare algorithm performance against the theoretical CCT lower bounds T_L^c and T_L^p.

[1]https://github.com/sunnyxhuang/sunflow

Category	O2O	O2M	M2O	M2M
Coflow%	23.4	9.9	40.1	26.6
Bytes%	0.005	0.024	0.028	99.943

Table 4: Coflow classified by sender-to-receiver ratio: One-to-One (O2O), One-to-Many (O2M), Many-to-One (M2O), and Many-to-Many (M2M).

Solstice: In the intra-Coflow evaluation, we also compare against the existing state-of-the-art circuit scheduling algorithm Solstice [26], whose performance is significantly better than TMS and Edmond for intra-Coflow scheduling. Specifically, in our experience, on average, Solstice services a Coflow more than 2× faster than TMS and more than 6× faster than Edmond.

Varys and Aalo: In the inter-Coflow evaluation, we compare against Varys and Aalo [12, 11], the state-of-the-art Coflow schedulers designed for packet switched networks. Both Varys and Aalo are designed with the objective of minimizing average CCT for a set of Coflows to mitigate the head-of-line blocking problem. We assume Sunflow uses the same Coflow priority policy as in Varys and Aalo, i.e. the shortest-Coflow-first policy.

5.3 Intra-Coflow Scheduling

5.3.1 Comparison with Circuit Switching Lower Bound

We begin our evaluation by comparing Sunflow's intra-Coflow performance against the theoretical lower bound T_L^c and Solstice. We observe the primary influence on optimality is the sender-to-receiver ratio in a Coflow, based on which we classify the Coflows as in Table 4, where 1) *One-to-One* Coflow is uni-cast with one sender, one receiver and one flow, 2) *One-to-Many* Coflow is with one sender and more than one receiver, 3) *Many-to-One* Coflow is in-cast with more than one sender and one receiver, and 4) *Many-to-Many* Coflow has more than one sender and more than one receiver.

How close is Sunflow to the optimal? Figure 3 highlights that Sunflow achieves near-optimal circuit scheduling. When $B = 1$ *Gbps*, the original setting of the Coflow trace (Figure 3a), and δ is 10 ms (typical of a 3D-MEMS optical switch), Sunflow CCT is 1.03× on average and 1.18× at the 95th percentile of the lower bound T_L^c. Note that Sunflow CCT/$T_L^c < 2$ always holds true. On the other hand, Solstice CCT/T_L^c performs much worse with 1.48× on average and 4.74× at the 95th percentile. Furthermore, we observe Solstice CCT can reach up to 10.63× of T_L^c.

Figure 3 also shows that, keeping δ constant, Solstice's performance is worse when B scales up, because the data processing time p becomes smaller compared with δ. The excessive preemptions scheduled by Solstice incurs heavy reconfiguration delays that significantly increase the CCT. From 10 Gbps to 100 Gbps, Solstice's

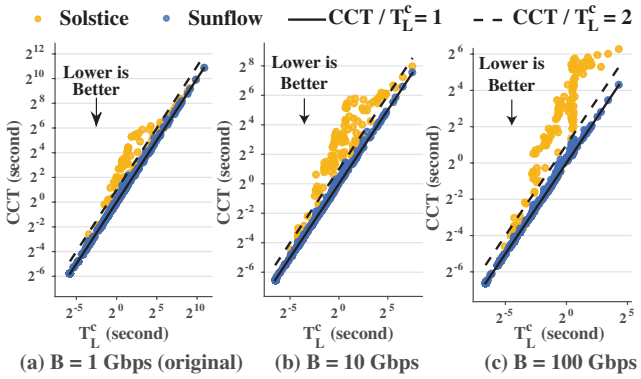

Figure 3: Sunflow and Solstice on different link rate B. Each subfigure shows CCT distribution w.r.t. T_L^c. Both axes are in logarithmic scale.

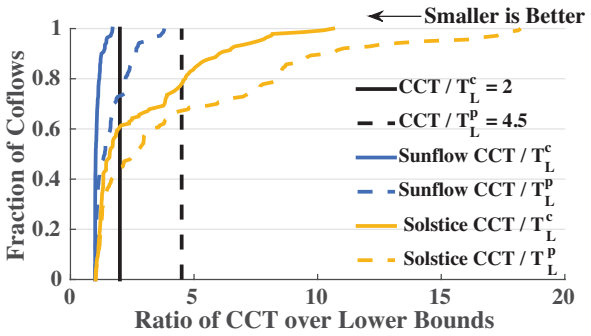

Figure 4: Distribution of CCT/T_L^c and CCT/T_L^p on many-to-many Coflows for Sunflow and Solstice.

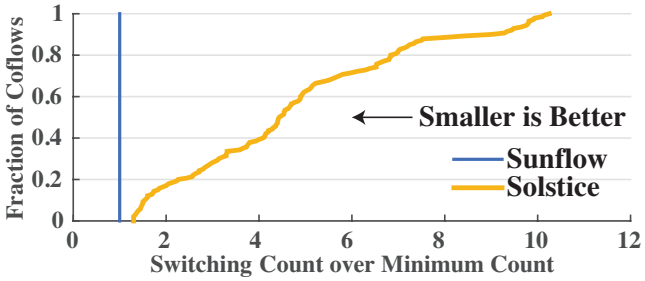

Figure 5: Distribution of switching count for many-to-many Coflows. Normalized over the minimum count for each Coflow.

average (95th percentile) CCT/T_L^c increases from 2.30 (10.06) to 3.17 (13.83), while Sunflow's average (95th percentile) CCT/T_L^c stays at 1.03 (1.24) and 1.04 (1.27).

Sunflow achieves optimal CCT of T_L^c for one-to-one, one-to-many and many-to-one Coflows. In one-to-one, one-to-many and many-to-one Coflows, all flows within a Coflow compete for the same port (visually the flows occupy a single row or column in the demand matrix). Simply scheduling circuits one by one for each flow would yield the optimal CCT in a circuit switching network, i.e. Sunflow CCT equals T_L^c for these Coflows. For these Coflows, Solstice also achieves optimal CCT because these specific Coflow structures happen to be handled by Solstice in a one flow per assignment manner.

Sunflow achieves near-optimal CCT for many-to-many Coflows (which account for over 99% of the bytes). Figure 4 shows the distribution of the ratio between CCT and lower bounds for these Coflows. Note that for Sunflow, CCT is bounded with CCT/$T_L^c < 2$. We find Sunflow's CCT/T_L^c is 1.10 on average and 1.46 at the 95th percentile. Again Solstice performs much worse with CCT/T_L^c of 2.81 on average and 7.70 at the 95th percentile.

Achieving good performance for many-to-many Coflows is paramount because they account for the vast majority of the Coflow traffic volume. Sunflow's approach excels in this many-to-many case. Figure 5 shows the number of circuit switching events, normalized by the minimum number of necessary switching, in these experiments. The minimum number of necessary switching for a Coflow is given by the number of subflows in the Coflow. Fewer switching events means less time wasted. As can be seen, Sunflow's switching count is optimal, while Solstice incurs numerous switchings for each subflow.

It is interesting to note that the switching count for Sunflow is always optimal regardless of the number of subflows, but for Solstice, the normalized switching count usually grows larger as the number of subflows increases. The linear correlation coefficient between Solstice's normalized switching count and the number of subflows is 0.84 for many-to-many Coflows. In other words, when more subflows are to be scheduled, Solstice would schedule more switchings per circuit.

Sensitivity to δ. We experiment with a range of δ from 100 ms to 10 μs, and the results are in Figure 6, where each CCT is normalized by the CCT when δ = 10 ms. We find that when δ is one order of magnitude slower than that of 3D-MEMS optical switch (i.e. δ = 100 ms), CCT is significantly worse. On the other hand, CCT can be somewhat improved with an optical switch that is one order of magnitude faster (i.e. δ = 1 ms). However, the marginal benefit from an even faster switch ($\leq$ 100 μs) is very small. The intuition is that at the intra-Coflow level, CCT is made up of the data transmission time and the switching delays. When δ gets smaller, CCT is dominated by the data transmission time, and δ becomes less relevant. For this reason, it should be expected that, as δ decreases, the inefficiency of Solstice linked to circuit switching overhead diminishes. For $B = 1$ *Gbps*, when δ drops below $O(100\mu s)$, Solstice's CCT performance is slightly better than Sunflow's because more circuit preemptions help Solstice handles skewed traffic slightly more efficiently.

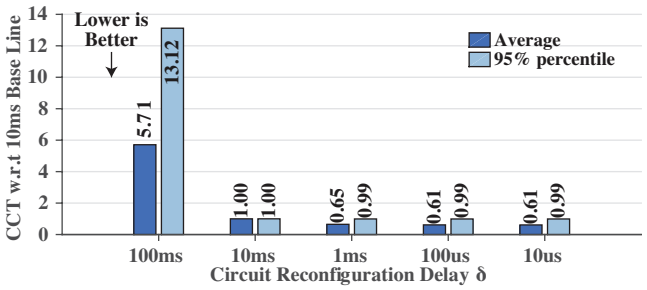

Figure 6: Sensitivity of intra-Coflow scheduling to δ. Normalized over CCT on $\delta = 10$ ms for each Coflow.

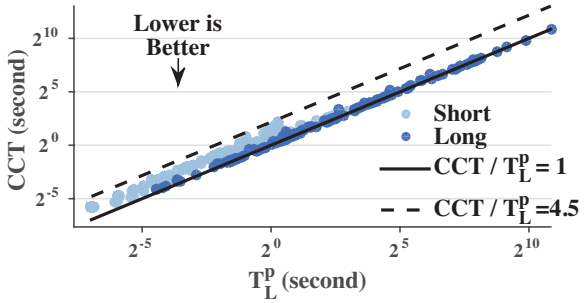

Figure 7: Sunflow CCT and T_L^p. Both axes are in logarithmic scale.

When δ=1 μs, Solstice's 95th-percentile CCT is equal to Sunflow's, and Solstice's average CCT is 0.98× of Sunflow's. Even so, Sunflow's non-preemptive strategy is still extremely critical for good performance since a preemptive algorithm like Solstice can incur more than 10× the switching overhead as shown in Figure 5.

Note that increasing B has the effect of making the circuit switch less efficient at a given δ. For example, if $B = 100$ *Gbps* and flow size does not increase, the Coflow transmission time will be reduced by approximately 100×. In that case, reducing δ by 100× from 10 ms to 100 μs does have benefits – CCTs are reduced to 0.49× at the 95th percentile, but going below 10 μs has marginal benefit.

Interestingly, these results suggest that if the typical data transmission time remains steady (i.e. flow size scales roughly 1:1 with link speed), it may be most fruitful to optimize switching technologies for $\delta = 1ms$, because the cost to develop faster switching technologies may outweigh the actual benefit.

Sensitivity to reservation ordering. To understand whether Sunflow's performance depends on reservation ordering, we conduct a sensitivity test. We change the order of reservations by ordering the demand in a Coflow in the following ways: (1) by sorting on the port label (called *OrderedPort*, used as the default in previous experiments), (2) by random (called *Random*), and (3) by sorting on the demand size (called *SortedDemand*). The average (95th percentile) CCT of *Random* is 0.94× (1.01×) the CCT of *OrderedPort*, while the average (95th percentile) CCT of *SortedDemand* is 0.95× (1.01×) the CCT of *OrderedPort*. These results show that Sunflow's performance is insensitive to reservation ordering.

5.3.2 Comparison with Packet Switching Lower Bound

Figure 7 compares Sunflow's CCT to the Coflow packet switching lower bound T_L^p and shows results for long Coflows separately. We consider a Coflow to be long if its average data processing time is larger than $40\times\delta$ (which corresponds to an average subflow size of $\geq$ 5 MB in the experiment). Specifically, the average data processing time for a Coflow C is $p_{\text{avg}} = \frac{\sum p_{i,j} \in C}{|C|}$, where $|C|$ is number of subflows in Coflow. Long Coflows account for 25.2% of Coflows and 98.8% of bytes in the trace.

Long Coflows have performance very close to the packet switching lower bound: CCT/T_L^p is 1.09 on average and 1.25 at the 95th percentile. For small Coflows, CCT/T_L^p is larger, but in absolute terms the extra time incurred by Sunflow is also small. Overall, CCT/T_L^p is 1.86 on average and 2.31 at the 95th percentile. Note that all Sunflow CCTs are within the theoretical bound of $\text{CCT}/T_L^p = 4.5$.

It is interesting to note that as p_{avg} increases, Sunflow's CCT gets closer to T_L^p. The rank correlation coefficient between p_{avg} and CCT/T_L^p is -0.96. For a Coflow with larger p_{avg}, circuits remain active longer after setup, circuit duty cycle is larger, and $\text{CCT}/T_L^p \to 1$.

Even though Sunflow's CCT can never be as low as packet switching due to the delay inherent to circuit switching, by providing CCT performance that is near that of packet switching, Sunflow enables networks to sacrifice some performance for the important data rate, energy, and longevity benefits of OCS (see § 1).

5.4 Inter-Coflow Scheduling

We now extend our evaluation from the intra-Coflow level to the inter-Coflow level. We begin our evaluation by comparing Sunflow against Varys and Aalo under the default settings. Note that Sunflow is based on optical circuit switching with $\delta > 0$, while Varys and Aalo are based on packet switching that amounts to $\delta = 0$. We first adopt the metric used in previous Coflow scheduling studies [12, 11, 38], which is the CCT ratio between competing schemes. Thus, we compute the ratios between Sunflow's CCT over Varys' and Aalo's CCT.

We find that Sunflow is 1.87× (2.52×) of Varys, and 1.69× (2.37×) of Aalo on average (at the 95th percentile). Note that Varys is slightly better than Aalo under the CCT ratio metric, which is consistent with the results in [11].

At first glance, these results are discouraging for Sunflow. However, it is also not hard to see why the average CCT ratio metric does not favor Sunflow. The circuit

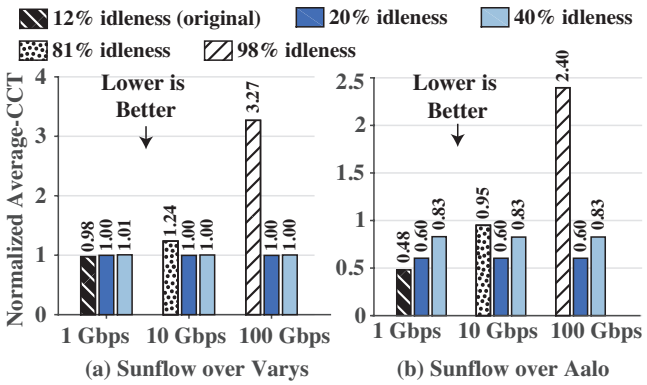

Figure 8: Sunflow average-CCT w.r.t. network idleness. Normalized over the average-CCT of (a) Varys and (b) Aalo under the same idleness.

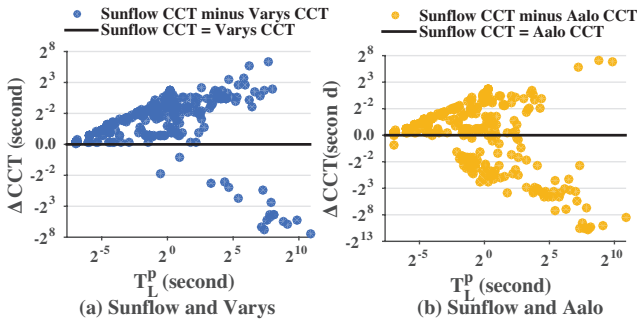

Figure 9: CCT difference between Sunflow (circuit switched) and Varys or Aalo (both packet switched). Coflows are spread out on the X-axis based on their T_L^p. Sunflow CCT is shorter with negative values. Both axes are in logarithmic scale.

switching overhead in Sunflow is significant for short Coflows, leading to large CCT ratios even though the absolute difference in CCT could be small. Specifically, for short Coflows, Sunflow is on average 2.16× to Varys, and 1.96× to Aalo. However, for long Coflows where the most data bytes come from, Sunflow CCT is comparable to Varys and Aalo: Sunflow is on average 1.07× to Varys, and 0.90× to Aalo. Since there are many more short Coflows than long Coflows, the average CCT ratio metric does not favor Sunflow.

Sunflow achieves comparable average CCT as Varys and Aalo. If we do not focus on pairwise CCT comparison, but instead focus on overall network efficiency and use the average CCT achieved by different schemes as the metric, the story becomes quite different.

Before we present the results, to further consider performance under various traffic load, we derive a metric that measures the network *idleness* for a specific set of Coflow traffic on a specific B. We consider a Coflow C to be active from its arrival time t^{Arr} to $t^{\text{Arr}}+T_L^p$, where T_L^p is CCT packet-switched lower bound given bandwidth B. At any time, we define network idleness as the portion of time when there is no active Coflow. This idleness metric considers Coflow characteristics and arrival rate, as well as the network bandwidth. Note that the metric is independent of the scheduling policy, and it is the upper bound on network idle time, because Coflows may stay longer than T_L^p in the network due to waiting for other Coflows. The network idleness in the original trace is 12%. When B is scaled from 1 Gbps to 10 Gbps and 100 Gbps, the idleness increases significantly from 12% to 81% and 98%. To evaluate under reasonable idleness, we scale the byte sizes of Coflows to attain 20% and 40% idleness for various B while preserving Coflows' structural characteristics.

In Figure 8, we compare the network efficiency by average CCT under various network idleness. By comparing across different network idleness from original 12% to 81% and 98%, we observe that Sunflow compares favorably to Varys and Aalo in modest or higher traffic load. When idleness is 81% or 98%, in many cases a Coflow finishes before the next one arrives. Under these underutilized settings, CCT for each Coflow is small, while Sunflow pays a relatively large circuit switching penalty. However, in a network with high or moderate traffic load, Sunflow achieves similarly high network efficiency as Varys and Aalo. This is because with more traffic, Coflows generally have longer waiting time that increases CCT, and thus the effect of δ on CCT is reduced. For 12%, 20% and 40% idleness, Sunflow's average CCT is within 1.01× of Varys and 0.83× of Aalo.

To compare the three algorithms more closely, we examine the individual CCTs under the 12% idleness setting, as shown in Figure 9.

On one hand, Coflows with small T_L^p may finish slower in Sunflow due to circuit switching delay. When multiple Coflows are involved in scheduling, a CCT is impacted by both the transmission time for the Coflow and the waiting time between Coflows that compete for bandwidth resources. In all schedulers, Coflows with smaller T_L^p are prioritized and do not have much waiting. Therefore, their CCT mainly consist of a small transmission time, with Sunflow paying a relatively large penalty due to the circuit reconfiguration overhead.

On the other hand, Coflows with large T_L^p may finish quicker in Sunflow than in Varys because Varys may leave bandwidth underutilized and prolong the CCT of large Coflows. Specifically, in Varys, subflows within the same Coflow may finish at different time because Varys opportunistically allocates different amounts of additional bandwidth to subflows when residual bandwidth is available. However, when a subflow finishes early, the associated bandwidth is left unused until the next rescheduling decision is made, which happens only on Coflow arrivals and completions [12]. Compared with Varys and Sunflow, Aalo introduces an extra inefficiency at the intra-Coflow level. Without known flow sizes, Aalo may allocate more bandwidth to small sub-

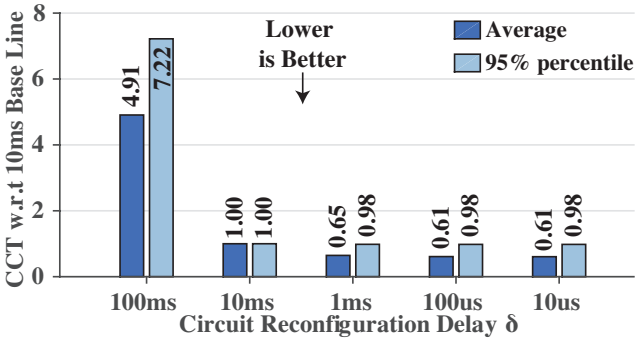

Figure 10: Sensitivity of inter-Coflow scheduling to δ. Normalized over CCT on $\delta = 10$ ms for each Coflow.

flows at the cost of delaying the long subflows, thus prolonging the overall CCT. This effect is more obvious for large Coflows.

When these two observations are combined, the average CCT of Sunflow, Varys, and Aalo become comparable. Thus, Sunflow enables networks to exploit the important data rate, energy, and longevity benefits of OCS (see § 1) while achieving good performance.

Sensitivity to δ. Our observations on Sunflow's sensitivity to δ under intra-Coflow scheduling (§ 5.3.1) generally also applies to inter-Coflow scheduling. Figure 10 shows the sensitivity analysis results. As before, compared to the 10 ms baseline, CCT can be somewhat improved with an optical switch with $\delta = 1$ ms. However, the marginal benefit from an even faster switch (≤ 100 μs) is very small. Thus, it is also the case for inter-Coflow scheduling, if the typical data transmission time remains steady (i.e. flow size scales roughly 1:1 with link speed), it may be most fruitful to optimize switching technologies for $\delta = 1ms$.

6. DISCUSSION

While this work focuses on the algorithmic aspects of circuit scheduling for Coflows, it is useful to discuss how Sunflow can be deployed. Fortunately, the facilities required to deploy Sunflow have been investigated in previous works [12, 25, 30]. Particularly, the OCS architecture for a compute cluster, the synchronization mechanism between host transmission and optical circuit setup, and the system for collecting Coflow requests as well as managing the flow transmissions have been proposed and implemented in previous works.

Centralized OCS control. A single controller can successfully schedule an OCS as shown in the Mordia design [30], which is based on a preemptive scheduling algorithm. Sunflow would require much less frequent circuit reconfigurations due to its non-preemptive nature, thus it would actually be easier to scale Sunflow up for a larger system. Sunflow is meant for controlling a single optical circuit switch. Adapting Sunflow for controlling a network of circuit switches is a subject of our future work.

Coordinating circuits and machines with ToR. As a ToR switch, REACToR [25] is capable of 1) synchronizing machines' transmission with circuit setup with explicit signals, 2) multiplexing between a packet switched network and a circuit switched network. Sunflow could use REACToR to achieve an efficient use of optical circuits. In addition, REACToR's hybrid architecture capable design allows a small-bandwidth packet switched network to help accommodate the little leftover traffic from the circuit switched network due to retransmissions or synchronization glitches.

Centralized Coflow scheduling. Centralized Coflow scheduling with Varys has been demonstrated in [12]. Similar to Varys, Sunflow reschedules only upon Coflow arrivals and completions, which makes centralized Coflow scheduling feasible. In Varys, a daemon agent runs on each machine to manage the flow sending rate, and collect the Coflow requests from the application running on the host. Sunflow asks for a small modification in the Varys agent: Rather than implementing per-flow rate control as instructed by the scheduler in Varys, for Sunflow scheduling, the agent on a sending machine would receive a row in the PRT corresponding to its port, count the circuit setup signals from REACToR, and when a circuit for its flow becomes active, the agent sends the flow at line rate.

Scheduler latency. With our untuned implementation in C++ when running on an Intel 3.5GHz processor, Sunflow's computation time is less than 1 sec for Coflows with up to 3,000 subflows. The performance can be further optimized with the use of more efficient data structures. Besides, unlike other schedulers that schedule circuits in a batch with *assignments* (§ 3.1.1), Sunflow may schedule each computed circuit individually, thus hiding the scheduling latency by overlapping circuit setup with data transmissions. To further reduce computation time, it is possible to leverage approximation schemes (e.g. by rounding subflow sizes up to thresholds to prune the loops of circuit release events in Algorithm 1 Line 10), or to leverage parallelism (e.g. by computing circuit schedules on partitioned demands in parallel). However, approximation and parallelization could reduce the optimality of the resulting schedules.

7. CONCLUSION

We have presented Sunflow, the first circuit scheduling algorithm for Coflow that is provably within a factor-of-two to the optimal, and we have shown that in practice the gap is much smaller. Sunflow handles different kinds of Coflows (e.g. one-to-many, many-to-one, many-to-many) equally well, and its performance is good even with the circuit switching delay typical of today's 3D-MEMS switches. As a Coflow's size increases, Sunflow's CCT performance approaches the packet switching lower bound. Under modest to heavy traffic load,

Sunflow achieves comparable average CCT to a packet switched network employing the state-of-the-art Coflow schedulers. In conclusion, with Sunflow's scheduling benefits, an OCS can be a viable component in a modern data-parallel cluster network.

Acknowledgements

We thank Simbarashe Dzinamarira, Dingming Wu, Yiting Xia, our shepherd Paolo Costa, and the anonymous reviewers for useful feedback. We also thank Mosharaf Chowdhury for sharing the Coflow traffic trace. This research is sponsored by the NSF under CNS-1422925, CNS-1305379 and CNS-1162270.

Appendix

This section contains the theoretical proofs and analysis for results presented in § 4.1.2 and § 4.1.3.

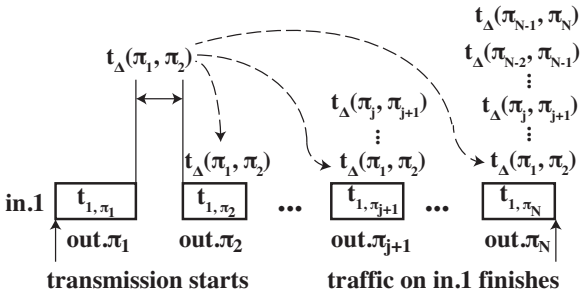

Figure 11: Sunflow scheduling circuits on $in.1$

Proof of Lemma 1. We begin by proving the following proposition: *Under Sunflow scheduling, any input and output port will finish traffic within $2T_L^c$ after transmission starts.*

Without loss of generality, consider an input port denoted as $in.1$. In Sunflow, the time spent on each flow is given by Equation (3). There exists a permutation of $\{1, ..., N\}$, denoted as $\{\pi_{(in.1,1)}, ..., \pi_{(in.1,N)}\}$ for $in.1$, so that Sunflow setup circuit [$in.1$, $out.\pi_{(in.1,j)}$] no later than [$in.1$, $out.\pi_{(in.1,j+1)}$]. Note that the permutation may differ for different ports. For notation simplicity, we refer to $\pi_{(in.1,j)}$ as π_j of $in.1$ in the following proof. Hence the ordering of output ports to be served for $in.1$ is $out.\pi_1$, ..., $out.\pi_j$, $out.\pi_{j+1}$, ..., $out.\pi_N$ (Figure 11). Note that t_{1,π_1} starts as soon as transmission starts, since $in.1$ can always match with an output port.

Denote $t_\Delta(\pi_j, \pi_{j+1})$ as the gap in time between demand t_{1,π_j} finishes and $t_{1,\pi_{j+1}}$ starts. During $t_\Delta(\pi_j, \pi_{j+1})$, $in.1$ is idle. Since Sunflow fails to find an output port to match $in.1$, *all* output ports of $out.\pi_{j+1}$, $out.\pi_{j+2}$, ..., $out.\pi_N$ must be transmitting. Thus, during $t_\Delta(\pi_j, \pi_{j+1})$, the demand on output ports $out.\pi_{j+1}$, ..., $out.\pi_N$ is decreased by $t_\Delta(\pi_j, \pi_{j+1})$, as shown by output port $out.\pi_{j+1}$ in Figure 11. Note that the demand served for any output port during $t_\Delta(\pi_j, \pi_{j+1})$ can be demand from any input port *other than* $in.1$.

Consider the output port $out.\pi_N$. t_{1,π_N} is the last to be served among $t_{1,\pi_1}, ..., t_{1,\pi_N}$. The demand on output port $out.\pi_N$ satisfies

$$\sum_{j=1}^{N-1} t_\Delta(\pi_j, \pi_{j+1}) \leq \sum_i t_{i,\pi_N} \leq T_L^c \tag{5}$$

Thus the total time that $in.1$ stays idle during transmission is at most T_L^c, so $in.1$ will finish its traffic in

$$\begin{aligned} T &= t_{1,\pi_1} + t_\Delta(\pi_1, \pi_2) + ... + t_\Delta(\pi_{N-1}, \pi_N) + t_{1,\pi_N} \\ &= \sum_{j=1}^{N-1} t_\Delta(\pi_j, \pi_{j+1}) + \sum_{j=1}^{N} t_{1,\pi_j} \\ &\leq T_L^c + T_L^c = 2T_L^c \end{aligned} \tag{6}$$

Equation (6) holds for all input and output ports, which proves our proposition.

According to our proposition, Sunflow would finish transmission for all input and output ports within $2T_L^c$, thus Sunflow CCT satisfies

$$T_S \leq 2T_L^c \tag{7}$$

The CCT under optimal scheduling in a circuit switched network must be at least T_L^c, i.e. $T_O^c \geq T_L^c$. Thus Sunflow's CCT is bounded by

$$T_S \leq 2T_L^c \leq 2T_O^c \tag{8}$$

Our proof holds for any ordering of the circuits to be scheduled. □

Proof of Lemma 2. For any flow in a Coflow, the relationship between $t_{i,j}$ (Equation (3)) and $d_{i,j}/B = p_{i,j}$ (Equation (1)) is given by

$$t_{i,j} \leq p_{i,j} + \delta \leq (1 + \alpha)p_{i,j} \tag{9}$$

Equation (2), Equation (4), and Equation (9) give

$$T_L^c \leq (1 + \alpha)T_L^p \tag{10}$$

Note that the port that gives T_L^c may be different from the port that gives T_L^p, but Equation (10) holds true regardless of whether such two ports are identical or not. Therefore we have

$$T_S \leq 2T_L^c \leq 2(1 + \alpha)T_L^p \leq 2(1 + \alpha)T_O^p, \tag{11}$$

which leads to the conclusion of Lemma 2 □

Time Complexity Analysis. Sunflow stops at each circuit release time in the PRT and tries to schedule circuits (Algorithm 1 Line 10). For a Coflow C, the number of circuit release events is less than $|C|$, where $|C|$ denotes the number of subflows in C. At each circuit release time, Sunflow scans through the Coflow demand to reserve circuits (Algorithm 1 Line 6). The number of entries in the Coflow demand is less than $|C|$. Hence the complexity for a single Coflow is $O(|C|^2)$. In the worst case, a Coflow may cover all input and output ports, resulting in $|C| = N^2$.

8. REFERENCES

[1] Apache Hive. http://hive.apache.org.

[2] Apache Tez. http://tez.apache.org.

[3] Based on private communications with Glimmerglass Networks Inc., an optical switch manufacturer.

[4] Coflow benchmark based on Facebook traces. https://github.com/coflow/coflow-benchmark.

[5] Al-Fares, M., Loukissas, A., and Vahdat, A. A scalable, commodity data center network architecture. In *ACM SIGCOMM* (2008).

[6] Allahverdi, A., Ng, C., Cheng, T. E., and Kovalyov, M. Y. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research 187*, 3 (2008), 985–1032.

[7] Balas, E., and Landweer, P. R. Traffic assignment in communication satellites. *Operations Research Letters 2*, 4 (1983), 141–147.

[8] Birkhoff, G. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán Rev. Ser. A 5* (1946), 147–151.

[9] Calient. S series optical circuit switch. http://www.calient.net.

[10] Chowdhury, M., and Stoica, I. Coflow: A networking abstraction for cluster applications. In *ACM Hotnets* (2012).

[11] Chowdhury, M., and Stoica, I. Efficient coflow scheduling without prior knowledge. In *ACM SIGCOMM* (2015).

[12] Chowdhury, M., Zhong, Y., and Stoica, I. Efficient coflow scheduling with Varys. In *ACM SIGCOMM* (2014).

[13] Dasylva, A., and Srikant, R. Optimal WDM schedules for optical star networks. *IEEE Transactions on Networking (TON) 7*, 3 (1999), 446–456.

[14] Edmonds, J. Paths, trees, and flowers. *Canadian Journal of Mathematics 17* (1965), 449–467.

[15] Farrington, N., Porter, G., Fainman, Y., Papen, G., and Vahdat, A. Hunting mice with microsecond circuit switches. In *ACM HotNets* (2012).

[16] Farrington, N., Porter, G., Radhakrishnan, S., Bazzaz, H. H., Subramanya, V., Fainman, Y., Papen, G., and Vahdat, A. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *ACM SIGCOMM* (2010).

[17] Glimmerglass. Intelligent optical system. http://www.glimmerglass.com.

[18] Gonzalez, T., and Sahni, S. Open shop scheduling to minimize finish time. *Journal of the ACM (JACM) 23*, 4 (1976), 665–679.

[19] Gopal, I. S., and Wong, C. K. Minimizing the number of switchings in an SS/TDMA system. *IEEE Transactions on Communications 33*, 6 (1985), 497–501.

[20] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. VL2: A scalable and flexible data center network. In *ACM SIGCOMM* (2009).

[21] Inukai, T. An efficient SS/TDMA time slot assignment algorithm. *IEEE Transactions on Communications 27*, 10 (1979), 1449–1455.

[22] Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. Dryad: Distributed data-parallel programs from sequential building blocks. In *ACM EuroSys* (2007).

[23] Kesselman, A., and Kogan, K. Nonpreemptive scheduling of optical switches. *IEEE Transactions on Communications 55*, 6 (2007), 1212–1219.

[24] Lin, H.-C., and Wang, C.-H. A hybrid multicast scheduling algorithm for single-hop WDM networks. In *IEEE INFOCOM* (2001).

[25] Liu, H., Lu, F., Forencich, A., Kapoor, R., Tewari, M., Voelker, G. M., Papen, G., Snoeren, A. C., and Porter, G. Circuit switching under the radar with REACToR. In *USENIX NSDI* (2014).

[26] Liu, H., Mukerjee, M. K., Li, C., Feltman, N., Papen, G., Savage, S., Seshan, S., Voelker, G. M., Andersen, D. G., Kaminsky, M., et al. Scheduling techniques for hybrid circuit/packet networks. In *ACM CoNEXT* (2015).

[27] Luo, S., Yu, H., Zhao, Y., Wang, S., Yu, S., and Li, L. Towards practical and near-optimal coflow scheduling for data center networks.

[28] McKeown, N. The iSLIP scheduling algorithm for input-queued switches. *IEEE Transactions on Networking 7*, 2 (1999), 188–201.

[29] Polatis. Series 7000 software defined optical switch. http://www.polatis.com.

[30] Porter, G., Strong, R., Farrington, N., Forencich, A., Chen-Sun, P., Rosing, T., Fainman, Y., Papen, G., and Vahdat, A. Integrating microsecond circuit switching into the data center. In *ACM SIGCOMM* (2013).

[31] Qiu, Z., Stein, C., and Zhong, Y. Minimizing the total weighted completion time of coflows in datacenter networks. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)* (2015).

[32] Rouskas, G. N., and Sivaraman, V. Packet scheduling in broadcast WDM networks with arbitrary transceiver tuning latencies. *IEEE Transactions on Networking (TON) 5*, 3 (1997), 359–370.

[33] Towles, B., and Dally, W. J. Guaranteed

scheduling for switches with configuration overhead. *IEEE Transactions on Networking 11*, 5 (2003), 835–847.

[34] Wang, C.-H., Javidi, T., and Porter, G. End-to-end scheduling for all-optical data centers. In *IEEE INFOCOM* (2015).

[35] Wang, G., Andersen, D. G., Kaminsky, M., Papagiannaki, K., Ng, T. S. E., Kozuch, M., and Ryan, M. c-Through: Part-time optics in data centers. In *ACM SIGCOMM* (2010).

[36] Wang, G., Ng, T. S. E., and Shaikh, A. Programming your network at run-time for big data applications. In *ACM HotSDN* (2012).

[37] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *USENIX NSDI* (2012).

[38] Zhang, H., Chen, L., Yi, B., Chen, K., Chowdhury, M., and Geng, Y. CODA: Toward automatically identifying and scheduling coflows in the dark. In *ACM SIGCOMM* (2016).

[39] Zhao, Y., Chen, K., Bai, W., Yu, M., Tian, C., Geng, Y., Zhang, Y., Li, D., and Wang, S. RAPIER: Integrating routing and scheduling for coflow-aware data center networks. In *IEEE INFOCOM* (2015).