# Dynamic Optical Switching for Latency Sensitive Applications

Henrique Rodrigues, Richard Strong, Alper Akyurek, Tajana S. Rosing
{hsr,rstrong,aakyurek,tajana}@eng.ucsd.edu
University of California, San Diego

## Abstract

Dynamic optical interconnects using fast Optical Circuit Switches (OCS) are emerging as a scalable and energy efficient alternative to increasing network demands. Initial concerns regarding slow switching speeds of OCSs were recently overcome, with prototypes enabling circuit setup in a few microseconds. This can potentially broaden the classes of traffic patterns that can be carried efficiently by an all-optical interconnect. However, application performance on such newer interconnects has not been fully understood yet. In this paper, we explore the the gap between advances in faster OCS hardware and the potential success of such newer technologies in terms of application performance and cluster energy efficiency. We evaluate the performance of latency-sensitive distributed applications running on a fast OCS environment, analyzing its impact to overall server and network energy efficiency. We also discuss scheduling inefficiencies of current fast OCSs and evaluate ideas to solve them. We find that while some distributed applications suffer minimal performance penalty when running on fast OCSs, more flexible schedulers, like the ones outlined in this paper, improve application performance and OCS efficiency by up to 2.44 times compared to the strategies in the literature.

## Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*circuit-switching networks, packet-switching networks, distributed networks*

## General Terms

Algorithm, Design, Experimentation, Performance

## Keywords

Data center Networks, Energy Efficiency, Optical Switching

## 1. INTRODUCTION

Digital Packet Switching (DPS), also called Electronic Packet Switching (EPS), has been the basic building block for interconnects in the data center. However, achieving scalable bandwidth with digital switches incurs a significant rise in power consumption at higher speeds. Some high speed switches, such as the Huawei CE12812 [2], require power supplies for up to 170 Watts (W) per 100Gbps port (16.2 kW total for 96 ports), ten times more than a traditional 10Gbps DPS switch port. The power comes from extra buffering space, cooling and processing capacity in the switch data plane, needed to process several million packets per second at 100Gbps. Providing scalable bandwidth with DPSs is not only challenging in terms of power consumption, but also in terms of cabling complexity and maintenance costs of non-oversubscribed packet networks [31, 33, 39].

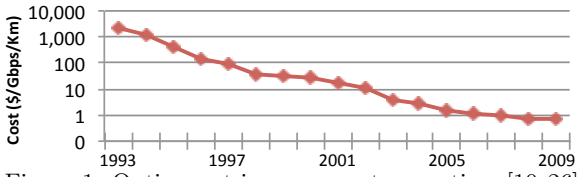

Figure 1: Optics cost improvements over time [10, 26].

In this scenario, Optical Circuit Switching (OCS) is emerging as a bandwidth-scalable and energy efficient alternative to DPS-only networks [11,17,25,26,33]. The technology provides abundant bandwidth at minimal power consumption, in the order of milliwatts per port. It can also reduce cabling complexity and costs, as multiple high speed data streams can be carried on a single fiber cable using Wavelength Division Multiplexing (WDM). Continuous cost reductions (see Fig. 1), might reduce the overall cost of high bandwidth interconnects and improve network performance [17, 26].

However, well known limitations of OCS interconnects restricted their use to relatively static networks [6, 11, 17, 41]. First, an OCS does not process packet headers and requires a separate control plane to configure optical circuits before data can flow through the interconnect. Establishing and tearing down a circuit is a slow process with high *reconfiguration delays* ($10^{-3}$ seconds for a MEMS OCS compared to $10^{-5}$ of total processing time for DPSs), which limits the interconnect *flexibility*. Another problem is the lack of efficient optical buffers, which hinders OCSs from absorbing bursts of traffic and dealing with port contention as a DPS.

Fortunately, recent works show that it is possible to reduce OCS *reconfiguration delay* dramatically, providing switching delays of $10^{-6}$, orders of magnitude faster than traditional OCSs [19]. However, while this has the potential to mitigate most OCS limitations, it is not yet clear what kinds of applications can make efficient use of a faster OCS. Even worse, evaluations of optical interconnects in the literature use either long lasting synthetic traffic patterns [25, 26, 33] or throttle DPS capacity to levels which inflate the apparent benefits of the OCS counterpart [41].

In this paper, we explore the gap between advances of faster OCS hardware and the potential success of such newer technologies in terms of performance and energy efficiency of real applications. We discuss several types of applications, but focus primarily on latency-sensitive distributed ones which were not suitable for traditional dynamically switched optical interconnects. We discuss OCS scheduler

inefficiencies of current prototypes, and propose two ideas to solve them. The first idea explores the ease of multicast provided by passive optical devices. We show how trading off bandwidth for latency can improve application performance over current schemes. The second shows how distributed scheduling can be used to increase flexibility of OCSs, resulting in better utilization and application performance.

In summary, we present the following contributions:

- We analyze the role of the network on the energy consumption of data centers, focusing on overall costs of hybrid packet-optical interconnects facing recent industry trends. We further discuss the potential of optics in terms of application performance and energy efficiency.
- Multiple questions on the feasibility of dynamic optical interconnects are discussed. These are often raised by vendors and researchers, seeking to understand requirements of future optical switches and applications.
- We study the impact of using an OCS to forward traffic from diverse applications, showing that current OCS technologies could lead to overall data center inefficiency without careful circuit scheduling.
- We pinpoint scheduler overheads of current fast OCS, and evaluate changes that improve performance and energy efficiency for latency-sensitive distributed applications.

# 2. BACKGROUND AND MOTIVATION

In this section we give a brief historical perspective on OCS interconnect adoption. The discussion introduces the current problem that prevents wider OCS adoption, which is part of our motivation for this work.

## 2.1 Evolution of dynamic optical interconnects

The idea of using all-optical switching to compliment the weaknesses of DPS (bandwidth scalability) is not new, and it has been deployed extensively for several years in Wide Area Networks (WAN) and High Performance Computing (HPC) clusters [6]. However, in these scenarios, optical switching has always been restricted to static bandwidth provisioning. In WANs for example, all-optical paths are allocated to enterprises or internet service providers (ISPs) to provide high speed connectivity between two distant sites or data centers. These optical paths remain active for time scales of several days or years. In traditional HPC clusters, optical switching usually takes place only at the beginning of a large scale computation. This allows for changing the network topology to match the interconnection that is most suitable for each specific application (e.g. Mesh, Hypercube, Torus, etc.).

Restricted use of all-optical networks was a result of the slow switching time (or *reconfiguration delay*) of traditional optical switches. Previously, they relied on mechanical systems or manual operation, resulting in delays well over a second. However, all-optical transmission is preferred for high bandwidths because transmission of modulated light through a fiber requires much less power than copper based interconnects [30]. Photonics switches also require less power than their electronic counterparts. Furthermore, optical transmission allows the transmission of multiple high speed data stream through the same fiber and/or OCS port using Wavelength Division Multiplexing (WDM), increasing the capacity of the port without significant rise in power.

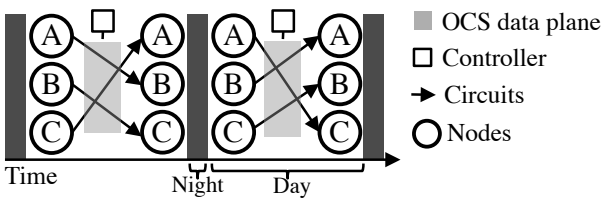


Figure 2: Representation of circuit scheduling in terms of days (circuits active) and nights (data plane reconfiguring).

Optical switching speeds have been improving over time, and can potentially be faster than a nanosecond [42], motivating its use in a dynamic optical interconnect. This idea is also not new, and suggestions dates back to at least ten years ago [6]. Several interconnects using dynamic optical switching were proposed since then [11, 17, 19, 26, 41]. Even so, the technology is still received with skepticism by most cloud providers and enterprise data centers. For example, the recently published Facebook network infrastructure [14], which defines the architecture for its next-generation data centers, does not include optical switching components. But what prevents the adoption of dynamic optical switching?

## 2.2 Problems and limitations

Current OCSs impose duty cycles with periods of complete network downtime that vary according to the scheduling scheme. Current technology requires minimum downtime of 10$\mu$s due to switch *reconfiguration delays* [21,33]. If we also account for *scheduling algorithms* and *network controller* delays, downtime can be much longer [11, 17].

The point-to-point restriction in fast optical switches [19, 21, 43] makes the problem worse. In this case, connectivity is restricted to 2 nodes at a time, and network downtime from *reconfiguration delays* happens whenever the interconnect state changes. Figure 2 depicts this resource sharing scheme with 3 nodes, while maximizing the possible number of active circuits between nodes. *Nights* and *days* represent the time intervals when the network is being reconfigured (no light can flow through the interconnect) and when it can carry data, respectively. This round robin schedule, which resembles Time Division Multiple Access (TDMA), was used in recent fast OCS prototypes due to its ease of implementation [25, 33]. However, it can result in extra application delays as sharing of network resources is limited. Consider for example an OCS interconnect with 11 nodes and 10$\mu$s *nights*. If a single node has demand towards all other nodes, these nodes will not wait only 10$\mu$s to receive their data, but much more. Assuming that each circuit is kept active for 90$\mu$s, to allow 90% of useful interconnect bandwidth, one of the nodes will have to wait 1.01 milliseconds to start receiving its data. As we discuss later, such delays can be disastrous for some applications.

Due to these limitations, it is not yet clear what kinds of applications can make efficient use of OCS interconnects. In this context, the next section provides a comparison between the costs of OCS and DPS interconnects of same bandwidth, taking in consideration OCS limitations.

## 2.3 Interconnect Cost and Efficiency

OCS interconnects are claimed to be cheaper than a DPS of same bandwidth capacity, both in terms of capital expenditure (CAPEX) and operating expenses (OPEX). This section analyzes this claim including a variable not considered

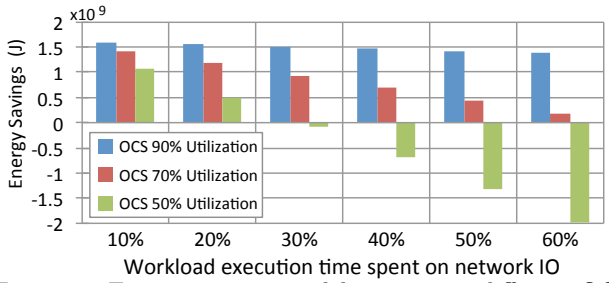

Figure 3: Energy savings model comparing different OCS efficiency to a 100G non-oversubscribed EPS.

in previous studies: optical interconnect efficiency. Since fast OCS technology is relatively new, we leave the discussion on CAPEX aside and focus instead on energy consumption, which is a significant part of OPEX in the data center and one of the biggest advantages of OCS over DPS.

Data center networks are usually composed of multiple layers of commodity DPSs, with the basic building block being a DPS with $k$ ports [33]. These are organized as a Folded Clos [5]. The number of layers depends on the switch radix $k$ and number of servers. In general, these networks can support $k^N/2^{N-1}$ servers, with each layer requiring $k^{N-1}/2^{N-2}$ switches (half of this amount for the top layer). The intuition behind this topology is that each layer provides as many outgoing paths as the number of incoming links. At the lower layer, there are $k^N/2^{N-1}$ incoming paths which are server links, and since all paths have the same bandwidth, full-bisection bandwidth is achieved.

Some cloud providers report data centers of up to 80K servers. We use as an example a 65K servers data center. A non-oversubscribed DPS network these servers requires two layers of 131072 ports. The third, or core layer, requires 1/2 of this amount, 65536 ports to allow full bisection bandwidth. Using 10Gbps ports at 12.5 W [17], this would result in 819 kW for the core, or 950 kW if we also consider 1 W of power for transceivers at each end of a link.

A hybrid 10% DPS, 90% OCS network can be built by replacing 90% of bandwidth provided by the DPS gear with OCSs, as suggested in Helios [17]. This would drive the DPS power down to 125 kW, as the capacity is reduced to 10% and is now 10:1 oversubscribed. The optical counterpart, if built using WDM and 8 wavelengths per fiber (i.e. 80Gbps per port), would require 7360 OCS ports, down from 58K ports which corresponds to 90% of the core layer. Helios used a Glimmerglass OCS at 240 mW/port, or a total of 1.96 kW. The number of transceivers is reduced to 72K as they are not needed for OCS ports, totaling 72 kW. The total power for the hybrid interconnect core would be 185 kW, more than 6x reduction compared to a non-oversubscribed DPS offering the same bandwidth capacity. We repeat this exercise for future servers with 40Gbps network interfaces cards (NICs) – which have increasing presence in the market [3] – and 100G network core, maintaining the non-oversubscription restriction. At 170 W/100Gbps DPS port [2], the power of the core layer reaches 4500 kW, compared to 447 kW for the hybrid interconnect; part of why oversubscription is still common [14].

From this we draw two important observations in terms of power consumption. First, this analysis reinforces the claim that OCS could solve the inherent limitation of power consumption for high bandwidth DPS interconnects. However, the analysis does not take into account the limitations of an OCS. In other words, would an OCS interconnect be able to carry 90% of the traffic with the same efficiency of a DPS?

Since DPS and OCS strengths are distinct (flexibility vs. bandwidth/power), it is not reasonable to compare DPS and OCS with similar bandwidth capacity considering only throughput and power consumption. To understand the tradeoff in terms of interconnect efficiency and energy consumption between OCS and DPS, we extend the power consumption model described above considering workload execution time. The motivation for that is twofold. First, the actual energy of a system is given by $E = \int_t P(t)dt$. Second, usage time dictates the cost of resources in cloud computing data centers. In the pay-as-you-go model of utility computing, the time required to complete workloads can be used to estimate the price difference between buying hardware or "leasing" cloud resources. Cloud providers use dynamic resource management systems to address variable demands, shutting down idle devices to save energy [22]. Introducing a timing component to the model accounts for resource usage in terms of completion time of applications, which depends not only on server processing speeds, but also on interconnect efficiency. If the network is oversubscribed (DPS), or if scheduling (OCS) affects application runtime, energy consumption of the workload is likely to increase.

We consider a hypothetical workload, composed of several data-intensive jobs and low latency applications, making use of all 64K servers and a non-oversubscribed DPS to its completion in 400s. Commodity servers normally consume between 170 to 350 W [7], with average server power around 210 W. We estimate the baseline power consumption for the data center modeled before (64K, 40Gbps servers and 100G network core) to be 18.35 MW. For simplicity, we assume a linear relationship between the network performance and flow completion time (i.e. a transfer of 1.25GB completes in 1s using a 10Gbps interconnect and ∼10s using a 1Gbps interconnect), which ignores protocol overheads and gives a lower bound in our analysis. For example, at 90% interconnect efficiency, we assume a 10% slowdown in the application time spent on network. As applications are composed by computation and network phases, we use Amdahl's law to calculate the interconnect's efficiency impact on total runtime. Figure 3 compares the gains on energy consumption for a hybrid OCS over a DPS interconnect under different levels of OCS interconnect efficiency. Each set of bars shows the energy consumption of servers and networking gear for total application runtime, varying the amount of time the workload spent on network (i.e. 10% in x-axis means that 10% of the workload is spent on network IO). With increased network demand, OCS can outperform a DPS-only network in terms of energy consumption, but this depends on how efficiently it can handle the imposed demand. However, if OCS efficiency is too low to carry offered traffic, this could instead increase the overall energy consumption.

## 3. REQUIREMENTS

Which applications can make efficient use of an OCS interconnect? To answer this question, one must take into account application characteristics in terms of both traffic patterns and computation dependencies, as well as the efficiency of an optical interconnect. Unfortunately, most optical switching prototypes described in literature were only evaluated with synthetic traffic [17,25,26,33] and fail to pro-

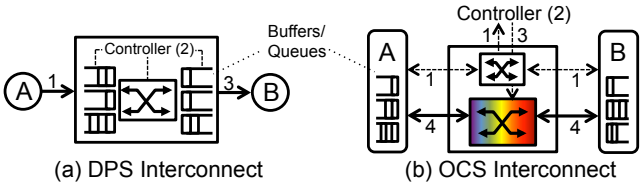


Figure 4: Operation of DPS and OCS interconnects. In DPS, forwarding information is read from packets (2), and controller has local access to demands. OCS requires an external controller which aggregate demands (1,2,3) and reconfigure its data plane (4) with the computed schedule.

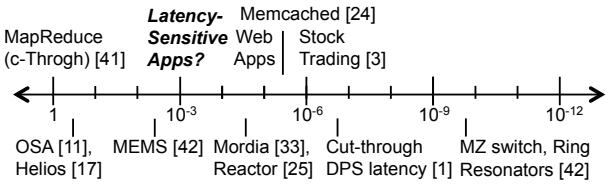


Figure 5: Overview of delays (seconds) induced by network latency (bottom) and common application runtime (top).

vide a good answer to this question. Synthetic traffic composed by independent flows don't capture application dependencies, and might provide a false impression of high OCS efficiency. To make matters worse, neither distributed application traffic patterns nor optical interconnect efficiency are easy to determine and reproduce in practice at $\mu$s time granularity. The former usually depends on the overlap of communication and computation phases of the application. When the network resource availability is unstable (as in a OCS), we also have account for interferences in performance from network algorithms such as TCP [25], which is constantly adapting to resource availability in the network. As a result, analyzing the performance of an application running on an optical switch using synthetic traffic does not necessarily provide accurate answers. Instead, we analyze the performance of real applications in an environment which emulates an optically circuit switched interconnect. However, before entering in the details of applications and their performance, it is useful to review the operation of each type of interconnect and clarify the differences between them.

## 3.1 Resource provisioning

The disparity between the interconnects comes from how data is handled at every switching node. DPSs can store and process data broken into packets as they arrive at the switch, with the actual forwarding decision often done by ASICs. Their total processing time can be less than 0.5$\mu$s [1]. Buffering allows the switch to absorb bursts of demand coming from several input ports and deal with output port contention. Steps 1, 2 and 3 of Figure 4 (a) show their operation, with data packets transmitted from node A to the DPS in step 1. The DPS stores and processes each packet in step 2, and sends it towards the destination node identified from the routing information in packet headers. Fiber optic cables are usually used in steps 1 and 3, and transceivers perform optical-electronic-optical (OEO) conversions at beginning and end of step 2.

OCS interconnects handle data transmission differently, as shown in Figure 4 (b). The first difference is that their data plane does not process packets, and thus do not require transceivers. This reduces the cost of the interconnect, as the (per port) cost of expensive transceivers is eliminated [25]. Because data flows through the switch in the optical domain without OEO, OCSs are bandwidth-agnostic, and can be used at 10Gbps, 100Gbps or more, consuming the same amount of power per port [17].

However, the lack of buffers in the switch means that circuits need to be configured before data transmissions. Since buffering is still required to prevent loss of application data, buffers are pushed to the ends of the interconnect, as shown in Figure 4 (b). Steps 1, 2 and 3 use a simpler and lower capacity Digital Interconnect (DI) to communicate buffered demands to an OCS controller. In step 2, the controller computes the best circuit switching schedule according to the demands collected in step 1 and sends the schedule to the OCS data plane in step 3 [11, 33, 41]. After the OCS has a schedule, data is consumed from the buffers in the boundaries of the interconnect and flows through the switch entirely in the optical domain without OEO. Data in buffers are usually organized in queues according to their destination. These queues are consumed only when a circuit between two nodes is active, and the schedule dictates when, and for how long, circuits will be active.

The data plane of an OCS interconnect can be composed of several distinct switching nodes (even though it was depicted in Fig. 4 as a single node), sharing a common controller [11]. The architecture described is general enough to represent such interconnects, since all the forwarding is done without OEO across multiple nodes. This design was used by several recent OCS interconnects [11, 17, 19, 25].

## 3.2 Network Requirements

Figure 5 provides an overview of the relationship between application latency requirements (top) and switching speeds (bottom). Optical switching speeds have been improving over time. On the left side, interconnects such as Helios [17] or OSA [11], built using MEMS switches, are not efficient for dynamic optical interconnects. They require traffic pattern stability of a few seconds to allow reasonable OCS efficiency [17]. Current OCSs have reconfiguration delays of a few $\mu$s and scale up to a few dozen ports [21,33], but have not been experimented with real applications. We focus on this type of switches. Faster technology might provide sub-$\mu$s switching with reasonable port counts in the future [33, 42].

In terms of application latency requirements, at the right is High Frequency Stock trading, which is perhaps the application where latency is the most critical. Algorithmic trading accounts for the majority of transactions in markets, and delays in transactions directly affect gains/losses [23]. Algorithms are placed as close as possible to the trading center, and in some cases are even embedded in the data plane of DPSs to avoid switching hops (for example, the Arista 7124FX [1] comes with a programmable integrated circuit).

Next is memory caching, which is widely used to mitigate high latencies from databases in secondary storage. The mean response time of a single request for memory cached data is about 90$\mu$s [24] (including network and processing time). At the current OCS speeds, a interconnect with no load would increase this time by at least 10$\mu$s, accounting for 10% of total time. However, requests hardly ever involve only one server, and the interconnect is not likely to be idle. Web applications such as Facebook or Google issue several small requests at the same time to various back-end servers for a single client-facing request. For Facebook, a single

request to a front-end server triggers, on average, 88 cache lookups and 392 RPC calls to back-end servers [16].

Would current optical interconnects support applications such as Facebook? We formalize the expected completion time of a single request for this type of application as follows. Let $P$ be the time required to transmit a single message fragment (i.e. a standard MTU sized packet) through the interconnect (e.g. a 1500 byte packet takes 1.2$\mu$s to be transmitted at 10Gbps). $F_i$ is the number of message fragments in the request message from the front-end server to each back-end server $i$, out of $N$ in total. $B_i$ represents the same, but for the reply message. In a DPS with switching delay $S$, the completion time for this task can be given by:

$$Time_{DPS} = \sum P \cdot F_i + \sum P \cdot B_i + 2N \cdot S \qquad (1)$$

In an optical interconnect, we have to account for controller delays $C$ and reconfiguration delays $R$. Due to the restriction of 1-to-1 connectivity, every request or response incurs one extra reconfiguration. Since *days* are normally fixed to allow predictable duty cycle, we assume days to be as high as transmission of the biggest message. Thus, the total time for this task in an optical interconnect would be:

$$Time_{OCS} = 2N\left(\frac{\max(P \cdot F_i, P \cdot B_i)}{\omega}\right) + 2N \cdot R + 2N \cdot C \quad (2)$$

where $\omega$ represents the advantage of an OCS over a DPS in terms of bandwidth (i.e. $\omega = 2$ allows an OCS to send twice as much data in the same amount of time).

Controller delay can be expressed as $C = 2P + 2S + \theta$, where $\theta$ is the schedule computation time (see 1, 2 and 3 in Figure 4 (b)). From (1) and (2), we can understand the limitations of OCS when we attempt to make $Time_{OCS} \leq Time_{DPS}$. If communication demands are uniform with $M = Fi = Bi$, we can satisfy the inequality above when:

$$M \geq \frac{R + 2P + S + \theta}{P(1 - 1/\omega)} \qquad (3)$$

Figure 6 shows the relationship between OCS/DPS bandwidth ratio and required traffic demands to satisfy Eq.3. The graph assumes fragments of 1500 bytes and 1$\mu$s scheduler overhead[1] ($\theta = 1\mu$s). For currently available optical switching (10$\mu$s nights at 10Gbps), we need about 50 (73KB) fragments in a circuit to make an OCS comparable in performance to a DPS for this type of application. With fast switching (1$\mu$s nights) and twice the speed ($\omega = 2$), this falls to 8 (11KB). Further increase in OCS bandwidth ($\omega$) does not help much as application imposed dependencies restrict the use of extra resources. When comparing a 100Gbps OCS to a DPS of same bandwidth, more than 500 fragments (730KB) are required per request/response messages to achieve comparable performance at 10$\mu$s nights (not shown in Fig. 6). Unfortunately, most Web Application flows are under 10KB [9], limiting their use to DPSs.

The analysis shows why some applications can directly benefit from the high bandwidth capacity of optical interconnects. Distributed frameworks for "Big data" processing, the last application in Figure 5 have higher tolerance to delays. Applications built using the MapReduce paradigm

[1]Although these values are optimistic, they can provide some guidance for future interconnect designs. In a hybrid interconnect for example, flow size can be used to decide if it should be carried over a DPS or OCS.

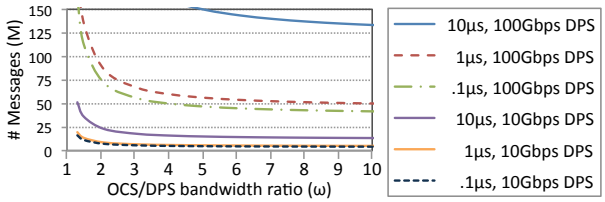

Figure 6: Message fragments required to satisfy Eq.3 under various reconfiguration speeds and DPS baseline bandwidth.

for example are often embarrassingly parallel data-intensive tasks. Since most computations are independent, data transfers can happen concurrently with computation without major performance impact. The only data dependency in this case is the Reduce phase, which depends on outputs from mappers. However, not all Map data is required to start each Reduce task, and some network delays can be tolerated as long as a portion of the data is ready to be processed. Previous work verified that optical interconnects can improve performance of large MapReduce applications [41].

This analysis does not guarantee that latency-sensitive applications can benefit from optical interconnects. Even in the cases where gains are possible, it is difficult to guarantee uniform load distribution ($M = Fi = Bi$) and controller delays of $\theta = 1\mu$s. Previous works either assume a perfect oracle controller or use uniform traffic for their experiments [11, 17, 25, 26, 33]. Note that in hybrid interconnects, the DPS can carry the latency-sensitive traffic, but it is not yet clear how to divide traffic between these two [25]. Therefore, we study situations when an optical interconnect carries all application traffic and what can be done to improve the interconnect efficiency in terms of resource scheduling.

In the next section we discuss our proposed scheduling schemes. They can be used to mitigate the low performance of controllers when facing skewed traffic demands. The schedulers also explore ideas to trade bandwidth for latency, with the goal of increasing interconnect flexibility and efficiency of latency sensitive applications running entirely on an optical interconnect.

# 4. TOPOLOGY SCHEDULERS

A major problem with previous optical switching interconnects is poor scheduling efficiency. Most algorithms in literature, such as TMS [18, 33], Edmonds (used in Helios [17]) and similar others [11, 38], require milliseconds to compute a feasible schedule for the interconnect. All of them share a key characteristic: a single *centralized* controller aggregates demands from all nodes before computing the circuit schedule. This follows the traditional design of DPSs, where a hardware controller has knowledge of all demands and tries to find a maximum (or maximal) matching between Virtual Output Queues (VOQ) and output ports. In DPSs the controller has immediate access to demands, as VOQs and the controller itself are on the same printed circuit board.

When centralized controllers are employed in bufferless optical interconnects, most of the delay in schedule computation comes from demand collection [17, 41]. This happens because the input for schedule computation is often a demand matrix, and delays from several different sources can increase the wait time until this matrix is finally ready for the algorithm. Data extraction delays, asynchrony between nodes announcing demands and pre-processing of demands are some examples. We describe three key requirements for an efficient optical interconnect scheduler:

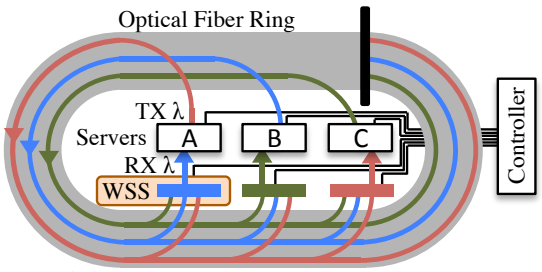


Figure 7: Architecture of Mordia, a fast OCS prototype with switching delay of 10$\mu$s [33].

- Fast schedule computation: a scheduler that allows maximum efficiency of optical interconnects needs to be fast. Preferably, overall reaction times should be lower than $\theta = 1\mu$s. Furthermore, as opposed to high complexity solutions such as the TMS, which has $O(n^{4.5})$ complexity [18], simpler demand collection and algorithms which can be easily implemented in hardware are preferred.

- Bandwidth allocation flexibility: previous schedulers are usually based on finding sequences of static network configurations which use point-to-point links. Ideally, a more flexible bandwidth allocation is preferable, as point-to-point connections and static duty cycles restrict the flexibility of the interconnect.

- Starvation-free: nodes should all be able to use the interconnect. Since computation dependencies are present in most distributed applications, a single starved node can impact the performance of the entire workload [8, 12].

We use these three requirements to guide the design and implementation of two new scheduling schemes. Each scheme is named in terms of its imposed duty cycle (which determines how long a day and night would last) and the network topologies imposed by the interconnect. We briefly describe these concepts before presenting schedulers.

An OCS interconnect state at a given time instant $i$ can be represented by a directed graph $G(i) = G_j = (V, E_j)$. OCS nodes are vertices in $V$ and circuits are edges in $E_j$, with $E_j$ being the $j^{th}$ subset of all $n^2 - n$ possible edges in $E = e_{uv} | u, v \in V, j \in [1, n^2 - n]$. Since these graphs represent the interconnect network topology at a given instant, we call them *topology graphs*. In traditional OCSs, point-to-point connectivity restricts the degree of each vertex to 1, i.e. $deg^+(v) = 1, \forall v \in V$, and each edge $e_{uv} \in E$ has weight $w(e_{uv}) = 1$. The OCS state, or its topology graph, changes when the OCS is reconfigured by the scheduler.

The scheduler dictates the order in which each topology graph is visited over time. We refer to this order as *topology sequence*. The OCS scheduler also decides how long each topology graph is kept active, which is the same as the circuit availability duration. OCS switching speed determines the time interval between each topology graph. These are OCS *day* and *night* durations respectively. Using these concepts, we classify and name the scheduling schemes explored in this work in terms of topology graph and topology sequence.

## 4.1 Static Circuit Duration, Static Topology Sequence (SCST)

The first scheduler we consider has a static circuit duration (i.e. fixed *days* for each topology graph) and uses a static topology sequence. The static topology sequence

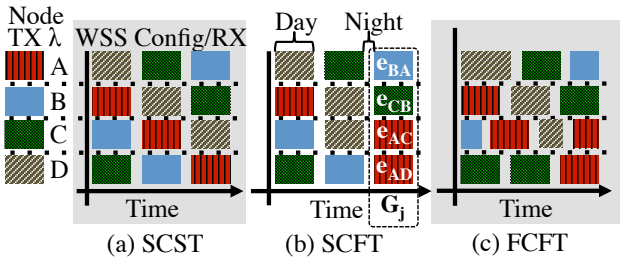


Figure 8: Visual representation of schedulers. Rightmost color is the node transmission wavelength. Colors within the graph have each node WSS configuration over time.

means that the order in which topology graphs are visited repeats in an infinite loop. Since node degree on the topology graphs is restricted to 1, SCST is the same TDMA scheduler used in the Mordia prototype [33]. Also, because the topology sequence does not change at runtime, SCST can achieve the desirable scheduler delays of $\theta = 0$.

We consider SCST as our baseline OCS scheduler because, to the best of our knowledge, it is the only scheduler that was in fact used in a real $\mu$s-capable OCS interconnect [25, 33]. The implementation of SCST requires minimal work from the OCS controller. Each node $v_i$ has $n - 1$ VOQs in their buffers and awaits notifications from an external controller. Each VOQ $j$ corresponds to an edge in $E$ between $v_i$ and $v_j$. At the beginning of each day, the controller reconfigures the OCS interconnect data plane with the desired topology graph $G(i) = G_j$ and broadcasts the identification of the current active topology graph $G_j$ to all nodes. Each node drain the VOQ corresponding to a single edge in $G_j$, until the controller notifies them that the current day is over and $G_j$ is no longer valid. This process repeats indefinitely. The fixed topology sequence (and graphs) makes the communication between controller and nodes simple, as the only information needed is the identification of current graph, and no demand collection is needed. However, the scheduler provides equal throughput among all the nodes, and performs best for all-to-all traffic. Figure 2 exemplifies SCST using 3 nodes and two topology graphs.

Although SCST is general enough to be implemented by several different optical cross-connects, we explain how the Mordia OCS [25, 33] implements it. Figure 7 depicts the implementation of this particular interconnect. Each node transmits (TX) data over a fixed wavelength $\lambda_v$ through a single fiber ring. Wavelengths are multiplexed into the ring using WDM. Each wavelength has the same bitrate. The optical switching happens at the reception (RX), when Wavelength Selective Switches (WSS) from each node "filters" one of the wavelengths ($\lambda_k, k \in 1..v$) in the ring and processes the data transmitted in that specific wavelength. At some point in the ring, each wavelength is attenuated (destroyed) to avoid the optical signal to cycle indefinitely through the ring, interfering with subsequent transmissions. This example depicts a single topology graph with edges $\{e_{AB}, e_{BA}, e_{CB}\}$. Figure 8(a) shows how the interconnect state (i.e. WSS configurations) changes over time, with each vertical set of blocks representing the topology graph and their width the time it is kept active. Note that the width of each block represents time, not data transmission.

The main drawback of SCST is that the delay imposed on each node and the bandwidth provided for each node are fixed. This enforces fairness, but has very small flexibility.

## 4.2 Static Circuit Duration, Flexible Topology Sequence (SCFT)

The key distinction between previous works and our schedulers is the inclusion of optical multicasting into scheduling decisions. We suggest the use of programmable optical components that enable multiplexing and demultiplexing of optical signals [36]. Note that, by optical multicasting, we refer to a medium shared by a group of nodes, where the packets sent are not necessarily destined towards the whole group. Each node within the group will receive the data, but only the destined node will continue processing it. In our schedulers, these components enable topology graphs with vertices of degree different than 1. However, the bandwidth available for each node is fixed. Therefore, the sum of edge weights in these topology graphs is bounded, i.e. $\sum_u w(e_{vu}) \leq 1$. The advantage of multicasting is that nodes can trade off bandwidth for latency, increasing interconnect flexibility by taking advantage of higher node degrees. For example, consider that node A has 1 day worth of data towards B and 1 towards C. This transmission would take 2 days + 2 OCS reconfigurations. If multicasting is available and A can send data simultaneously to B and C, the transmission would still require 2 days, but only 1 reconfiguration.

The second scheduler differs slightly from SCST as there is a larger number of possible topology graphs enabled by an interconnect with optical multicast. SCFT decides the next topology graph based on VOQ demands collected by the controller using a simple, but fast greedy algorithm. The controller keeps bitmaps $B_v$ of size $|V|$ for each node $v$, together with a variable which stores the next preferred bit $\rho_v$ in bitmap $B_v$. VOQ demands are collected from all nodes each day, and if node $u$ announces $VOQ_{uv}$ as non-empty, the controller sets bit $u$ in $B_v$. Before the current day is over, the controller counts the number of bits set in each $B_v$. If their sum is higher than $\beta$, the controller uses SCST for the next $|V|$ days. If not, for each $B_v > 1$, it sets $B_v[\rho_v] = 0$ and advances $\rho_v$ by 1 bit until it finds the next bit set in $B_v$. At the end, it reconfigures the interconnect with $G_j = (V, E_j)$ such that $E_j = \{e_{uv} | \rho_v = u\}$ and broadcasts $G_j$ to all nodes. When the controller falls back to SCST and finishes $|V|$ steps, it reconfigures values of $\rho_v$ to match the last topology graph used, i.e. $\rho_v = u$ for each $e_{uv}$ present in the last topology graph $G_j = (V, E_j)$.

Note that this simple algorithm has similar properties as the RRM scheduling algorithm for DPSs [27]. In the worst case, the value of all $\rho_v$ become synchronized across all $v$'s. This means that the scheduler will progress in lock-step, servicing demands of a single node each day. The parameter $\beta$ helps minimize this problem, falling back to SCST execution, which is optimized for throughput. A reasonable value for $\beta$ is 63%, determined in previous evaluation of RRM [27].

SCFT can be deployed in optical interconnects that use a broadcast-and-select architectures such as Mordia. In such case, the value of $\rho_v$ in the controller dictates the state of each WSS. Figure 8(b) depicts the execution of SCFT over time using such strategy. Note that in such case each vertex $v$ in the topology graph would have $\max(deg^+(v)) = |V|$. Note that topology graphs in SCST can be seen as a bipartite graph matching, but this is not true for SCFT.

SCFT is a greedy algorithm that service only non-empty queues regardless of their size. Although it does not achieve optimal throughput for each controller decision, it is a low-complexity and fast algorithm aiming to minimize delays. Compared to SCST, it eliminates the reconfiguration penalty imposed on small flows towards different nodes, and increases interconnect flexibility by combining them into a single multicast stream.

## 4.3 Flexible Circuit Duration, Flexible Topology Sequence (FCFT)

The use of multicasting allows achieving higher interconnect flexibility. However, a major disadvantage in previous scheduling schemes is the centralized design. In SCFT for example, the controller needs to wait for a certain period of time in order to receive demands from all nodes. Although this also results in predictable duty cycle, as in previous works [25,33], synchronous interconnect reconfiguration can lead to lower utilization. This happens when circuits are granted to nodes which do not have enough demand to utilize all bandwidth available in a single circuit/day. We explore a solution to this problem by distributing scheduling decisions across all nodes, removing the restrictions imposed by a centralized approach. This scheme provides flexible duration of circuit availability and flexible topology sequence. This resembles SCFT, but every topology graph serviced by the OCS can vary in duration. OCS reconfigurations (nights) affect only circuits being reconfigured.

FCFT can also be implemented in a broadcast-and-select optical interconnect as the one depicted in Figure 7. However, it requires each node to store interconnect state information and participate in scheduling decisions. We suggest the use of a general purpose DPS, with lower bandwidth capacity than the OCS for the control plane, as nodes would require to communicate among themselves to perform to distributed scheduler decisions. Although such control plane architecture adds additional cost compared to using a centralized controller with a simpler, specific purpose digital control plane, we believe the benefits in terms of interconnect flexibility and efficiency would compensate the higher cost of the interconnect.

Our implementation of FCFT can be described as follows. Each node keeps a single bitmap $B$ of size $|V|$, which indicates advertised node demand, and a variable $\rho$, which indicates the next preferred node from which the current node would want to receive data. Whenever a node $v$ has data to sent, it simply advertises its demand by sending a broadcast message in the control plane with a bitmap $D$ of size $|V|$, with bit $u$ set case $VOQ_{vu}$ is non-empty. Upon receiving a message from $v$, node $u$ sets the bit $v$ in its bitmap $B$. If $u$ has $\rho = v$, or if there is no other node with demand towards $u$, edge $e_{vu}$ would be added to the topology graph, representing a new circuit between $v$ and $u$. We impose a maximum duration for such connection to avoid starvation. When the current day is over, or when $v$ has no more data to send, node $u$ remove $e_{vu}$ from the topology graph and advance $\rho$ by 1. In a real scenario, with an interconnect as the one depicted in Figure 7, the control of each WSS would be independent as opposed to centralized, and nodes would decide which optical wavelength they would be "listening" to. Figure 8(c) contains a representation of FCFT.

Compared to SCST and SCFT, FCFT improves flexibility using asynchronous scheduling. The use of optical multicast still benefits multiple short flows with distinct destinations as in SCFT, but asynchronous scheduling allows nodes to reassign interconnect resources independent of a controller. This means nodes can adjust day durations according to

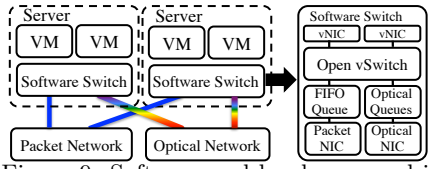


Figure 9: Software and hardware used in experiments.

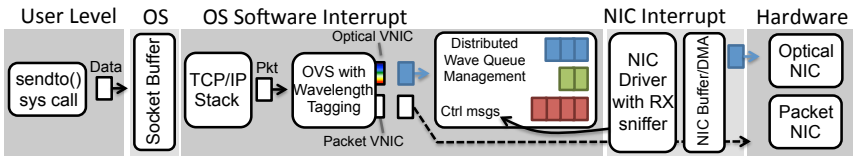


Figure 10: Virtual Switching and wavelength-based queue management used to emulate OCS interconnect. A server synchronizes all using broadcast messages.

their traffic demands, resulting in higher flexibility and utilization. The added flexibility comes at the cost of the additional communication costs.

# 5. IMPLEMENTATION

We used OCSEMU [35], an optical interconnect emulation platform, to explore proposed schedulers and application performance on a fast optically switched environment. The reasons for such strategy is twofold. First, we wanted to experiment application performance in an actual cluster, with unmodified servers, applications, operating systems, and network stacks. This allows us to reproduce realistic scenarios, capturing interactions between all subsystems of a computing cluster, such as interactions of applications and OS network stack with a circuit-like network transmission. Second, limited access to hardware components and the high effort to build special purpose OCS prototypes prevented us to quickly deploy and experiment new scheduler ideas. We briefly describe OCSEMU and the implementation of proposed schedulers. A more detailed description and evaluation can be found in the OCSEMU publication [35].

The emulation environment runs on top of a traditional packet network and is implemented entirely in software. Figure 9 gives an overview of the experimentation setup. Distributed coordination and fine grain queue management are used to emulate $\mu$s circuit switching behavior. OCSEMU code is placed both within and right before the NIC driver to achieve low-latency queue management. Combined with CPU core isolation (e.g. *isolcpu* parameter for Linux Kernel), OCSEMU has minimal intervention of the OS scheduler and ensure high precision in packet transmission times.

Figure 10 gives an overview of a packet's life from an application-level *send()* request to the Network Interface Card (NIC) transfer in OCSEMU. Before a packet is enqueued for transmission, a virtual switch decides what to do with it according to its headers. We use Open vSwitch to act as an extended OpenFlow switch for hybrid packet-optical networks. It decides 1) whether each network flow is transmitted through the packet or the emulated optical network and 2) if optical, which *"optical channel"* would carry this flow. In a real optical interconnect, an optical channel could be either a circuit, a specific wavelength (when the transmitter is equipped with tunable lasers) or a time slot, for TDMA interconnects. In our case, an optical channel is an edge in the current topology graph. We extend forwarding rules in the OpenFlow protocol to accommodate 2). After software switching, packets are enqueued into VOQs. OCSEMU listens for synchronization messages from an external controller, announcing the availability of optical channels. These messages are captured with a sniffer inserted into the NIC driver for minimal delay. Fine grain, $\mu$s control over NIC transmission is achieved with a queue management algorithm which implements both a packet pacer and leaky bucket. The token scheme guarantees that an emulated circuit carries only the amount of traffic dictated by the emulated optical channel.

A major challenge in emulating a $\mu$s scheduler is that transmission across all end-points need to be synchronized with the availability of circuits. To achieve synchronization, OCSEMU uses an external server which broadcasts time information periodically. For centralized schedulers, this server also acts as the network controller, implementing scheduling algorithms. One server CPU core is allocated to the controller, and it stays in a busy loop announcing circuits at specific times according to the software logic. We also tried to implement the controller in a FPGA, but found that the software controller achieves similar precision and is easier modify/extend.

## 5.1 Scheduler Implementation

***Static Circuits, Static Topology Sequence (SCST):*** SCST implementation is straightforward using OCSEMU. Each node keeps a table which maps topology graph id to edges (VOQs) which are allowed to transmit data according to the topology graph. The controller simply broadcasts the topology graph id, together with a single binary variable, which indicates the beginning of a day or a night. Upon the notification of a new day, nodes transmit data from the VOQ associated with that day, subject to the queue draining policy. At nights, nodes stop transmitting data. The controller repeats the topology sequence indefinitely.

***Static Circuits, Flexible Topology Sequence (SCFT):*** SCFT implementation is similar to SCST. However, tables which map topology graph ID to edges are much larger, because $0 \leq deg+(v) \leq |V|$. When $deg+(v) > 1$, we service active edges, or VOQs, using Round Robin. Nodes use the packet switched network during days to announce non-empty VOQs, which are encoded in a single control message as bitmaps of size $|V|$. The controller decides the next topology graph and broadcasts its id as in SCST.

***Flexible Circuits, Flexible Topology Sequence (FCFT):*** In FCFT we use the external controller for synchronization, but it has no participation in scheduling decisions. We configure the controller such that it sends synchronization messages periodically, with interval between message equals to the expected night duration of the interconnect. Changes in the topology graph are made between two nodes, similar to the MACAW used in wireless networks. When a node has non-empty VOQs, it uses the packet network to broadcast a "Request to Send" message with a bitmap indicating VOQ states. When a node decides to "listen" to a request according to the algorithm described, it replies using an "OK to send" message. Night durations are enforced using synchronization messages from the external controller. Once a node receives an "OK to send" message, it waits for the next $\phi$ synchronization messages to start transmitting. This guarantees that nights are at least as long as $\phi$ times the expected duration of nights.

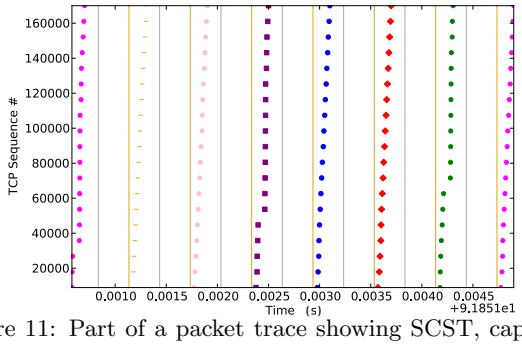

Figure 11: Part of a packet trace showing SCST, captured at a node which sends TCP streams to multiple other nodes, each in a different color (with day=night=300$\mu$s).

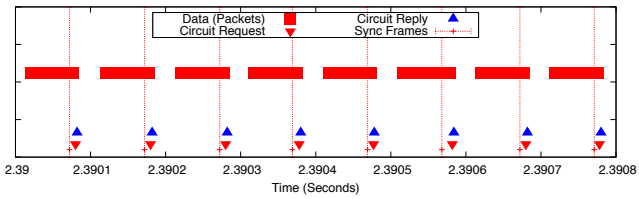

Figure 12: Part of a packet trace showing FCFT in practice handing data transmission from node A to B, collected at node B (with day=100$\mu$s and night=0$\mu$s).

Figures 11 and 12 show examples of how the interconnect behaves in practice using implementations of SCST and FCFT respectively. For Fig. 11 the interconnect is configured with SCST and all nodes send TCP traffic to each other. Days and nights are set to 300$\mu$s. The figure shows (relative) TCP sequence numbers to different destinations in a packet trace collected at one node. Data to each destination is transmitted in a distinct day, and nights (no data transmitted) are enforced using the emulation environment. Vertical bars show day (gold) and night (gray) frames.

Figure 12 shows the behavior of FCFT while server A transmits UDP data to server B. The network trace, collected at node B, contains synchronization messages (shown as vertical bars), scheduling messages and squared dots representing data reception. For this trace we set $\phi = 0$, so that nodes negotiate circuit allocation as soon as synchronization messages are received, and start transmitting as soon as they receive an "OK to send", using $night = 0\mu$s.

# 6. EVALUATION

Our evaluation focus on schedulers and their impact on application performance. We evaluate our ideas in a testbed with 20 HP DL380G6 servers with Intel E5520 CPUs, 24 GB of RAM each, 500 GB 7,200 RPM disks, and Myricom PCI-8B2-2S+E NICs which operate at 10 Gb/s. Each server is connected to multiple networks for management, control and data transmission. A 1 Gbps network is used for management. For data, we use a Cisco 5596UP 10 Gb/s switch, which interconnects all 20 servers.

Our evaluation seeks to answer the following questions:

- How does oversubscription impact application performance? Our goal in this case is to quantify the impact of degraded network offerings to applications. This allows us to understand the possible benefits that could be achieved using an efficient optical interconnect. In other words, if there is no significant room for improvement, there is no need to upgrade the interconnect.
- What is the application performance with current circuit switched prototypes, and how does it compare to a similar digital interconnect of same bandwidth capacity? This represents an evaluation of SCST, which, to the best of our knowledge, is the only scheduling scheme that was actually used in a real fast OCS prototype [25,33]. We also seek to understand the impact of trading off bandwidth for latency, and how it impacts application runtime.
- Last, we explore how different scheduling schemes compare to each other, and how this impacts application runtime, since this is an indicator of the overall energy consumption of the entire cluster.

We start by describing the applications used in our experiments, and the reasoning why we chose them. The subsequent sections explore each one of the topics listed above.

## 6.1 Applications

We deployed and experimented several applications in our testbed, from "Big data" frameworks, to latency-sensitive benchmarks and Web Applications. The list includes Redis (in memory key value store/database), Rubis (auction store benchmark, such as eBay.com), the HiBench Suite (a set of applications for Hadoop, such as PageRank and K-Means) and Spark benchmarks (an in-memory MapReduce framework) and several applications from the NAS Parallel Benchmarks (NPB). However, in this evaluation we focus primarily on applications which perform distributed computation which 1) uses solely main memory as primary storage and 2) run machine code instead of interpreted code.

We decided to limit our scope to these applications for a few reasons. First, we found that "Big data" applications are often limited by disk. Unless a high throughput storage solution is used, the network is not going to be a bottleneck or impact the application significantly. The disk is going to be the main limiting factor. In previous works, application speedups using optical networks could only be achieved when the network was severely throttled [41]. Instead, we compare application performance using an optical interconnect of *same bandwidth* as the target digital packet switched interconnect. This aligns well with our goal, to determine if optical interconnects can be as efficient as a digital switched interconnect for latency sensitive applications.

Second, application frameworks which use virtual execution environments, such as the Java Virtual Machine (JVM), impose a lot of extra memory and computing overhead, as well as extra logic to run the computation workflow. In Java, for example, multiple factors can increase memory consumption, such as the use of inefficient data structures (i.e. an empty linked list still consumes memory to keep its state, and an Integer requires more than a few bytes). Garbage collection also inflates memory usage. We observed several of these problems in Spark applications, and verified that a 10Gbps network is far from being a limiting factor in our testbed, which agrees with other researchers' observations [28]. We also draw our attention away from Web Applications for the reasons described in Section 3.

We select multiple distributed applications in the NAS Benchmarks Suite (NPB), which process in-memory datasets in an iterative fashion. These benchmarks resemble popular applications such as K-means and Page Rank, which process large amounts of data iteratively until clusters are sufficiently good or the graph of ranks converges to a certain

Table 1: Description and communication characteristics of applications used in the experiments

| Short name | Description | Communication | Traffic Volume | Procs |
|---|---|---|---|---|
| MG | Approximate solution to a 3D discrete Poisson Equation | Regular | High | 32 |
| CG | Estimates the smallest eigenvalue of a large sparse matrix | Irregular | High | 32 |
| DT | Data transfer benchmarks based on a graph structure | Irregular | High | 85 |
| EP | Embarrassing Parallel, independent computation | Regular | Low | 32 |
| IS | In memory sorting | Irregular | Low | 32 |
| LU/BT/SP | Iterative algorithms for fluid dynamics applications | Regular | High | 25 |

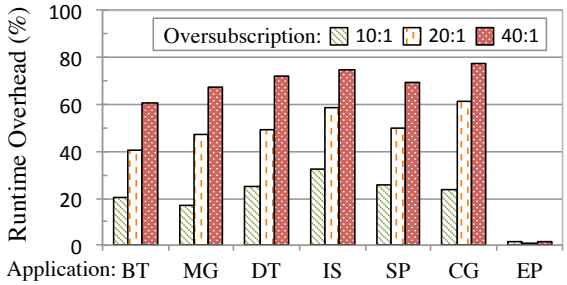

Figure 13: Overhead of distributed applications on oversubscribed networks (compared to optimal 1:1 runtime).

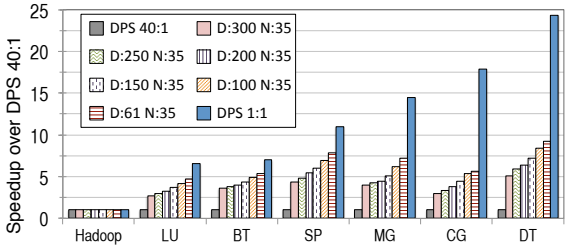

Figure 14: SCST speedup as function of day length, compared to the baseline runtime in a 40:1 oversubscribed DPS

state, respectively. With entire datasets fitting in memory, the network is more likely to become a bottleneck. In fact, growing interest in memory-based storage [32, 37, 40], highlights the need of better interconnects.

NPB uses Message Passing Interface (MPI) and is composed by several distributed algorithms, which make use of several types of collective operations (Broadcast, Multicast, Reductions, etc.). However, these are usually translated to point-to-point connections by the operating system network stack, making them similar to regular applications. Table 1 provides a brief description and communication patterns present on each of the applications we study. Since NPB is a widely used benchmark, we refer the reader to previous publications on characteristics of NPB applications for more details on traffic characteristics [13, 15]

## 6.2 Oversubscription vs Application Runtime

For all experiments, we run each application using all servers, and present results as an average of multiple runs. For this experiment in particular, we adjust network oversubscription ratio for each set of runs. We emulate interconnect oversubscription by rate limiting each link that connect servers to the interconnect. Since our maximum network speed is 10Gbps, this is the bandwidth available for the no-oversubscribed case (1:1 ratio)[2].

We measured total application runtime under no oversubscription (without rate limiting) and compared this result to the runtime of the same application under different oversubscription ratios. Based on application runtimes, we can determine the overall overhead caused by limited network IO performance and the extra energy needed to accomplish the same amount of work. Figure 13 shows this overhead for all applications, subject to an oversubscription ratio ranging from 10:1 to 40:1. As most benchmarks experience up to 40:1 oversubscription, average energy consumption increases from 50% to 80%. Note that the EP benchmark does not show significant difference, as it is strictly compute-bound and has very little communication. Moving forward, we decided to remove it from the set of results.

Note that performance penalty with our imposed oversubscription is "optimistic" for two reasons. First, the number of processing nodes for each application is higher than the number of servers, and multiple nodes can be allocated to a single server. This means that data transfer between two different nodes can occur locally (without going through the actual physical network) and do not suffer oversubscription. Furthermore, we rate limit just outgoing transfers. This means that oversubscription is present in transmission, but a single server may receive packets at full rate.

## 6.3 Responsiveness, Throughput and Latency

The tight dependency between network and computation in latency-sensitive applications makes responsiveness more important than throughput. Thus, we explore how applications behave with the simple scheduler used in current fast OCS prototypes (SCST), evaluating how the choice of design parameters affect application performance.

We take as baseline for comparison the execution time of each application facing network oversubscription of 40:1. We then use this value to compute the speedup achieved with an improved interconnect for each different scenario. (i.e. if the runtime is 1/2 of the baseline, then speedup=2). We compare application performance under 40:1 oversubscription, no oversubscription (1:1 ratio) and SCST under different values for day and night. Note that the comparison between the application running with a 1:1 DPS and the OCS represents the efficiency of the OCS compared to a DPS with same capacity.

Figure 14 shows the results for this experiment with a fixed night duration of 35$\mu$s and days ranging from 61–300$\mu$s[3]. We add a bar for the 40:1 oversubscribed DPS (base comparison) scenario for visual reference. The picture shows that speedups are consistent for most applications, even with application presenting completely different traffic patterns and distinct throughput demands. Note that Hadoop applications in the HiBench are mostly limited by

[2]An analysis using a 100 Gbps would be preferred, but we do not have access to higher speed switches and thus draw conclusions in a downgraded testbed [25]

[3]We use 61/35$\mu$s as minimum day/nights to match the limitations of a real $\mu$s OCS [33]

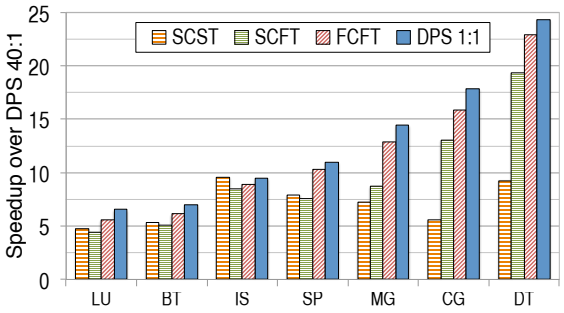

Figure 15: SCFT and FCFT speedup compared to the baseline runtime in a 40:1 oversubscribed DPS

disk throughput in our testbed, leading to no speedup for improved network configurations, as mentioned before.

At first glance the results indicate that a non-oversubscribed DPS is far better than current OCS prototypes, offering speedups as high as 24.3× for the distributed graph-based processing benchmark (DT). When using an OCS with current link scheduling (SCST), the overall speedup is far off than in a DPS, with values ranging from 2.7× to 9.25×. The OCS interconnect duty cycle varies from 90%-63% with days of 300 $\mu$s to 61 $\mu$s, respectively. Under the worse duty cycle SCST offers the lowest RTT, resulting in gains of 9.25×.

These results are far from good, and indicate that current prototypes are not suitable for latency sensitive applications. One major cause is that the scheduling wastes significant bandwidth capacity by not leveraging knowledge of demands, and this results in increased interconnect RTT due to long days for small loads. At low network demand, circuits will often become idle for a fraction of the their availability, preventing other important flows from making progress. This inefficient resource distribution results in performance degradation for iterative applications.

On the other hand, high network demand, but uneven, might also lead to inefficiencies. This happens because iterative applications often have data dependencies on individual flows, which prevents them to advance to future iterations. Furthermore, a large chunk of data may transmit over several days, causing the application to wait for multiple rounds of SCST to make progress. Uneven traffic patterns and misbalanced demands are common, and one example of such case is the graph processing benchmark ($DT$). Bursts of network demand for short durations interact poorly with this scheduler, which can not reassign bandwidths quickly to adverse demands. These issues make the case for a scheduler with greater flexibility, which supports demand-aware network resource provision at shorter timescales.

## 6.4 Centralized vs. Distributed Schedulers

In this last set of experiments we compare OCS interconnect efficiency using distinct schedulers, but with same parameters for day and night durations. As in the previous set of experiments, we use each application execution time under 40:1 oversubscribed DPS interconnect as a baseline, and present the speedup achieved with different schedulers.

Figure 15 shows the results of experiments with both demand-aware schedulers, SCFT and FCFT, implemented in our testbed. In all scenarios using the emulated OCS, days lasts up to 70$\mu$s and nights 35$\mu$s.

These results show that SCFT greatly outperforms the schedule in current prototypes. Furthermore, it demonstrates that demand aware scheduling can be made in short time scales of a few $\mu$s for mid-sized interconnects. Since SCFT provide resources based on demand, it prevents misallocating resources to nodes which have no demands. Also, decisions are made on a day-by-day basis. This means that SCFT can be modified to adjust its scheduling policy based on instantaneous changes to demands, resulting in a more responsive and flexible interconnect than other which produce week-long schedules, such as TMS [18].

However, when compared to a DPS of same bandwidth, we can see that SCFT does not offer comparable performance. The problem with SCFT, and other centralized controllers, is that the scheduling decision is made synchronously across all nodes. This means that if a node finishes its transmission and has no extra data to send through its currently assigned circuit, it has to wait for the next day, when the controller reconfigures the interconnect with the next topology graph. It can not move forward with data transmission across other nodes until the controller takes the next action. This is the case when a distributed OCS scheduler such as FCFT would outperform a centralized controller.

From Figure 15 we can also see that FCFT outperforms SCFT significantly, with average performance improvement of 18.3%. The biggest improvement is for $DT$, 22.9× (2.44× over SCST). Furthermore, the figure shows that the improvement is close to a DPS of same capacity. This experiment thus demonstrates that OCS scheduling can have comparable performance as a DPS for latency sensitive applications, but careful circuit scheduling is necessary to increase interconnect efficiency and flexibility.

# 7. RELATED WORK

**Oversubscription:** there are several proposals to address network bottlenecks. Scaling-out the network to offer full bisection bandwidth [5, 20] is an expensive alternative [33], and oversubscription is still common for big computing clusters [14]. Software can be used to mitigate the problem. Frameworks, middlewares and Virtual Machine (VM) managers can be made aware of network bottlenecks and minimize communication across overloaded links [29]. Others propose network reservation schemes to explicitly allocate resources for applications [34]. Optics is a promising alternative to solve the problem [11, 17, 26, 33, 41], and we contribute with ideas to improve current OCS schedulers.

**Energy Efficiency:** dynamically adjusting the speed of links according to demand can improve energy efficiency of DPSs [4]. However, reaction times of a few $\mu$s are needed to avoid impacting application performance [4]. Another approach is to turn off idle links entirely, as in ElasticTree [22].

**Optical switching:** initial use of dynamic optical switching suggested complimenting a bandwidth-limited DPS interconnect by offloading *hotspots* of traffic to an OCS interconnect [6, 17, 41]. Hybrid packet-optical networks can offer high bandwidth capacity at low power cost, but initial designs were too slow for dynamic traffic patterns [11, 17]. Mordia [19,33] demonstrated faster OCSs, which opened the possibility of revisiting previous findings. We explore application performance on such fast OCS interconnects and show how their scheduling schemes can be improved.

**Schedulers:** Multiple algorithms were proposed for input queued DPSs, such as the RRM and iSLIP [27]. However, slow switching speeds of OCSs and lack of instantaneous access to demands complicates their use in an optically switched environment. For OCSs in particular, the

closest work to ours is the TMS scheduling algorithm [18]. However, TMS was never used in practice [33] due to its high computational cost $O(n^{4.5})$ and long reaction times. We describe simpler algorithms and strategies to estimate demand and evaluate our ideas with real application traffic.

# 8. CONCLUSION

This work evaluates the performance of distributed applications running on an emulated OCS environment with three distinct schedulers. Compared to traditional electronic packet switches, OCS offers much higher power efficiency, but the technology still pays the price of decreased flexibility. We verified that faster OCS switching is a promising energy efficient technology when compared to digital switching. However, it needs increased resource sharing and better schedulers, like the ones presented in this paper, to efficiently forward traffic from a broader class of applications. For the applications evaluated, our proposed schedulers were able to provide up to 2.44x improvements in performance over schedulers used in current $\mu$s OCS prototypes.

## Acknowledgements

This work was partially supported by the CIAN NSF ERC under grant #EEC-812072 and EU ACROSS program.

# 9. REFERENCES

[1] Arista 7124FX Application Switch Datasheet.
[2] Huawei CE12812 Switches Datasheet.
[3] Mellanox sees rapid increase in 40g revenue http://seekingalpha.com/article/117329-mellanox-sees-rapid-increase-in-40g-revenue.
[4] D. Abts et al. Energy Proportional Datacenter Networks. In *Proc. International Symposium on Computer Architecture*, pages 338–347, 2010.
[5] M. Al-Fares et al. A Scalable, Commodity, Datacenter Network Architecture. In *ACM SIGCOMM*, 2008.
[6] K. J. Barker et al. On the Feasibility of Optical Circuit Switching for High Performance Computing Systems. In *ACM/IEEE Conference on Supercomputing (SC)*, 2005.
[7] L. A. Barroso and A. Hölzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 40(12):33–37, 2007.
[8] H. Bazzaz et al. Switching the Optical Divide: Fundamental Challenges for Hybrid Electrical/Optical Datacenter Networks. In *ACM SOCC*, 2011.
[9] T. Benson et al. Network traffic characteristics of data centers in the wild. In *ACM Internet Measurement Conference (IMC)*, 2010.
[10] J. E. Berthold. Optical networking for data center interconnects across wide area networks. In *IEEE Symposium of High Performance Interconnects*, 2009.
[11] K. Chen et al. OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility. In *NSDI*, 2012.
[12] M. Chowdhury et al. Managing data transfers in computer clusters with orchestra. In *ACM SIGCOMM*, 2011.
[13] Y. Dotsenko. *Expressiveness, programmability and portable high performance of global address space languages*. ProQuest, 2007.
[14] Introducing the next-generation Facebook data center network, Nov 2014. `https://code.facebook.com/posts/360346274145943/`.
[15] A. Faraj et al. Communication characteristics in the nas parallel benchmarks. In *IASTED PDCS*, 2002.
[16] N. Farrington and A. Alexey. Facebook Data Center Network Architecture. In *OSA OFC*, 2013.
[17] N. Farrington et al. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. In *ACM SIGCOMM*, 2010.
[18] N. Farrington et al. Hunting mice with microsecond circuit switches. In *ACM HotNets Workshop*, 2012.
[19] N. Farrington et al. A multiport microsecond optical circuit switch for data center networking. *IEEE Photonics Technology Letters*, 2013.
[20] A. Greenberg et al. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM*, 2009.
[21] S. Han et al. Monolithic 50x50 mems silicon photonic switches with microsecond response time. In *OSA Optical Fiber Conference (OFC)*, 2014.
[22] B. Heller et al. ElasticTree: saving energy in data center networks. In *Proc. USENIX NSDI*, 2010.
[23] Wall Street's Quest To Process Data At The Speed Of Light. http://informationweek.com/wall-streets-quest-to-process-data-at-the-speed-of-light.
[24] R. Kapoor et al. Chronos: predictable low latency for data center applications. In *ACM SoCC*, 2012.
[25] H. Liu et al. Circuit switching under the radar with reactor. In *ACM/USENIX NSDI*, 2014.
[26] Y. J. Liu et al. Quartz: a new design element for low-latency dcns. In *ACM SIGCOMM*, 2014.
[27] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 1999.
[28] F. McSherry. Scalability! But at what COST? http://www.frankmcsherry.org/graph/scalability/cost/2015/01/15/COST.html.
[29] X. Meng et al. Improving the Scalability of Data Center Networks with Traffic-Aware Virtual Machine Placement. In *IEEE INFOCOM*, 2010.
[30] D. A. B. Miller and H. M. Ozaktas. Limit to the Bit-Rate Capacity of Electrical Interconnects from the Aspect Ratio of the System Architecture. *Journal of Parallel and Distributed Computing*, 1997.
[31] J. Mudigonda et al. Taming the flying cable monster: A topology design and optimization framework for data-center networks. In *USENIX ATC*, 2011.
[32] J. Ousterhout et al. The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM. *ACM SIGOPS OSR*, 2010.
[33] G. Porter et al. Integrating Microsecond Circuit Switching into the Data Center. In *SIGCOMM*, 2013.
[34] H. Rodrigues et al. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. *USENIX WIOV*, 2011.
[35] H. Rodrigues et al. Accurate emulation of fast optical circuit switches. In *IEEE International Conference on Communications (ICC)*, 2015.
[36] G. N. Rouskas. Optical layer multicast: rationale, building blocks, and challenges. *IEEE Network*, 2003.
[37] S. M. Rumble et al. It's time for low latency. In *USENIX HotOS Workshop*, 2011.
[38] A. Singla et al. Proteus: A Topology Malleable Data Center Network. In *ACM HotNets Workshop*, 2010.
[39] A. Singla et al. Jellyfish: Networking Data Centers, Randomly. In *USENIX HotCloud*, 2011.
[40] D. B. Strukov et al. The missing memristor found. *Nature*, 453(7191):80–83, 2008.
[41] G. Wang et al. c-Through: Part-Time Optics in Data Centers. *ACM SIGCOMM*, 2010.
[42] H. Wang et al. Rethinking the physical layer of data center networks of the next decade: Using optics to enable efficient*-cast connectivity. *ACM SIGCOMM Computer Communication Review*, 43(3):52–58, 2013.
[43] X. Yu et al. Enhancing performance of cloud computing data center networks by hybrid switching architecture. *Journal of Lightwave Technology*, 2014.