Towards An Application Objective-Aware Network Interface

Sangeetha Abdu Jyothi* — *UC Irvine, VMware Research*
Sayed Hadi Hashemi* — *UIUC*
Roy Campbell — *UIUC*
Brighten Godfrey — *UIUC, VMware*

Abstract

The network representation used for conveying an application's objective in cloud environments, which we refer to as the Application Network Interface (ANI), has steadily evolved — from packet to flow and flowlet, and more complex abstractions such as coflow. In this paper, we argue that state-of-the-art ANIs still fail to capture important application needs. Using distributed deep learning as a representative application, we show that application performance achievable using current ANIs are up to 25% lower than optimal. We analyze these ANIs to understand the missing pieces and put forward CadentFlow, an ANI with per-flow metrics and an optimization objective, to capture application requirements effectively. We discuss the opportunity for real-world implementation of a more expressive ANI and its implications on the design of network controllers and scheduling algorithms.

1 Introduction

A key goal of data center networks is to enable peak application performance. To achieve this, it is necessary to translate the application's high-level performance needs or application objective to network-level requirements that are actionable for a network controller. This objective is expressed to the controller through a representation that we refer to as the Application Network Interface (ANI). The expressiveness of the ANI can affect application performance significantly.

ANIs and the flexibility they offer have evolved over time. The earliest congestion control and traffic engineering schemes focused on simple proxies for application performance at *packet* level — throughput, per-packet delay, and jitter. Rate Control Protocol [14] made a step towards application-level performance goals with *flow* as the ANI and emphasis on Flow Completion Time (FCT) or the time of arrival of the last packet. Another leap towards an ANI that captures the requirements of cloud applications was the *coflow* [8]. Inspired by cloud applications such as MapReduce, coflow considers a set of parallel flows within an application as a single entity where the FCT of the last flow determines the performance. This enables scheduling schemes to borrow bandwidth from lighter flows in the coflow to speed up the heavier flows, thereby improving coflow completion time.

We observe that even the coflow abstraction is insufficient to support requirements of today's sophisticated applications. Applications such as distributed deep learning and interactive analytics have a complex interplay of communication and computation at the participating nodes. In this scenario, not all flows within a coflow are equivalent from the perspective of the application. Depending on the nature of computation, the application may benefit by finishing some flows sooner than others within a coflow. For example, multiple parameters are exchanged between the parameter servers and a worker in distributed deep learning. These parameters are consumed at different times based on the underlying computational model in frameworks such as TensorFlow. Hence, the iteration time can be improved significantly when the relative priorities of flows (parameter transfers) are made known to the network controller. Different applications may have other dependencies that require metrics such as relative weights or deadlines. More importantly, an application may have an explicit optimization objective different from minimizing the completion time that cannot be conveyed through current ANIs.

In this paper, we argue that it is an opportune moment for narrowing the gap between application objective and its network representation through a more expressive ANI for cloud applications. The stringent performance *needs* of cloud applications coupled with the *opportunity* to extract fine-grained application characteristics using sophisticated learning techniques inspire us to rethink cloud ANI design. First, we analyze several cloud applications to understand communication patterns that are not captured by current ANIs. Second, we quantify the performance benefits achievable with a more expressive ANI in a popular application, distributed deep learning. We show that iteration time in deep learning training can be improved by up to 25% using additional information made available through CadentFlow. In a shared network environment, the improvements increase further up to 46%. Finally, we put forward an application objective-aware ANI and discuss its implication for cloud systems design.

We propose CadentFlow, an ANI whose semantics include component flows of an application with associated per-flow metrics, and an optimization objective for capturing the application objective. Finally, we discuss several research directions including (i) *Extraction of CadentFlow attributes* which requires analysis of applications' models or learning-based inference. (ii) *Redesign of network controllers* with novel scheduling algorithms that can leverage the richer semantics of CadentFlow, and (iii) *In-network implementation* of CadentFlow-aware scheduling using programmable switches.

2 Motivation

In this section, we demonstrate the need for rethinking ANI in the cloud environment by (a) illustrating shortcomings of state-of-the-art ANIs and (b) examining distributed applications that can benefit from an improved ANI.

*equal contribution

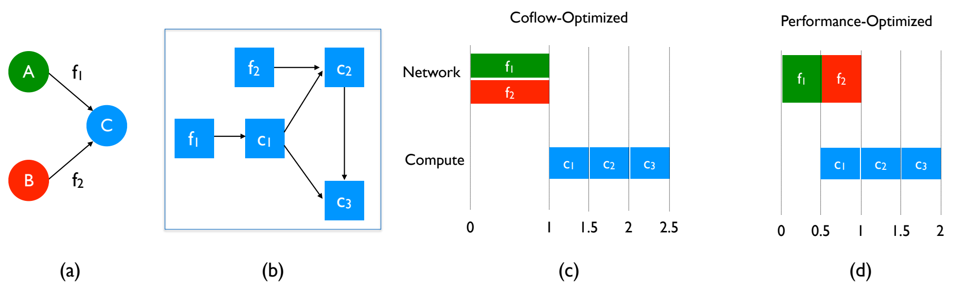


Figure 1: Importance of understanding application objective: (a) Coflow with two component flows. f_1 and f_2 (size 500Mb each) share a 1Gbps bottleneck link. (b) Computation model at C has 3 operations, c_1, c_2, and c_3 with dependencies between flows and computations as shown. Each computation operation takes 0.5s to execute. Completion times with (c) coflow completion time-optimized transfers and (d) application objective-based optimization for transfers.

2.1 The application perspective

In large-scale cloud applications, network transfers involving multiple concurrent flows account for $> 50\%$ of the job completion time [10]. State-of-the-art coflow abstraction [8], which considers the correlated flows as a single entity, is proving insufficient for capturing application requirements due to complex relationship between flows in emerging applications.

For example, consider the simple coflow in Figure 1a with two flows of equal size: f_1 and f_2. The computation at node C has 3 tasks with dependencies shown in Figure 1b. The application completion time with coflow optimization is 2.5s while the optimal time with f_1 prioritized (based on computation in succeeding stage) is 2s. While this simple example can be solved by splitting the single coflow into two (f_1 only, f_2 only) and adding dependency between them similar to multi-stage DAG scheduling in Aalo [9], this approach cannot be generalized to several applications such as DNN training.

Coflow ANI has several limitations. First, coflow completion time (CCT) is used as a proxy for application performance or job completion time. However, as we show in § 4.2.1 with deep learning workloads, a lower CCT may not always correlate with a lower job completion time and may even hurt performance. Thus, an application may have other explicit objectives and finer-grained preferences that cannot be captured by FCT/CCT. Second, coflow dependencies cannot be determined apriori in certain systems (e.g., graph processing systems operating on time-evolving graphs). Third, some applications may prefer weighting across flows instead of strict ordering enforced by a DAG (e.g., stream processing systems). Fourth, in multi-application environments, significant performance benefits can be achieved with deadlines (§ 4.2.2), a benefit not achievable with a DAG used in coflows.

While prior work has considered inter-flow relationships [3, 5, 8–11] and deadlines [19, 37] in the cloud, the set of objectives handled by the network controller has been limited. Inter-coflow scheduling schemes [9–11], even those which are information-agnostic [3, 38], primarily focus on a single objective, minimization of mean CCT, which may not always be a good proxy for application performance. Flexible packet scheduling in the network [25] is also limited to simple schemes such as FIFO and SJF. Moreover, application-level order enforcement independently at each edge [3, 16, 17, 21, 29] is not sufficient since flows originating at multiple edge nodes often need to be coordinated.

2.2 Which applications will benefit?

We analyze different families of advanced data flow systems to understand their requirements.

Distributed Deep Learning: This involves iterative computation with multiple workers and Parameter Servers (PSs). The parameters, updated at the end of each iteration, are acted upon by the worker nodes at different time instances determined by the underlying computational model in frameworks such as TensorFlow [2] and PyTorch [28]. In this case, the iteration time can be improved by prioritizing parameter transfers in the order in which they are consumed [16, 17, 21, 29].

Partition-Aggregation: In online services such as Web page delivery and search query responses, a request is partitioned across multiple workers and the response aggregated by the front-end server/proxy is often sent back to the user before the complete response is available. In this scenario, the application can benefit by prioritizing those flows which are critical to the response. For example, a web proxy waiting for components in a web page can choose to delay fetching a large image. While application-level solutions exist [27, 36], a controller with visibility into multiple such applications can further improve performance across them (similar to § 4.2.2).

Graph Processing Systems: In graph processing systems [6, 7, 24], a graph is partitioned across multiple nodes which process vertices in a sequence and exchange the results of computation after each iteration. This can be accelerated by prioritizing flows corresponding to vertices that need to be processed first. In systems which handle time-evolving graphs [15, 20], these priorities may change over time.

Interactive Analytics: Big data systems analyzing real-time low-latency queries (e.g., Naiad [26]) incorporate explicit application deadlines on each stage of the data flow. This provides opportunity for accelerating/slowing down the network transfers based on deadlines. In stream processing systems [4, 23, 34] with load balancing across parallel components, the end-to-end performance may be improved by weighting the flows according to application preferences.

Thus, in a variety of scenarios, awareness of application objective and dependencies between network flows and computation can help in prioritizing/accelerating those flows which are critical to the application. The coflow abstraction is sufficient when the application has *hard barriers* after each stage [12, 33]. However, several common distributed applications have more complex workflows, inspiring us to rethink the network interface for cloud applications.

3 Application Objective-Aware ANI

Analyzing a wide range of cloud applications, we identify two essential features missing in state-of-art ANIs.

Missing pieces in current ANIs: The first missing piece is a means for explicitly conveying application's objective. In addition to minimization of coflow completion time, applications may have other performance objectives such as maximizing an application-specific utility in terms of per-flow completion time and bandwidth. Hence, we argue that the optimization objective should be an explicit part of the ANI.

The second deficiency in state-of-the-art ANIs is the inability to represent complex dependencies across flows (beyond membership in a set defined by coflows). Dependencies across coflows have been considered with inter-coflow DAGS [9, 35]. However, in practice, dependencies may take several forms (weighted splitting of bandwidth among component flows, deadlines per flow, etc.).

Defining CadentFlow: A *CadentFlow* is a set of correlated flows between a collection of machines with an application-level optimization objective denoted by Γ and a set of tagged flows. Each component flow, f_i, has an associated list of tags, where each tag is a tuple with metric type and metric value. $T_i = (t_{i1}, m_{i1}), .., (t_{ik}, m_{ik})$. We propose weights, deadlines, and priorities as preliminary candidate metrics for denoting the inter-dependencies between flows. As applications and their requirements evolve, more metrics may be added to this set. A CadentFlow can be represented as: $CF = \{\{(f_i, T_i), (f_2, T_2), ..., (f_n, T_n)\}, \Gamma\}$.

CadentFlow Representation of Applications: We present CadentFlow representations of applications discussed in § 2.2. (i) *Distributed deep learning:* There are two possible representations for DNN training. (a) Frameworks such as TensorFlow provide application-level DAG with inter-dependencies between communication and computation. This information can be used to determine priorities of flows, and the objective will be minimizing completion time of last flow, subject to scheduling based on priorities. (b) Since, DNN training is an iterative process, we can estimate the time required by computation operations in a given system. Combining this information with the DAG, deadlines can be estimated for individual flows. This provides additional scheduling flexibility. The objective is minimizing maximum delay subject to deadlines for all flows. This representation improves flexibility by allowing delayed scheduling of flows with flexible deadlines (as shown in § 4(b)). (ii) *Partition-Aggregation:* In Web page delivery, priorities can be set based on rendering preferences. The objective is minimizing completion time subject to prioritized transfers. (iii) *Graph processing systems:* Weights used as metrics for load-balancing, objective is minimize completion time subject to weighted transfers. (iv) *Naiad (interactive analytics)*: Deadline is the metric and minimization of sum of delays with respect to deadline is the objective.

4 Experiments

We quantify benefits achievable with CadentFlow using distributed deep learning as a representative application.

4.1 Methodology

Workload: We test deep learning workloads using TensorFlow under two scenarios: training and inference. In training, during each iteration workers (each with an identical copy of the model) send parameter updates to the Parameter Servers (PS). PS aggregates the changes and returns the updated parameters to all workers. In inference, the inference agents read the parameters from the servers and run the inference. This captures the online inference scenario where agents (separate from the workers and Parameter Servers) read the latest version of parameters during reinforcement learning and serve inference queries. We test 11 popular Neural network models (including [18, 22, 30–32]) using [1] on TensorFlow 1.8 using the standard batch size for each model.

Estimating metrics: The TensorFlow computational model is a DAG with dependencies between computation and communication operations/parameter transfers. Given a hardware configuration, computation in DNN training and inference workloads is highly predictable. In every iteration, since batch size of input remains constant, time taken by each computation operation in the DAG can be accurately estimated. Hence, total computation before a communication operation can be estimated accurately with a few initial runs. Also, the model at all workers is identical.

DNN Training has two phases: (a) forward pass (FP) where latest parameters are read from PS and loss function is computed, and (b) backpropagation (BP) where parameters are updated and sent to PS. Thus, we have two CadentFlows (FP and BP) in one iteration of training. In the FP CadentFlow (PS to workers), all flows have the same estimated start time at 0, i.e., all parameters are ready for transfer at PS at the beginning of a new iteration. However, not all parameters are consumed at the same time (e.g., parameters of layer 1 are used before layer 2 during computation). Hence, the deadline of a parameter is estimated as the total computation time before the read operation of that parameter. Thus, parameters of initial layers have earlier deadlines. In the BP CadentFlow (workers to PS), the estimated start time is computed as the total computation time before the parameter is ready to be sent to the PS. Parameters are updated in the reverse order of layers in DNN. Thus, parameters of earlier layers in the DNN have later start times. All flows share the same deadline determined by the end of computation. The inference workload only has forward pass and hence a single CadentFlow with deadline estimation similar to training forward pass.

Note that the deadline estimation depends only on the DAG and not on the available bandwidth. Thus, the deadlines represent the preferences of the application and the flexibility available between flows from the computation perspective.

We evaluate the workloads under two scenarios: CadentFlow with flow metrics as (a) priorities and (b) deadlines. The TensorFlow computational DAG is available through an API. We estimate priorities based on position of parameter reads and updates in the DAG. We estimate deadlines by tracing TensorFlow runs. We collect the runtime information on Standard NC6 virtual machines (6 cores, 56 GB RAM, 1 X Nvidia K80 GPU with 12GB RAM) in Azure. We run the same model 10 times on a single machine to measure time taken by an operation. We estimate the deadlines using DAG structure and mean time taken by each computation operation.

Control Schemes: In our experiments, we test 3 schemes: (i) TCP simulated with max-min fair sharing across flows sharing a link, (ii) Coflow scheduling using Minimum Allocation for Desired Duration (MADD) used in Varys [11] where lighter flows are allocated a lower bandwidth in such a manner that all flows in a coflow finish at the same time, (iii) CadentFlow scheduling where flows are transferred based on priorities/deadline estimated from the TensorFlow model. For a fair comparison, we assume that the baseline TCP connection transfers the parameters in the best possible order from a given node. This baseline is similar to node-level prioritization (similar to [3]) without inter-node considerations.

Using runtime estimates in the previous step and a network controller, we simulate the network schemes under multiple configurations (1 to 16 PS/workers and 1/10 Gbps NIC). For Coflow and CadentFlow scheduling, the centralized controller can make globally optimal decisions and enforce them at the edge. We evaluate performance benefits achievable with a single active application as well as multiple applications.

Application Simulation: We measure two metrics on the distributed deep learning application: the iteration time and the CCT flexibility ratio. The ***iteration time*** is the time taken by one complete iteration composed of computation time (empirically estimated) and communication time (evaluated using the three control schemes in our network simulator). In the training workload, this includes the computation in forward pass, backpropagation phase, and two CadentFlows (FP and BP). In the inference workload this is composed of one CadentFlow for fetching parameters from the servers and the inference computation. We assume that all the workers have identical computation time, i. e., no stragglers.

We introduce ***CCT flexibility ratio*** to measure the flexibility available in flow scheduling with *deadlines* as flow metrics. CadentFlow deadlines may be higher than the best possible completion time. This allows the network controller to *delay* some flows without affecting application performance and use the saved bandwidth for other applications with tighter deadlines. We measure this flexibility using CCT flexibility ratio, defined as the ratio of the time available until the last deadline in a CadentFlow (max feasible CCT) to the minimum CCT achievable for that CadentFlow in the given network. For example, at $t = 10s$, a new CadentFlow arrives with last deadline as $40s$ with the best possible transfer time of $15s$ (ending at $10+15 = 25s$), CCT flexibility ratio is $\frac{40-10}{15} = 2$.

4.2 Results

We present results on training and inference workloads across the three control schemes in an environment with 10Gbps network. The conclusions were also verified with 1Gbps.

4.2.1 Single Application

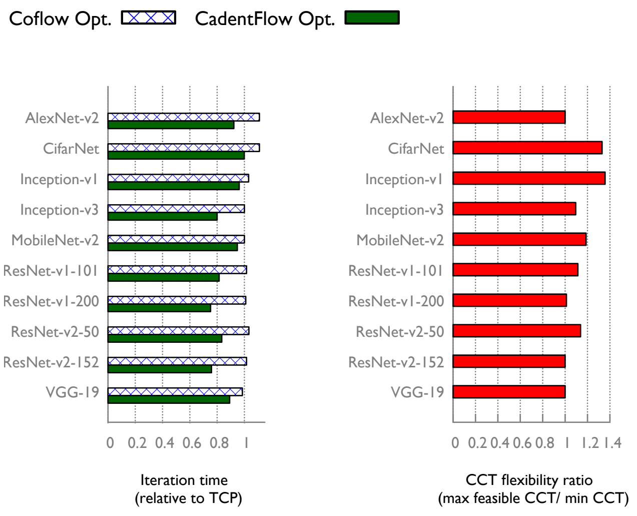
To understand the differences between control schemes, we first conduct experiments with a single active application in the network. The iteration time of coflow optimized scheme (MADD) and the CadentFlow optimized scheme are compared with the TCP baseline in Figures 2(a,c) and 3(a). Lower iteration time implies better performance. The iteration time results are the same with both priority-based and deadline-based CadentFlow optimization, with a maximum reduction in iteration time of 25% (ResNet-v1-200 with 8 W, 8 PS).

We note that Coflow scheduling may result in worse performance than TCP in some cases. For example, the iteration time on CifarNet is 11% higher with coflow optimization compared to TCP in a system with 8 workers and 8 PS. This is due to the coflow scheduling mechanism that delays smaller transfers to accelerate large transfers. This hurts transfers of small parameters in earlier layers of DNN. Note that the CCT is same for both Coflow and CadentFlow optimizations in this scenario. However, application iteration time is lower for CadentFlow optimization. Also, CadentFlow optimization performs at least as good as TCP in all tested scenarios.

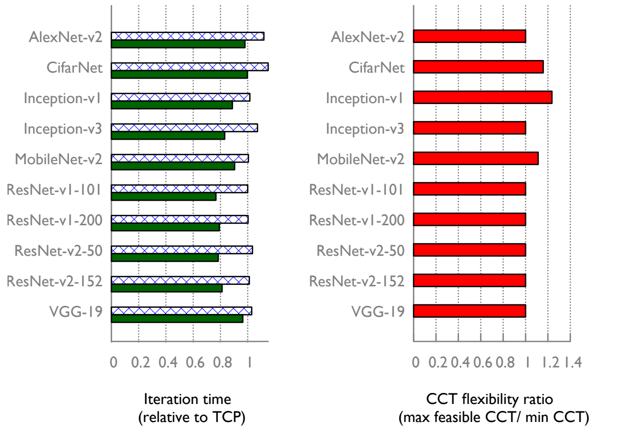
With deadline-based Cadentflow, we can also obtain CCT flexibility (ability to delay flows until deadline without affecting iteration time) as seen in Figures 2 (b,d) and 3 (b). In some models, we obtain both improvement in iteration time and high CCT flexibility (e.g., Inception-v3 with 20% lower iteration time and 9.5% of added flexibility in coflow completion times). The flexibility depends on the ratio of time taken by computation and communication in various models. There is higher flexibility in computation-heavy models, giving opportunity to delay transfers.

In the inference workload (Figure 3), coflow optimization leads to significant performance degradation (up to 68.5%

increase in iteration time compared to TCP in ResNet-v1-101). However, CadentFlow performance is comparable to that of TCP with significant flexibility in flow scheduling across many networks.



(a) Iteration time: 8 W, 8 PS (b) CCT flexibility: 8 W, 8 PS



(c) Iteration time: 16 W, 16 PS (d) CCT flexibility: 16 W, 16PS

Figure 2: (a) Coflow and CadentFlow optimizations plotted relative to TCP. Lower iteration time is better. (b) CCT flexibility shows the window of flexible time available for scheduling with respect to minimum Coflow Completion Time for deadline-based CadentFlow.

4.2.2 Multiple Applications

With deadline-based CadentFlow, we demonstrate benefits of CCT flexibility by running two applications in a shared network (Figure 4(b)). We run two jobs corresponding to the same DNN model across three racks, where one rack is dedicated to each application (12 servers) and one rack is shared (4 nodes each in a rack per application). The fair-share for each application in the shared link S_1-S_4 is 30 Gbps while the bandwidth requirement is 40GBps. CCT flexibility allows us to schedule flows in a manner that maximizes 40GBps allocation to each application instance. In Figure 4(b), we compare the three control schemes in this multi-app environment normalized with respect to the iteration time with CadentFlow scheduling when only a single app is active. We observe that CadentFlow scheduling has the best performance across all models due to the flexibility provided by deadlines. Coflow scheduling can result in up to 46.3% higher iteration time (as in ResNet-v2-50) since deadline information is not available for flexible allocation. Since deadline information is necessary for accruing these benefits, even a DAG of coflows cannot provide this performance improvement.

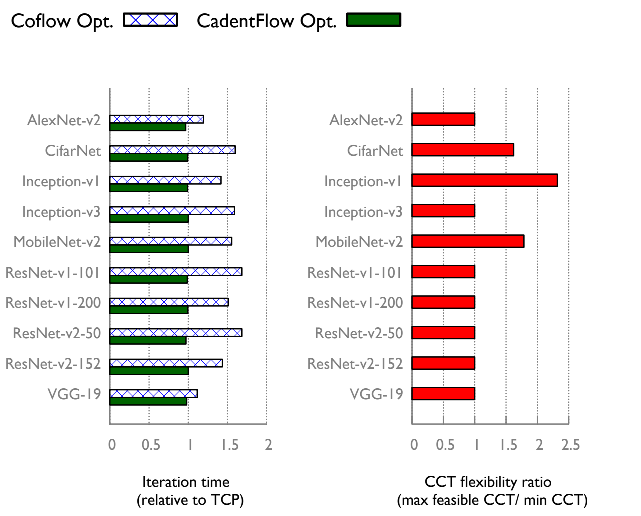
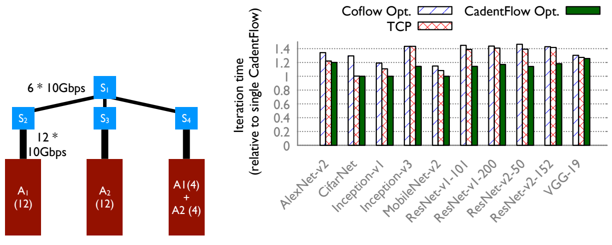


Figure 3: Inference workload (16 workers, 16 Parameter Servers)



(a) **Network setting** (b) **Performance improvement**

Figure 4: (a) 12 servers of rack 1 connected to switch S_2 runs application instance A_1. 12 servers of rack 2 connected to switch S_3 runs instance A_2. Out of 12 servers in rack 3, 4 servers belong to A_1 and 4 to A_2. (b) Performance improvement achievable with efficient overlap of multiple jobs in a shared network environment with deadline-based CadentFlow.

In summary, we make the following observations:

- Flow-level priorities/deadlines in CadentFlow allow optimizing performance by up to 25% (a significant improvement for long-running DNN workloads).
- Deadline-based CadentFlow can improve network-wide performance (up to 46%) by providing increased flexibility.
- Coflow optimization can hurt application performance when there are complex dependencies. In such cases, falling back on TCP should be preferred.

5 Discussion

Preliminary experiments show that current ANIs are insufficient for representing requirements of emerging applications. However, a more expressive ANI like CadentFlow raises several challenges.

Extracting the application objective: While application DAG is easily available for deep learning workloads, determining flow metrics and application objectives can be cumbersome for most applications. Do we need application developers to express their network requirements more explicitly? Alternatively, are machine learning-based frameworks that learn and adapt the deadlines/priorities/weights based on application performance more suited in this context? System-level cues such as reads and writes to disk/memory may help in deriving deadlines/priorities based on application behavior. Recent advances in flow prediction [13] are encouraging for the possibility of learning finer grained network characteristics of applications. As application requirements become more stringent and resource usage more heavy, how can we express the relationship between computation and communication across multiple nodes more effectively? Is CadentFlow a complete ANI? If not, what are the missing features that we need to add to a comprehensive ANI? CadentFlow considers flows as the basic unit. Are there applications that need metrics at packet level or flowlet level instead of flow level?

Application Objective-Awareness in Network Scheduling: An ANI which captures application objectives more effectively opens avenue for design of resource allocation schemes that can handle diverse objectives and constraints. Some of the challenges include defining fairness across applications with varied objectives (beyond CCT) and handling coexistence of CadentFlows with regular TCP connections. A more difficult scenario involves coexistence of applications with conflicting objectives. How can we ensure fairness when multiple application objectives are incompatible? Scheduling under limited information is also an interesting problem to tackle. While our experiments focused on a single application with explicit knowledge of flow sizes and priorities, in practice, one or more of these factors may be unknown. How can we extend scheduling schemes to handle probabilistic estimates of flow metrics while also accounting for accuracy of metrics?

Application-Aware Network Control Implementation: An enhanced API that supports multiple objectives and novel scheduling schemes can be implemented in a wide range of settings: in-host implementation at the edge, using traditional network hardware with limited number of priority queues, using programmable network switches with higher flexibility, or a combination of host and network devices. Implementation challenges include extracting/learning application network requirements in real-time across distributed hosts, making and propagating decisions across network elements in a timely fashion, determining overheads involved in delaying flows (such as buffer overheads, location for buffering, etc.)

References

[1] TensorFlow-Slim image classification model library . https://github.com/tensorflow/models/tree/master/research/slim, 2019.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*, volume 16, pages 265–283, 2016.

[3] Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. Sincronia: Near-optimal network design for coflows. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 16–29, New York, NY, USA, 2018. ACM.

[4] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[5] L. Chen, W. Cui, B. Li, and B. Li. Optimizing coflow completion times with utility max-min fairness. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[6] Rong Chen, Jiaxin Shi, Yanzhe Chen, and Haibo Chen. Powerlyra: Differentiated graph computation and partitioning on skewed graphs. In *Proceedings of the Tenth European Conference on Computer Systems*, page 1. ACM, 2015.

[7] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *Proceedings of the VLDB Endowment*, 8(12):1804–1815, 2015.

[8] Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 31–36, New York, NY, USA, 2012. ACM.

[9] Mosharaf Chowdhury and Ion Stoica. Efficient coflow scheduling without prior knowledge. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 393–406, New York, NY, USA, 2015. ACM.

[10] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I. Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. In *Proceedings*

of the ACM SIGCOMM 2011 Conference, SIGCOMM '11, pages 98–109, New York, NY, USA, 2011. ACM.

[11] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 443–454, New York, NY, USA, 2014. ACM.

[12] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[13] Vojislav Dukic, Sangeetha Abdu Jyothi, Bojan Karlas, Muhsen Owaida, Ce Zhang, and Ankit Singla. Is advance knowledge of flow sizes a plausible assumption? In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, Boston, MA, 2019. USENIX Association.

[14] Nandita Dukkipati and Nick McKeown. Why flow-completion time is the right metric for congestion control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, January 2006.

[15] Wentao Han, Youshan Miao, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Wenguang Chen, and Enhong Chen. Chronos: A graph engine for temporal graph analysis. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, pages 1:1–1:14, New York, NY, USA, 2014. ACM.

[16] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy H Campbell. Priority-based Parameter Propagation for Distributed DNN Training. In *The Conference of Systems and Machine Learning*, 2019.

[17] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, Brighten Godfrey, and Roy Campbell. Caramel: Accelerating decentralized distributed deep learning with computation scheduling, 2020.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[19] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. Finishing flows quickly with preemptive scheduling. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 127–138, New York, NY, USA, 2012. ACM.

[20] Anand Padmanabha Iyer, Li Erran Li, Tathagata Das, and Ion Stoica. Time-evolving graph processing at scale. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, GRADES '16, pages 5:1–5:6, New York, NY, USA, 2016. ACM.

[21] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, and Gennady Pekhimenko. TicTac: Accelerating Distributed Deep Learning with Communication Scheduling. In *The Conference of Systems and Machine Learning*, 2019.

[22] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[23] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M Patel, Karthik Ramasamy, and Siddarth Taneja. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 239–250. ACM, 2015.

[24] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[25] Radhika Mittal, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Universal packet scheduling. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, page 501–521, USA, 2016. USENIX Association.

[26] Derek G Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: a timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 439–455. ACM, 2013.

[27] Ravi Netravali, Anirudh Sivaraman, James Mickens, and Hari Balakrishnan. Watchtower: Fast, secure mobile page loads using remote dependency resolution. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, pages 430–443, New York, NY, USA, 2019. ACM.

[28] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration, 2017.

[29] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo.

A generic communication scheduler for distributed dnn training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 16–29, New York, NY, USA, 2019. Association for Computing Machinery.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[31] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[32] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[33] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2(2):1626–1629, August 2009.

[34] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM, 2014.

[35] J. Wang, H. Zhou, Y. Hu, C. d. Laat, and Z. Zhao. Deadline-aware coflow scheduling in a dag. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 341–346, Dec 2017.

[36] Xiao Sophia Wang, Arvind Krishnamurthy, and David Wetherall. Speeding up web page loads with shandian. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 109–122, Santa Clara, CA, 2016. USENIX Association.

[37] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 50–61, New York, NY, USA, 2011. ACM.

[38] Hong Zhang, Li Chen, Bairen Yi, Kai Chen, Mosharaf Chowdhury, and Yanhui Geng. Coda: Toward automatically identifying and scheduling coflows in the dark. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 160–173, New York, NY, USA, 2016. ACM.