Accord: Application-driven Networking in the Datacenter

Seyed Hossein Mortazavi, Hossein Shafieirad, Mahmoud Bahnasy, Ali Munir*
Huawei Technologies
Canada

Yuanhui Cheng, Anudeep Das†
University of Waterloo
Canada

Yashar Ganjali‡
Huawei Technologies
Canada

ABSTRACT

Resource optimization algorithms in the cloud are ever more data-driven and decision-making has become reliant on more and more data flowing from different cloud components. Applications and the network control layer on the other hand mainly operate in isolation without direct communication.

Recently, increased integration between the network and application has been advocated to benefit both the application and the network but the information exchange has mostly been limited to flow level information. We argue that in the realm of datacenter networks, sharing additional information such as the function processing times and deployment data for planning jobs and tasks can result in major optimization benefits for the network.

In this study we present Accord as a Network Application Integration solution to achieve a holistic network-application management solution. We propose a protocol as an API between the network and application then we build a system that uses the processing and networking data from the application to perform network scheduling and routing optimizations. We demonstrate that for a sample distributed learning application, an Accord enhanced solution that uses the application processing information can yield up to 27.8% reduction in Job Completion Time (JCT). In addition, we show how Accord can yield better results for routing decisions through a reinforcement learning algorithm that outperforms first shortest path first by %13.

CCS CONCEPTS

• **Networks** → **Network design principles**; **Data center networks**.

ACM Reference Format:
Seyed Hossein Mortazavi, Hossein Shafieirad, Mahmoud Bahnasy, Ali Munir, Yuanhui Cheng, Anudeep Das, and Yashar Ganjali. 2021. Accord: Application-driven Networking in the Datacenter. In *2021 IEEE/ACM 14th International Conference on Utility and Cloud Computing (UCC'21), December 6–9, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3468737.3494102

*Corresponding author: seyed.hossein .mortazavi@huawei.com
†Work done during an internship at Huawei, Canada
‡Also with University of Toronto.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
UCC'21, December 6–9, 2021, Leicester, United Kingdom
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8564-0/21/12…$15.00
https://doi.org/10.1145/3468737.3494102

1 INTRODUCTION

Modern datacenters use a wide range of techniques and strategies to optimize resource usage and lower costs while providing improved services and guarantees to customers. These optimizations such as improving CPU utilization [19], VM placement [20], VM selection [32], cache placement [34], etc. rely on information that is constantly gathered and transferred from a wide range of components in the cloud. Networks were slower in adapting to this data driven evolution because (I) Network components are rigid to change and difficult to innovate in and (II) Network components and protocols mainly approximate and estimate the state of other components rather than effective information exchange or querying. An example of a component that has limited communication with the network is the user application. Both the application and the network generally operate in isolation [11] and both often go to great measures to guess and predict the state of the other without direct communication.

Information exchange between the network and the application can lead to:

- Application-Aware Networking (AAN): where the network configuration policies can be determined based on the application requirements, intent, and/or properties, to (i) provide performance guarantees to the application, (ii) optimize the scheduling and resource allocation, and (iii) improve network utilization.
- Network-Aware Applications (NAA): where application decisions (such as worker selection for a distributed application) can be based on the link states, congestion or topology design. This will lead to (i) improved application performance, and (ii) cost efficient use of network resources.

Fortunately, advances in network programmability with Software Defined Networking (SDN) and P4 switches [3] has enabled the network to move towards application aware networking and network aware applications due to the centralized logic control and integrated application and network management. In addition, various studies [13, 16, 18, 21, 24, 26] have proposed solutions, protocols and interfaces to connect the applications and the network. These approaches have opened a new paradigm in networks called Network-Application Integration (NAI).

However, to the best of our knowledge, none of these studies focus on cloud datacenter networks and investigate how the collaboration between the network, application and the other components can result in better optimizations for the cloud provider. In addition,

these solutions only suggest exchanging networking information between the application and the network while the network can make more informed decisions based on other information such as application processing and deployment state. Consequently, algorithms that utilize this extra information have not been developed.

We believe that the future intelligent network management systems for datacenters will be built upon automated, open API's that allow information exchange between the network provider and the application. Our vision is that NAI systems will enable AAN and NAA in datacenter environments by exchanging a wide range of data (such as processing, networking, storage and cost) with the application. This information will be used in cross component optimizations that were restricted up to now because of how their different components were either isolated from each other or were only exposed to each other in a limited scope. The cloud provider has control over different types of resources namely processing, execution, storage, and power, so optimizations will result in more efficient and more intelligent services with lower costs that are reactive to changes in the application. For example, the network control layer in a datacenter network can utilize information such as application deployment logic, function processing times, and traffic patterns between the functions to better optimize network resources for scheduling and routing. Similarly, the application can utilize the information such as DCN topology and network link state to better optimize task scheduling. In addition, real-time information exchange between application and network can allow for dynamic adaptation to the network and application state and requirement changes.

In this paper, we propose Accord; an NAI system for datacenters that uses networking, processing and deployment information from the application to make routing and scheduling optimizations in the network. Accord aims to answer the following two questions:

(i) *How should the networking and processing information be communicated over the application network interface* and (ii) *how can the network use this extra information for resource usage optimization?* We suggest a method to capture the intent and state information of applications and the network through the application *job* abstraction and also present two novel approaches that use networking processing and deployment information from the network for routing and scheduling. Using our methods, in addition to reducing cost through network usage optimization, the network can also potentially provide guarantees in form of SLAs to the application. The proposed approaches can benefit from various applications such as distributed machine learning, stream processing, distributed databases, and batch processing.

We implement a prototype of Accord and demonstrate the benefits of Accord in a small environment by exchanging job information and application objective to achieve improved service. Our experiments show that our scheduling algorithm achieves 27.8% improvement in Job Completion Time (JCT) compared to the fair share strategy. Also, we show how our routing algorithm outperforms First Shortest Path First (FSPF) by reducing resource utilization by 13% while increasing the acceptance rate by 8%.

Our contributions can be listed as follows:

- We design and implement an application network interface that enables our vision for application-aware networking by transferring application intents and properties for networking and processing to the network.
- We present a network scheduling algorithm that schedules flows based on application function processing and inter-flow information.
- We propose a deep reinforcement learning algorithm for network routing that uses information from the application to choose optimized paths for flows.
- We implement a prototype of our system and show how the network benefits from the additional information exchange.

The rest of the paper is organized as follows; Section 2 presents the motivation behind this work. The design of our proposed protocol is discussed in Section 3. In Section 4 we present our application-aware networking algorithms for routing and scheduling. Section 6 includes the implementation details or our protocol and experimental results. We conclude the paper in Section 7.

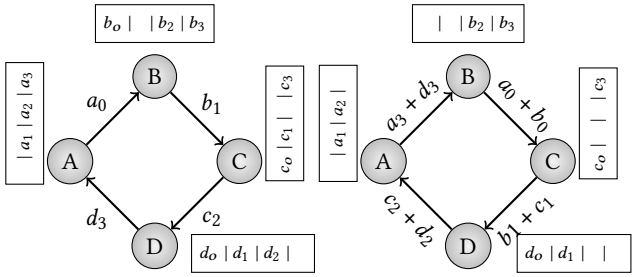
2 BACKGROUND AND MOTIVATION

2.1 Related Work

Several protocols have been created to facilitate the information exchange between the applications (or host) and the network. These approaches generally fall into two categories: (i) Network to Application exposure [2, 24] and (ii) Application to Network exposure [18, 21, 26]. Other approaches such as [9, 17] are only limited to optimizing certain applications (such as MapReduce) or use cases. The Alto protocol [2], defined in RFC 7285, provides the capability to define abstractions in form of network maps and cost maps, and allows users applications to query these maps which can then be used to reduce costs. Perez et al. [24] optimize network views based on network service requirements and suggest network inventory capabilities for distributed cloud environments. To send the application requirements to the network, APN6 [18] uses the programmable space in IPv6. APN6 sends the application identifiers and their requirements along with the packet to the network which then can be used to facilitate service deployment and for the network to provide SLA guarantees. In [26], Schmidt et al. propose *socket intents* which adds socket options to the socket API to allow applications to convey their intents such as information about a flow to the network.

Several frameworks have been developed on how this information can be organized and utilized [16, 23, 24]. In [16], Lachos et al. emphasize on the benefits and potentials of direct communication between the network and the application. In order to realize these potentials, they present PED which is a framework for information discovery and exposure between the network and the applications. PED proposes a unified, abstract representation of network information using mathematical programming constraints, as well as a declarative language for applications to express their intents on discovering network information. It also includes a compiler to translate application intents to constraint programming problems and discover corresponding network information. The idea of PED is further expanded between multi-domain networks by Lachos et al. in [23, 24].

Several studies have suggested NAI for carrier networks. Zhang et al. [35] advocate for a software defined fine-grained QoS framework that enables cross layer application and network integration



(a) Time $t = 0$ **(b) Time $t = 1$**

Figure 1: Ring all-reduce: message exchange

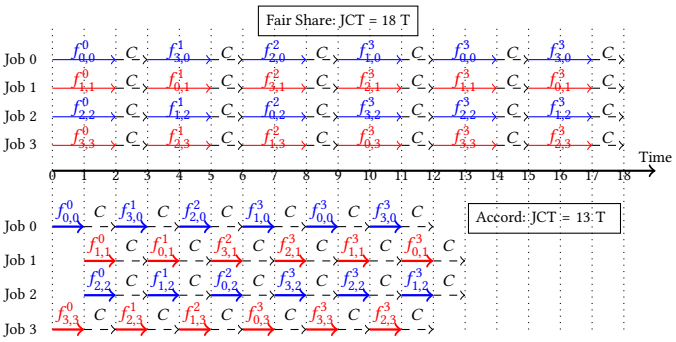


Figure 2: Ring all-reduce scenario

in 5G networks, and in [33] the authors develop a network information exposure system for application in ISP networks using the ALTO protocol.

Various studies have considered the relationship between flows in terms of coflows [1, 5], or in terms of inter-coflow scheduling [6, 7, 29]. These studies do not capture the application intent and requirements specifically and assume the coflow completion time as the application requirement. However, different applications may have different requirements and a lower job completion time does not necessarily translate into better performance for application jobs.

More recently, Jyothi et al. [13] proposed CadentFlow, an application network interface to capture application objectives and intents for each flow and use it to improve the performance of distributed machine learning application.

Limitations: Existing studies only leverage network level information and application processing time information to improve performance as these protocols are generally not designed for data center networks. While using the processing and storage information for managing cloud resources has been extensively researched in various studies [14, 25, 28], to best of our knowledge, there are no studies that use this information for network optimization as the collaboration or data exchange between the network control and the application or other Cloud components [16] has been limited. We argue that using processing data as well as application placement information is essential for further optimizing network resources in data center networks and a tighter coordination with the application can result in many more benefits for the application and network. We show how this extra information can help the network controller in making better decisions for routing and scheduling.

2.2 Example Application NAI

In order to show the network can be optimized by exchanging application information, we present a motivating example: A distributed deep learning application that achieves better results in JCT by exchanging information with the network. A distributed deep learning application involves iterative computation with the parameters on the workers (nodes) being updated and exchanged with each other at the end of each iteration. The ring all-reduce algorithm is one way to exchange parameters among all workers.

In this algorithm, data is partitioned into equal chunks and distributed among processing nodes which we show in Fig. 1. In Fig. 1a the message exchange process at the first cycle where each node transmits a unique part of the parameters to the next processing node. In the next cycle, each node performs the reduce process on received data and its local data. Then it transmits the result to the next node as shown in Fig . 1b (for more details review [22]).

Such a process can be presented as four parallel *jobs* where each job has six dependent flows that form a directed acyclic graph (DAG). Flows are noted as $f^k_{i,j}$ where i is the edge index (e.g., edges $A \rightarrow B, B \rightarrow C, C \rightarrow D$ and $D \rightarrow A$ are labeled 0, 1, 2, 3 respectively), j refers to job number, and k is the time cycle. Each flow $f_{i,j}$ transmits fixed data size S every T units of time with some processing time in between. Flows within the same job have dependency as follows: $f_0 \preceq f_1 \preceq f_2 \preceq f_4 \preceq f_5 \preceq f_6$ (where $f_m \preceq f_n$ denotes f_n depends on f_m). Each node requires a processing time equal to one time unit T. We assume f_0 and f_2 share the same link to induce congestion on that link. The same applies between f_1 and f_3. We compare a network-aware approach with the fair share approach. Note that the goal here is to minimize the job completion time (rather than individual flows or coflows). As shown in Fig. 2, the fair share approach finishes all jobs in $18T$, while a network-aware approach that knows about the dependency between flows for each job, can finish all jobs in $13T$ with around 28% reduction in JCT.

3 DESIGN

In this section, we present the design details of Accord. First, we present the overall architecture and then discuss how it can be used to capture the application intent and state. Later we show how this architecture helps in the scheduling (section 4) and routing (section 5) use cases.

3.1 Overview

In Fig. 3 we show Accord's architecture. Accord uses two key components, a *Driver* module on the application host and a *Broker* on the network controller, to establish a reliable channel for the communication between the network and application. Applications use the local driver on the host to share application information and potentially fetch network state information. Similarly, the network controller uses the broker to fetch application information and share network state with the application.

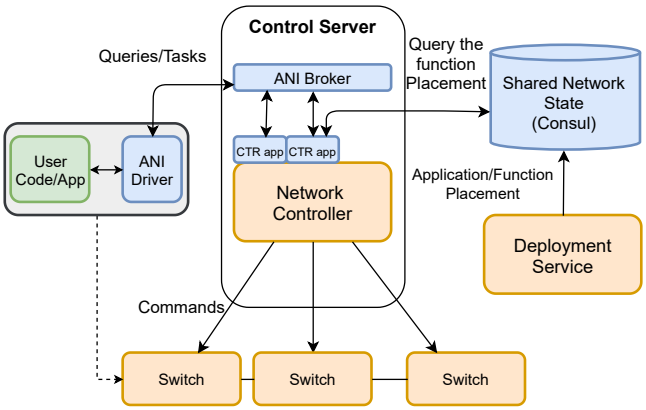


Figure 3: Accord architecture. Accord 's components are shown in blue while the traditional datacenter components are in orange and the application in green

We use a programming interface to facilitate this communication. Programming interfaces are commonly used to access cloud resources (e.g., Boto3 for AWS). The application makes function calls or receives calls back on the local *driver* and the driver forwards the request with other statistical information to the broker. This model hides the communication complexity between the broker and driver. The driver and the broker communicate using the Accord protocol (see § 3.3).

3.2 Capturing Intent & State

We believe that the missing piece in current NAI platforms is the ability to incorporate processing information (intent and state) within their systems. While in general networks, especially geo-distributed networks such information may be hard to obtain, in a datacenter environment, where generally the provider also controls the processing, the application is a tenant and accessing this information is attainable.

In Accord, the applications convey their intent and state through the *jobs* abstraction. Jobs are the basic units of execution defined by the application and are a set of correlated application functions and flow. Jobs are represented by the applications as a Directed Acyclic Graph (DAG) and contain the relationships between flows and application functions. The edge nodes represent the flows and the vertices are the application functions.

Each edge node contains tuples representing the properties for respective flow such as expected flow size, required latency or deadline and the relationship to other flows as well as functions. The objective of each flow can also be expressed in the tuples. This model is similar to CadentFlows proposed by Jyothi et al. [13]. Each vertex contains the estimated function processing time as well as connected flows and the relationship between them. In addition to the mean and variance values for processing times, flow sizes and required latencies, we also include an optional *distribution* field for a more comprehensive characterization of a job.

Accord's architecture is built upon the idea of aggregating and using any data that is available by the application or the cloud provider. While the requested information on the flows and the functions can be extensive, the cloud provider can obtain some of this information through Machine Learning (ML) techniques without direct application involvement. For example, predicting the cost of workloads for function as a service (FaS) platforms [8, 10] and also the flow size prediction [30, 31] is an active field of study. Accord can leverage such information to its use. The network provider also maps between the application functions and the physical machines using application deployment information provided by the cloud provider.

3.3 Protocol

The driver on the application and the broker communicate over the Accord protocol. This protocol is designed to transfer the application intents, requirements and properties to the network and at the same time allow the application to get the network state. The driver communicates with the broker using the REST protocol on ZeroMQ [12]. After the handshaking and authentication phase, all message headers include an application ID and a token that identifies the application to the network. The driver submits *JOB* messages to the broker that include information about the application functions and the expected processing time for each function as well as the expected bandwidth and latency requirements, loss tolerance and deadlines of flows that transfer data between the functions.

4 USING ACCORD TO ENHANCE FLOW SCHEDULING DECISION

In this section, we show how using Accord to convey application intent such as job DAG, flows sizes, and processing time yields better performance in scheduling.

4.1 Example Scenario

Here, we consider a network topology represented as a directed graph $G = (V, E)$, where $v \in V$ represents both network nodes and computational nodes, and $e \in E$ is the set of edges (links between nodes). Each edge e connecting node i to node j is represented by e_{i-j} and assigned with a non-negative capacity $0 \leq C(e_{i-j})$. Each flow $f_{m,n}$ is defined by a tuple $(a_{m,n}, z_{m,n}, t_{m,n}, s_{m,n}, d_{m,n}, c_{m,n})$, where a, z, s, d, c are source, destination, arrival time, size, deadline and processing time at the receiving node, respectively. The deadline defines a minimum limit for the requested rate as $r^{req}_{m,n} \geq s_{m,n}/(d_{m,n} - t_{m,n})$. In addition, flows can't send with a rate higher than what they can generate which is defined using flow size $s_{m,n}$ and the processing time $c_{m,n}$ as follow $r^{req}_{m,n} \leq \frac{s_{m,n}}{c_{m,n}}$.

Flow dependency inside jobs is defined as a list of two-item tuples $\mathcal{D}_{F_m} = \{(f_{m_{n'}}, f_{m,n}), ...\}$. We define $ts_{m,n} = \{0, .., 1, 1, .., 1, .., 0\}$ to be equal to the time slot vector for job m and flow n. Each time slot T is equal to the transmission time of one packet. packet time, minimum divisible number(flow sizes/C)) We also assume that each flow is served in adjacent time slots that start at index x that refers to the first time slot that flow $f_{m,n}$ is being served.

4.2 Problem Formulation

To this end, we design an optimization problem to define an optimal first time slot $x_{m,n}$ and rate $r_{m,n}$ for each flow that minimize the

maximum flow completion time $\tau_{w,n}$. We can formulate the problem as follows:

$$\min_{x,r} \quad \max_{m,n} (\tau_{m,n}) \tag{1a}$$

$$\text{s.t.} \quad x_{m,0} = t_{m,0} \qquad \forall m \in M, \tag{1b}$$

$$(\tau_{m,n} - x_{m,n}) \cdot r_{m,n} = \lceil s_{m,n} \rceil \qquad \forall n \in N, \forall m \in M, \tag{1c}$$

$$\tau_{m,n} \cdot T + c_{m,n} \leq x_{m,n'} \cdot T, \ (f_{m_{n'}}, f_{m_n}) \in \mathcal{D}_{F_m}, \tag{1d}$$

$$\frac{s_{m,n}}{c_{m,n}} \geq r_{m,n} \geq \frac{s_{m,n}}{(d_{m,n} - t_{m,n})} \qquad \forall n \in N, \forall m \in M, \tag{1e}$$

$$\sum_{f^l_{m,n}} (ts_{m,n}[i] \cdot r_{m,n}) \leq 1 \forall i \in \{0, 1, ..k\}, \forall l \in \text{ all links} \tag{1f}$$

As shown in the formulation, constraint 1b states that the first flow starts at the arrival time of the job. In constraint 1c, τ is defined as the index the first available time slot after finishing flow $f_{m,n}$. Hence, this constraint limits the transmission bytes per flow to its size. We also added fake dependency between flows based on their sizes (shorter flows first) to imitate Shortest Job First (SJF) module. All flow dependencies requirements are met using constraint 1d. Constraint 1e ensures that the transmission rate to be greater than the minimum rate that meets the flow deadline. Flow transmission rates are limited by the maximum processing capability of the receiving node. Finally, constraint 1f guarantees that link utilization does not exceed link capacity.

We simplified such a problem considering that flows are sent with maximum link capacity for each reserved time slot as stated below. One can notice that the new constraints; namely 2c, 2d and 2e, simplify the problem as shown in the new problem formulation. Note that the constraint 2e states that all flows that share one link can't use time slots more than their maximum utilization. Therefore, we no longer need to manage each slot in the ts vector. We rather need to manage the starting and ending transmission time for each flow (x and τ).

$$\min_{x} \quad \max_{m,n} (\tau_{m,n}) \tag{2a}$$

$$\text{s.t.} \quad x_{m,0} = T_{m,0} \qquad \forall m \in M, \tag{2b}$$

$$(\tau_{m,n} - x_{m,n}) \cdot C = \lceil s_{m,n} \rceil \qquad \forall n \in N, \forall m \in M, \tag{2c}$$

$$\tau_{m,n} \cdot T + c_{m,n} \leq x_{m,n'} \cdot T, \ (f_{m_{n'}}, f_{m_n}) \in \mathcal{D}_{F_m}, \tag{2d}$$

$$\max_{f^l_{m,n}}(\tau_{m,n}) - \min_{f^l_{m,n}}(x_{m,n}) \geq \sum_{f^l_{m,n}} (\tau_{m,n} - x_{m,n}) \quad \forall l \in \text{ all links} \tag{2e}$$

4.3 Flow Scheduling Algorithm

To solve the optimization problem, we propose Shortest Available Flow First (SAFF), a heuristic scheduler that takes advantage of the knowledge provided by Accord protocol. The SAFF scheduler uses the DAG provided by the application to extract its intent. In this regard, consider the example depicted in Fig. 4 where flows f_1, f_2 and f_3 are served and flows f_4 and f_5 are available and ready to be scheduled. The DAG allows us to know that after serving f_4, f_6 becomes available. Therefore, it is better to schedule f_6 before f_5. Hence, the scheduler avoids possible delays due to the large size of f_5 that might result in delaying future flows and therefore increasing the Flow Completion Time (FCT). Traditionally, such an objective can be achieved using flow preemption. However, with the information available in the DAG such optimized decision can be fulfilled in one step while avoiding extra overhead and delays.

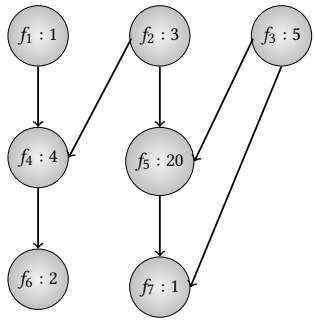


Figure 4: Dependency graph example

To take advantage of available information from the application through the DAG, we design SAFF that considers functions' processing and link utilization for scheduling. The proposed algorithm considers a finite time horizon divided into T time slots, each with length w sec. The proposed approach, shown in Algorithm 1, allocates rates per time slot for all flows.

We start by extracting all flows from the incoming request. Then, we sort all available flows based on both their expected transmission time, motivated by SJF scheduling, and dependency constraints. Flows are sorted using the SAFF scheduling strategy described in Section 4.4. The outputs of Algorithm 1 are: (i) flow rates, and (ii) reserved time slots per flow. For each flow, if there is no feasible rate due to the delay constraint, the available rate is sent to the application to decide whether to accept the assigned rate or to resubmit its request with different criteria (e.g., different service node or different rate).

At each time slot t, the assigned rate for flow f, denoted by r^t_f, is generated as shown in line 9 considering the processing time P_f, remaining flow size S_f and the available bandwidth for that flow at the mentioned time slot. Link sharing is also allowed among flows sharing links which helps better rate allocation of flows. The remaining size of flow is updated at the end of each time slot (line 10). The *AssignStartingTime(·)* function assigns the first time slot that a flow can be scheduled based on other flows it depends on as well as available bandwidth.

Given the flow size and considering that only one congested link exists per datapath, Shortest Flow First (SFF) is shown to result in the optimal average FCT. However, considering the dependency constraints, the scheduling problem becomes a challenging NP-hard combinatorial problem. Therefore, we propose SAFF, a centralized greedy scheduling algorithm, which by taking the advantage of DAG, uses both size and flow dependencies for the scheduling problem of our interest to minimize the FCT.

Considering the dependency between flows, an efficient strategy among others is to first, sort all flows based on their sizes. Second, relocating flows based on their dependency constraints as dependent flows must be scheduled after the flows they depend on. Although

Algorithm 1: Scheduling Algorithm

1 **Algorithm** `Main(`*NAI Message*`)`**:**
  Data: Network link occupancy, New job (NAI message)
  Result: Flow Scheduling Table (FST)
2 $F \leftarrow$ flows from NAI message;
3 $L \leftarrow SAFF(F)$;
4 **forall** $f \in L$ **do**
5 $R = AssignRoute(f)$;
6 $\Gamma = FlowsOnPath(R)$;
7 $t = AssignStartingTime(f)$;
8 **do**
9 $r_f^t = min(\frac{S_f}{P_f}, AvailableBW(\Gamma(R))$;
10 $S_f \leftarrow S_f - r_f^t \times w \; t \leftarrow t + 1$
11 **while** $0 \leq S_f$;
12 **Function** `SAFF(`F`)`**:**
  Data: Network link occupancy, New job (NAI message)
  Result: Flow scheduling sorted list L
13 $L \leftarrow \{\}$;
14 Initialize S with active flows;
15 $F \leftarrow$ flows from NAI message;
16 **do**
17 $\mathbb{A} \leftarrow AvailableFlows(F)$;
18 $f_s = ShortestFlow(\mathbb{A})$;
19 $L.append(f_s)$
20 **while** $AvailableFlows(F)! = \emptyset$;
21 **return** L;

the mentioned two-stage sorting algorithm is simple and efficient, it does not always result in the best average FCT.

For example, Fig. 1 demonstrates (white nodes) the output of such a simple strategy for the DAG is demonstrated in Fig. 4. One can notice that f_5 of size 20 unit delays smaller size flows f_4 and f_6 while if f_4 is scheduled before f_5, the average FCT decreases. This motivates us in proposing the Shortest Available Flow First (SAFF) scheduling strategy. SAFF strategy considers both flow sizes and dependency constraints into account simultaneously. In what follows, we show how this greedy approach can improve the scheduling process, when the DAG is available.

4.4 Shortest Available Flow First SAFF

The SAFF strategy, as represented in Algorithm 1, is a centralized greedy scheduling approach that at each step, a set of available flows that can be scheduled according to DAG, denoted by $\mathbb{A}$, is generated. The set $\mathbb{A}$ is updated at each step based on flows ready to be scheduled according to both available bandwidth and also other flows they depend on. All flows in $\mathbb{A}$ are sorted based on their expected transmission time and the flow with the shortest transmission time (f_s) is selected. This procedure repeats until all flows are selected and listed in the scheduling table L.

In the example depicted in Fig. 4, we assume flows f_1, f_2 and f_3 are available in the first step and f_1 has the smallest transmission time (considering there is only one congested link per data-path). Therefore f_1, is selected to be the first flow to be served. The set $\mathbb{A}$ then is updated as $\mathbb{A} = \{f_2, f_3, f_4, f_5\}$ and flow f_2 and after that f_4 are selected. In this way, f_4 is selected before f_5 resulting in the final sorted list of flows depicted in Table 1 using SAFF (gray nodes). The sorted list of flows using SAFF is then used as explained in lines 3 – 15 of Algorithm 1 for generating flows' assigned rates according to the available link capacities, flows' processing time and size.

$f_1 : 1$	$f_2 : 3$	$f_3 : 5$	$f_5 : 20$	$f_7 : 1$	$f_4 : 4$	$f_6 : 2$
$f_1 : 1$	$f_2 : 3$	$f_4 : 4$	$f_3 : 5$	$f_6 : 2$	$f_5 : 20$	$f_7 : 1$

Table 1: Scheduling for the example depicted in Fig . 4. Shortest flows algorithm (white) vs the SAFF algorithm (gray)

5 USING ACCORD TO ENHANCE ROUTING

In this section, we present a routing optimization algorithm that uses information provided from the applications by Accord.

We propose a machine learning-based module to make routing decisions knowing the network state and application intent. This model uses Deep Reinforcemnet Learning (DRL) to optimize the routing decisions where we adopt a simple Deep Q-Network (DQN) as our agent. Our model is inspired by the model discussed in [4, 27]. The main objective of DRL is to learn the optimal policy that yields maximum cumulative reward. This problem can be modeled using Markov Decision Process (MDP). In this research, we focus on optimizing the routing actions taken for all jobs to maximize the acceptance ratio and reduce network resource usage. Therefore, such a model fits our expectations.

The MDP is defined by the tuple $\{S, A, T, R\}$ where the state set (S) must represent enough information about the environment. The action set (A) refers to all possible actions that can be applied at any state. In our model, we adopt discrete action space as we train the agent to select the best possible path among certain candidates. The transition function ($T(s, a)$) defines how the environment, the network in our case, evolves from the state ($s \in S$) after taking action ($a \in A$). In our case, the transition function is defined by three main steps. First, we validate if the action is a valid action and the network can support it. Second, we update the network state by subtracting the requested rate from all links that are in the selected path. Third, we randomly generate a new request using a uniform distribution.

In this model, we define the environment as the network topology and the current state of link occupancy. We provide a feature extraction module that simplifies the agent task by extracting useful information from the network state. This process considerably reduces the training required from the agent to learn the relationship between network state and useful features. To that end, we represent the state of the environment using the features of the shortest available paths that can serve application requests. The request information is also encoded in the environment state. Current model representation assumes that the DRL agent stores network state information in local storage, and it can access application intent through Accord protocol. The action space is defined as the probability of selecting a certain path among the shortest ones.

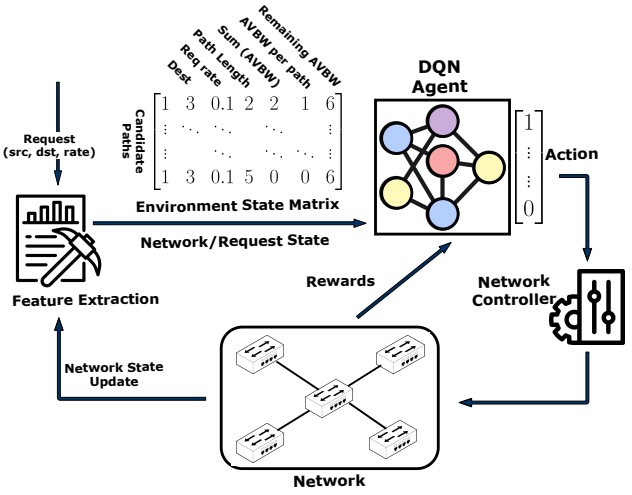


Figure 5: Reinforcement learning model for routing optimization

Fig. 5 depicts the model architecture. First, application requests, provided by Accord, is passed alongside the environment state through a feature extraction module. The feature extraction module represents the network state and the application request as the state of the shortest $K = 10$ paths that can serve this request. The output from this stage is indicated by the environment matrix. Such matrix contains information of the source/destination pair, requested rate, path length, the sum of Available Bandwidth (AVBW) per path, AVBW per path, and the remaining AVBW in the network (features are shown in the same order in Fig. 5). In this experiment, we use FSPF as our baseline for comparison with DQN.

6 EXPERIMENTAL EVALUATION

In this section, we conduct two main experiments to illustrate the benefits of our approaches. In Section 6.1, we experimentally demonstrate the benefits of exchanging job information and application objective in achieving better service. We specifically show how using the processing time information from functions can lead to better scheduling. In addition, with another experiment in section 6.2, we illustrate how information provided from the application yields better routing decision and higher acceptance rate.

6.1 Experiment I: Using application information to enhance scheduling

We use Mininet to emulate the spine-leaf topology depicted in Fig. 6. In this experiment, services A, B, C, and D are deployed at hosts 1, 4, 2, and 3 respectively. One can notice that such configuration creates a bottleneck between leaf switches and the spine switch. Such a bottleneck is intended to throttle rate transmission to depict the enhancement in performance while using our proposed mechanism. Ryu controller is used to apply flow rules that are calculated using our proposed SAFF algorithm. Link capacity is set to 10 Mbps unless stated otherwise. Flow scheduling, route selection, and rate allocation are carried out using a control-plane application running

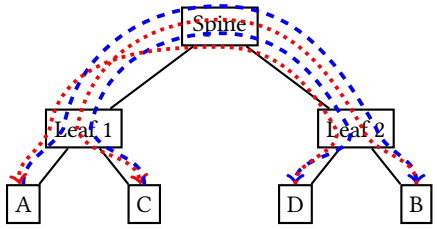


Figure 6: Network Topology

inside the Ryu controller. Moreover, a Remote Procedure Call (RPC) is used to exchange ANI messages between our control application and hosts.

In this experiment, we emulate the ring all-reduce discussed in Section 2.2. The performance metric used here is the FCT as illustrated in Fig. 7. We use fair share approach as the baseline to demonstrate the advantage in using the proposed algorithm. One can notice that while using fair share, four flows share the network in both directions at the same time. Therefore, all flows are throttled to 5 Mbps (dashed lines). However, our approach succeeded in better scheduling flow transmission to utilize maximum link capacity all time and eliminating idle time (solid lines). The results demonstrated in Fig. 7 shows that JCT while using Accord is 27.8% shorter than the fair share approach . The unused gap between flows using fair share approach, as depicted in Fig . 7, is mainly due to the processing time of flows which is efficiently used in Accord. The results achieved in this experiment match our expectation shown in Fig. 2 in Section 2.2.

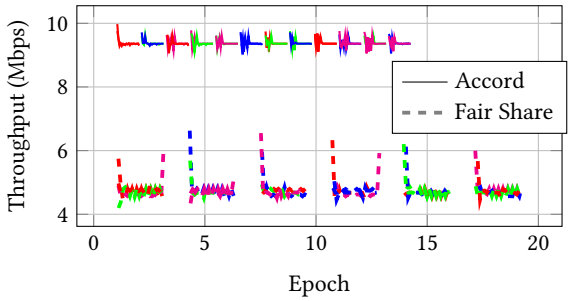


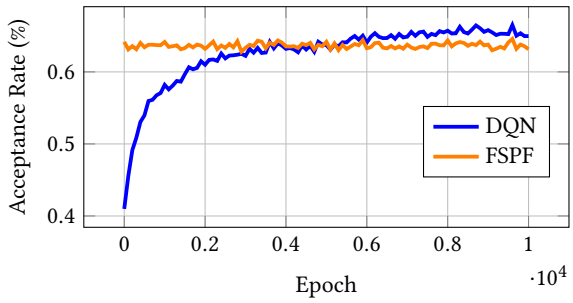
Figure 7: Throughput (Fair share vs Accord)

6.2 Experiment II: Using application requirements to enhance routing decision

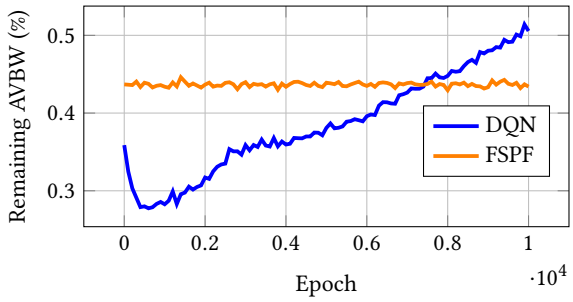
In this experiment, we demonstrate a different method to use application-network interaction to enhance network performance.

We conducted such an experiment by simulation using network topologies provided by Topology Zoo [15]. For this experiment, we normalize link capacity to 1 unit, and we generate flows randomly by generating source/destination pairs from a uniform distribution. Flow rates are also sampled from a uniform distribution between $[0, 0.1]$ unit.

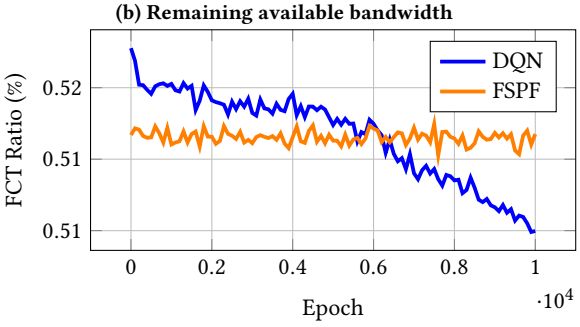
Two key pieces of information that Accord provides are the *processing time* and *application deadlines*. Thus, we use this information to define the minimum and the maximum rate that a flow can use. To this end, a flow rate r can not exceed the processing time required to generate the data of this flow $r \leq \frac{s}{c}$, where s is



(a) Acceptance Rate

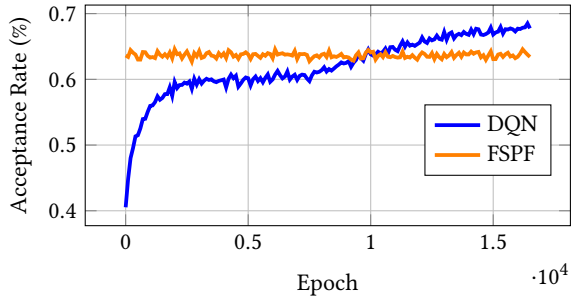


(b) Remaining available bandwidth

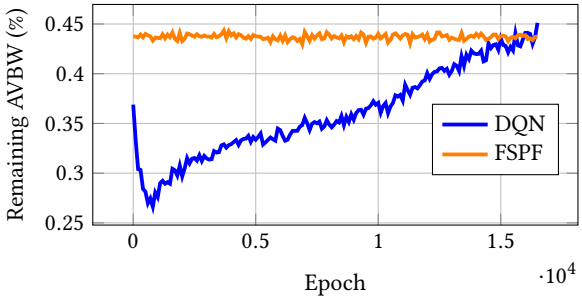


(c) FCT reduction

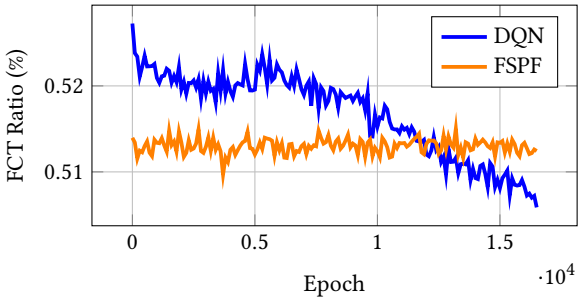
Figure 8: Results for Reward = Acceptance Rate



(a) Acceptance Rate



(b) Remaining available bandwidth



(c) FCT reduction

Figure 9: Results for Reward = $\frac{acceptance\_rate}{\sum_{e\in E} avbw \,/\, \sum_{e\in E} C}$

the flow size and c is the processing time. In addition, to satisfy deadline requirements, the rate must be greater than the minimum transmission rate defined by the deadline d and arrival time t as follows: $r \geq \frac{s}{(d-t)}$.

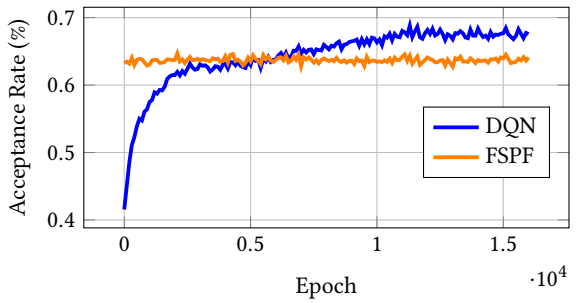
We build a reinforcement learning module using DQN network as depicted in Fig. 5. We compare the performance of DQN to the FSPF. We aim to train the DRL agent to serve the maximum number of flows with minimum resources. Therefore, for performance evaluation we adopt three main metrics; namely, i) the percentage of remaining available bandwidth, ii) the Acceptance rate, and iii) the FCT ratio to the maximum value. To satisfy flows' deadlines and rate requirements stated by their processing time, we introduce the FCT ratio as set it to the maximum value as a third metric to evaluate the performance of the DQN routing module. Such a metric is calculated by dividing the FCT of a flow, if served, with a higher rate depending on the available bandwidth of the selected path compared to its minimum rate that satisfies deadline constraint. Hence, our objective is to maximize acceptance rate, and remaining available bandwidth while maximizing the FCT ratio.

The performance of reinforcement learning depends heavily on the choice of the reward function. Thus, we proposed three reward functions that match most network providers' requirements. First, we defined the reward function to be equal to the percentage of the remaining available bandwidth on all links after admitting each request (EQ. 3). In such a way, we aim to reduce network utilization that is required.

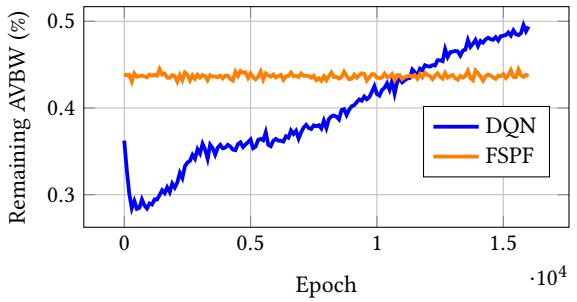
$$r = \frac{\sum_{e\in E} avbw}{\sum_{e\in E} C} \tag{3}$$

where E is the set of all edges in the network.

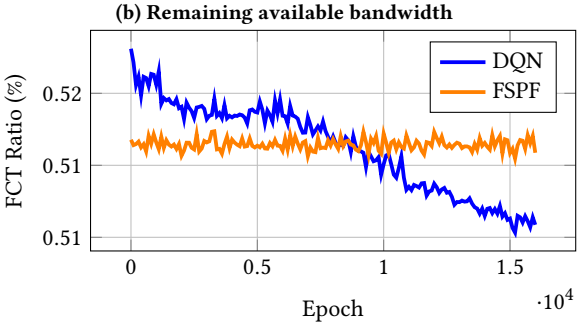
Fig. 8a, 8b and 8c show the percentage of the acceptance rate, the percentage of the remaining available bandwidth and FCT ratio respectively. One can see that our agent is able to outperform FSPF in terms of the percentage of remaining available bandwidth and



(a) Acceptance Rate



(b) Remaining available bandwidth



(c) FCT reduction

Figure 10: Results for Reward $= 1 - \frac{req\_rate * path\_length}{\sum_{e \in E} avbw}$

FCT ratio. However, the enhancement in the acceptance rate is marginal as shown in Fig. 8a.

To enhance the performance regarding the acceptance rate, we design the second reward function to consider both the acceptance rate and the percentage of the remaining available bandwidth. As stated in (4), the reward function is defined as the division of the acceptance rate and the ratio of the remaining available bandwidth to the sum of all link capacities (C).

$$r = \begin{cases} \frac{acceptance\_rate}{\sum_{e \in E} avbw \,/\, \sum_{e \in E} C} & If\ Served \\ 0 & Otherwise \end{cases} \quad (4)$$

Such a reward function is indeed capable of enhancing the acceptance rate and FCT ratio as shown in Fig. 9a and 9c respectively. However, this reward function is not able to train the DRL agent to achieve better resource utilization as shown by the percentage of the remaining available bandwidth in Fig. 9b.

Finally, we adopt the cost of fulfilling flow requests to guide the DRL agent training. With that in mind, we add both the requested rate and path length in the reward calculation as depicted in (5). In addition, such a function considers the ratio of the cost ($req\_rate * path\_length$) to the sum of all link capacity (C).

$$r = \begin{cases} 1 - \frac{req\_rate * path\_length}{\sum_{e \in E} avbw} & If\ Served \\ 0 & Otherwise \end{cases} \quad (5)$$

As depicted in Fig. 10a, 10b and 10c, such a reward function allows the DRL agent to outperform FSPF in all three metrics; namely acceptance rate, remaining available bandwidth and FCT ratio. The figure shows also that DQN reduces the resource utilization by 13% while increases the acceptance rate by 8%.

Therefore, we conclude that enriching routing algorithms with application intent provides better routing decisions and enhances network and application performance. It is worth mentioning that the related state-of-the-art technologies does not consider and discuss the performance of their proposed reward functions from the perspective of all three metrics we discussed above.

7 CONCLUSION

Networks are evolving to become more receptive to application requirements and properties through Network Application Integration with advances in Software Defined Networking and Application Network interfaces. This integration will result in benefits both for the application and the network. However, the current NAI frameworks are not targeted for the datacenter and do not integrate processing and deployment information for optimizations.

In this paper, we presented Accord , an NAI framework for the cloud datacenter that transfers networking and processing data between the application and the network and uses this information to better optimize network resources with two novel scheduling and routing algorithms. Accord takes the first steps of enabling the vision of application-aware networking inside the datacenter. Using distributed deep learning as an example application, we demonstrate that our proposed scheduling algorithm can significantly improve application performance compared to the fair share approach. We also show how our proposed reinforcement learning framework can improve resource utilization and acceptance rate in routing.

As for future work our goal is to further improve network services using processing, network and storage information by providing QoS guarantees in form of SLA's for applications and also extend our framework for network aware application optimizations. We are also testing our solution on other applications and we are working on generalizing our API. Experimental evaluation in a more complex/larger environments with different applications is another goal we are actively pursuing.

REFERENCES

[1] Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. 2018. Sincronia: Near-optimal network design for coflows. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 16–29.

[2] Richard Alimi, R Penno, Y Yang, S Kiesel, S Previdi, W Roome, S Shalunov, and R Woundy. 2014. Application-layer traffic optimization (ALTO) protocol. *RFC 7285* (2014).

[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[4] Xiaoliang Chen, Jiannan Guo, Zuqing Zhu, Roberto Proietti, Alberto Castro, and S. J. B. Yoo. 2018. Deep-RMSA: A Deep-Reinforcement-Learning Routing, Modulation and Spectrum Assignment Agent for Elastic Optical Networks. In *2018 Optical Fiber Communications Conference and Exposition (OFC)*. 1–3.

[5] Mosharaf Chowdhury and Ion Stoica. 2012. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. 31–36.

[6] Mosharaf Chowdhury and Ion Stoica. 2015. Efficient coflow scheduling without prior knowledge. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 393–406.

[7] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. 2014. Efficient coflow scheduling with varys. In *Proceedings of the 2014 ACM conference on SIGCOMM*. 443–454.

[8] Simon Eismann, Johannes Grohmann, Erwin Van Eyk, Nikolas Herbst, and Samuel Kounev. 2020. Predicting the Costs of Serverless Workflows. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. 265–276.

[9] Monia Ghobadi, Soheil Hassas Yeganeh, and Yashar Ganjali. 2012. Rethinking end-to-end congestion control in software-defined networks. In *Proceedings of the 11th ACM Workshop on Hot Topics in networks*. 61–66.

[10] Alim Ul Gias and Giuliano Casale. 2020. COCOA: Cold Start Aware Capacity Planning for Function-as-a-Service Platforms. In *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 1–8.

[11] Jetmir Haxhibeqiri, Amina Seferagic, Ramyashree Venkatesh Bhat, Ingrid Moerman, and Jeroen Hoebeke. 2021. Tighter application-network interfacing to drive innovation in networked systems. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration*. 53–57.

[12] Pieter Hintjens. 2013. *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.".

[13] Sangeetha Abdu Jyothi, Sayed Hadi Hashemi, Roy Campbell, and Brighten Godfrey. 2020. Towards An Application Objective-Aware Network Interface. In *12th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.

[14] Kamal Kc and Kemafor Anyanwu. 2010. Scheduling hadoop jobs to meet deadlines. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. IEEE, 388–392.

[15] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. 2011. The internet topology zoo. *IEEE Journal on Selected Areas in Communications* 29, 9 (2011), 1765–1775.

[16] Danny Lachos, Qiao Xiang, Christian Rothenberg, Sabine Randriamasy, Luis M Contreras, and Börje Ohlman. 2020. Towards deep network & application integration: Possibilities, challenges, and research directions. In *Proceedings of the Workshop on Network Application Integration/CoDesign*. 1–7.

[17] Ziyang Li, Yiming Zhang, Yunxiang Zhao, and Dongsheng Li. 2016. Efficient semantic-aware coflow scheduling for data-parallel jobs. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 154–155.

[18] HE Lin, Peng KUANG, WANG Shicheng, LIU Ying, LI Xing, and PENG Shuping. 2019. Application-aware IPv6 networking. *Telecommunications Science* 36, 8 (2019), 36.

[19] Karl Mason, Martin Duggan, Enda Barrett, Jim Duggan, and Enda Howley. 2018. Predicting host CPU utilization in the cloud using evolutionary neural networks. *Future Generation Computer Systems* 86 (2018), 162–173.

[20] Sambit Kumar Mishra, Deepak Puthal, Bibhudatta Sahoo, Prem Prakash Jayaraman, Song Jun, Albert Y Zomaya, and Rajiv Ranjan. 2018. Energy-efficient VM-placement in cloud data center. *Sustainable computing: informatics and systems* 20 (2018), 48–55.

[21] Takuya Miyasaka, Yuichiro Hei, and Takeshi Kitahara. 2020. NetworkAPI: An In-band Signalling Application-aware Traffic Engineering using SRv6 and IP anycast. In *Proceedings of the Workshop on Network Application Integration/CoDesign*. 8–13.

[22] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel and Distrib. Comput.* 69, 2 (2009), 117–124. https://doi.org/10.1016/j.jpdc.2008.09.002

[23] Danny Alex Lachos Perez and Christian Esteve Rothenberg. 2020. MUDED: Integrating Networks with Applications through Multi-Domain Exposure and Discovery Mechanisms. In *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 132–137.

[24] Danny Alex Lachos Perez, Christian Esteve Rothenberg, Mateus Santos, and Pedro Henrique Gomes. 2020. Ani: Abstracted network inventory for streamlined service placement in distributed clouds. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 319–325.

[25] Zhengwei Qi, Jianguo Yao, Chao Zhang, Miao Yu, Zhizhou Yang, and Haibing Guan. 2014. VGRIS: Virtualized GPU resource isolation and scheduling in cloud gaming. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 2 (2014), 1–25.

[26] Philipp S Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. 2013. Socket intents: Leveraging application awareness for multi-access connectivity. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. 295–300.

[27] Jose Suarez-Varela, Albert Mestres, Junlin Yu, Li Kuang, Haoyu Feng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2019. Feature Engineering for Deep Reinforcement Learning Based Routing. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 1–6. https://doi.org/10.1109/ICC.2019.8761276

[28] Amoghvarsha Suresh and Anshul Gandhi. 2019. Fnsched: An efficient scheduler for serverless functions. In *Proceedings of the 5th International Workshop on Serverless Computing*. 19–24.

[29] Hengky Susanto, Hao Jin, and Kai Chen. 2016. Stream: Decentralized opportunistic inter-coflow scheduling for datacenter networks. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE, 1–10.

[30] Vojislav Đukić, Sangeetha Abdu Jyothi, Bojan Karlaš, Muhsen Owaida, Ce Zhang, and Ankit Singla. 2019. Is advance knowledge of flow sizes a plausible assumption?. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 565–580.

[31] Su Wang, Shuo Wang, Dong Zhou, Yiran Yang, Wenjie Zhang, Tao Huang, Ru Huo, and Yunjie Liu. 2020. Large-scale and rapid flow size estimation for improving flow scheduling. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 1141–1146.

[32] Rahul Yadav, Weizhe Zhang, Huang Chen, and Tao Guo. 2017. Mums: Energy-aware vm selection scheme for cloud data center. In *2017 28th International Workshop on Database and Expert Systems Applications (DEXA)*. IEEE, 132–136.

[33] Jingxuan Zhang, Luis Contreras, Kai Gao, Francisco Cano, Patricia Cano, Anais Escribano, and Y Richard Yang. 2021. Sextant: Enabling automated network-aware application optimization in carrier networks. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 586–593.

[34] Yu Zhang, Ke Zhou, Ping Huang, Hua Wang, Jianying Hu, Yangtao Wang, Yongguang Ji, and Bin Cheng. 2020. A machine learning based write policy for SSD cache in cloud block storage. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1279–1282.

[35] Zhi-Li Zhang, Udhaya Kumar Dayalan, Eman Ramadan, and Timothy J Salo. 2021. Towards a Software-Defined, Fine-Grained QoS Framework for 5G and Beyond Networks. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration*. 7–13.