ML-based Incast Performance Optimization in Software-Defined Data Centers

Kokouvi Benoit Nougnanke, Yann Labit, Marc Bruyere

ML-based Incast Performance Optimization in Software-Defined Data Centers

Kokouvi Benoit Nougnanke*, Yann Labit*, Marc Bruyere†
* *LAAS-CNRS, Université de Toulouse, CNRS, UPS, F-31400 Toulouse, France*
† *IIJ Innovation Institute, University of Tokyo, Tokyo, Japan*
nougnanke@laas.fr, ylabit@laas.fr, marc@iij.ad.jp

Abstract—**Traffic optimization is fundamental to achieve both great application performance and resource efficiency in data centers with heterogeneous workloads, including incast. However, general performance models, providing insights on how various factors affect a certain performance metric used in the network optimization process, are missing. For the special case of incast, the existing models are analytical models, either tightly coupled with a particular protocol version or specific to certain empirical data. This paper proposes an SDN-enabled machine-learning-based optimization framework for incast performance optimization in data center networks that leverages learning-based performance modeling. Evaluations based on intensive NS-3 simulations show that we can achieve accurate performance predictions that enable finding the efficient switch buffer space to achieve optimal incast completion time in different configurations. We expect this framework to be a building block for autonomous data center network management.**

Index Terms—**Data centers, SDN, Machine Learning, Traffic Optimization, QoS.**

I. Introduction

Datacenter traffic is continuously increasing with the number of web applications on the internet and business applications from multi-cloud environments. Even if the data center workload could be classified into two main groups, elephant and mice flows, it includes diverse types of traffics with various QoS requirements. Besides the classification above, data center workloads often require sending requests to large numbers of servers and handling their near-simultaneous responses. This many-to-one communication and its associated traffic pattern are called incast traffic [1]. This many-to-one pattern in data centers is used for applications such as distributed storage (e.g., BigTable, HDFS, and GFS), web-search with partition/aggregation design pattern and cluster computing platforms (MapReduce, Spark, etc.) [2].

The dynamic workload with heterogeneous QoS requirements complexify data center management tasks. These tasks include congestion control, buffer management, load balancing, and network optimization in general. Datacenter network optimization aiming to achieve both great application performance and resource efficiency is necessary. This optimization is more important for its critical traffics, such as incast.

Such an optimization process involves continuous network monitoring (performance and resource utilization) and performance modeling [3]. On the first hand, continuous monitoring is needed to have deep visibility on the network. It may ease autonomous optimization through continuous adjustment. This can be done with SDN, with its telemetry capabilities and its management flexibility. On the other hand, modeling is fundamental for network optimization, as reported in [4]: ”we can only optimize what we can model.” Indeed, to optimize incast performance, a model providing insights on how various factors affect it is required.

The classical approach for network modeling is the design of handcrafted and specialized performance analytical models. For incast traffic, mostly carried with TCP, performance modeling is very challenging [5]. Indeed, TCP's stack is a complex system that involves many heuristics to handle network conditions and application behaviors [6]. Subtle changes in its parameters may lead to entirely different performance. The existing incast performance analytical models [5], [7], [8] are either tightly coupled with a particular protocol version or specific to certain empirical data. Relying only on analytical performance modeling for incast performance optimization is then not a practical solution.

In this context, we propose a machine-learning-based optimization framework for incast performance optimization in datacenter networks. This framework leverages SDN and learning-based incast performance modeling [9]. The model is learned from historical data. There is no need for domain-specific assumptions, and the ML model generalizes easily. The machine-learning-based model can capture complex relationships from diverse system parameters (number of incast senders, link bandwidth, switch buffer space, congestion algorithm, etc.) to predict incast performance (its completion time) accurately. Our optimization framework can then provide efficient incast performance with resource efficiency. It can also be leveraged for proactive management for future planning in what-if-scenario modeling. Last but not least, the framework's data-driven approach makes it self-driving so that it can be easily integrated with autonomous data center management schemes.

The main contributions of this paper are summarized below:

- We propose a machine-learning-based optimization framework for incast performance optimization. This framework can achieve great incast performance with efficient resource utilization (Section III).
- We design a random forest incast performance model using a dataset generated from intensive NS-3 simulations. These simulations use mixed incast-elephant traffics scenarios (Section IV).

- Using the ML model, we propose an optimization algorithm to minimize incast completion time with the optimal switch buffer space (Section V).
- Finally, we present the performance evaluation results of the random forest prediction model and the optimization algorithm (Section VI).

II. Background and Problem Formulation

A. Mixed Elephant and Incast Traffic Scenario and Notations

Datacenter workloads are composed essentially of long-lived flows or elephant flows (e.g., backup, replication, data mining, etc.) and short flows or mice flows (e.g., delivering search results). This makes the data center a very dynamic and complex system. Moreover, data center workloads often require sending requests to large numbers of servers and then handling their near-simultaneous responses. This many-to-one communication pattern is called incast traffic. Depending on the size of the servers' responses, we can distinguish between long-lived incast and short-lived incast. But it's worth mentioning that incast manifests as short flow [10].

The co-existence of incast traffic with elephant flows brings other troubles and challenges. Incast flows may get queued up behind packets from large flows in the presence of congestion if ever available buffer space remains, experiencing performance degradation (long queuing delay or tail drops) [11]. Switches must accommodate this mixed traffic with full throughput and low latency while efficiently handling incast.

Fig. 1 shows a dumbbell topology of mixed elephant and incast traffic. All senders and clients are connected to the switches with a 1Gbps link. The bottleneck link between the two switches S1 and S2, has the bandwidth C. Incast senders are connected in a star shape to the switch S1. In this figure, N incast servers send each other the quantity SRU (Server Request Unit) simultaneously to the incast sink node linked to S2. In this scenario, the elephant traffic corresponds to a bulk transfer from the elephant source to the elephant receiver (e.g., background continuous file transfer or server migration). We consider the setting parameters as in TABLE I. These notations hold for the rest of the paper.

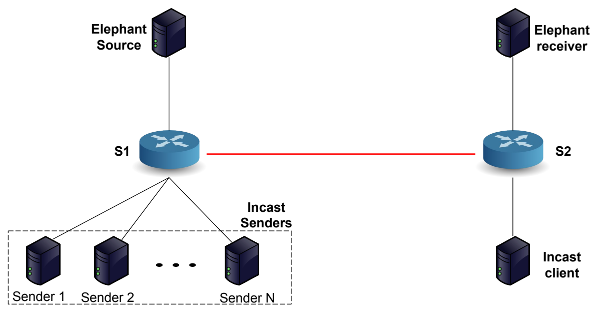

Fig. 1. Basic topology of mixed elephant-incast scenario

TABLE I
Parameters and Notations

Parameters	Description
N	Number of incast senders
SRU	Server Request Unit size, per sender; SRU = 256 KB
B	Switch buffer size in packets; e.g. 64 pkts or 96 KB
C	Bottleneck link capacity; e.g. C = 1 Gbps
$baseRTT$	RTT without queuing delay; e.g. $baseRTT = 200\mu s$
RTO_{min}	Minimal TCP Retransmission timeout; e.g. 10 ms
S	TCP segment size, S=1446 bytes. Packet size = 1.5 KB
τ	Overall Incast Completion Time

B. Problem Formulation

In the scenario described above, we suppose the background elephant flow not too critical. The challenge is to guaranty optimal performance for incast the critical traffic in such conditions. Our objective is to minimize incast flow completion time by finding adequate parameter settings. This goal is formalized with the optimization problem as follows.

$$\begin{aligned} \underset{P}{\text{Minimize}} \quad & \tau = f(P) \\ \text{subject to} \quad & P \in \Omega_P \end{aligned}$$

With P the set of parameters P_i (N, C, B, SRU, CC, qdisc etc.) with their respective values spaces (e.g. $SRU_{min} \leq SRU \leq SRU_{max}$; $B < B_{max}$). Ω_P is the resulting overall variables domains, not to say the optimization problem constraints.

Formalized this way, the problem seems obvious to solve. Unfortunately, that is not the case. Indeed f expressing τ based on the parameters P_i is an unknown function. In other words, there is no analytical model providing incast performance based on system parameters. This makes our optimization problem challenging, and it falls in the category of the so-called black-box optimization.

Faced with this challenge, we propose an SDN-enabled Learning-based Incast Performance Optimization framework. This framework leverages Software-Defined Networking (SDN), Machine-Learning and black-box optimization. This framework is presented in the next section.

III. SDN-enabled Machine Learning Traffic Optimization Framework

With this framework (see Fig. 2), it is possible to optimize incast traffic in a mixed incast-elephant scenario. Indeed, by separating the network's control plane from its data plane, SDN introduces flexibility in network management, provides a global view of the network, and facilitates telemetry. Moreover, it eases the use of machine learning techniques in the management plane [12]. Using machine learning, we could provide good predictions of incast performance τ for different parameter combinations. With these predictions, we can construct $\hat{f}$, an estimate of f that will be used during the optimization process.

From Fig. 2, the workflow of the framework is as follows. Firstly, the prediction model is trained offline on the historical data. The samples of the historical dataset represent a combination of feature values and the associated target

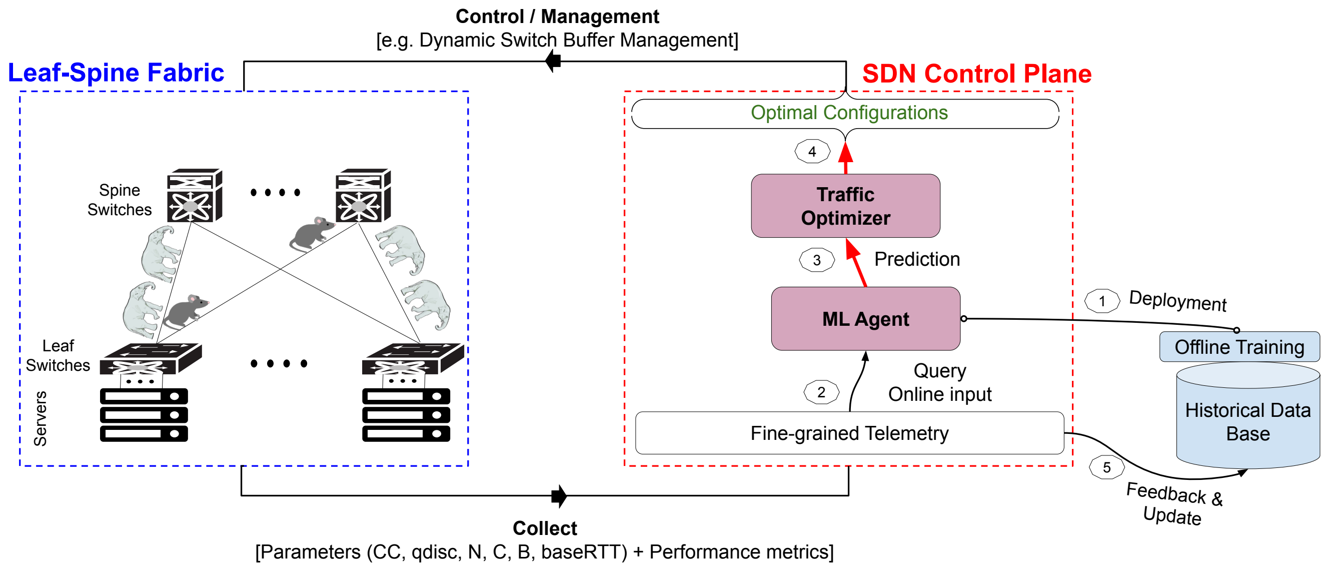

Fig. 2. SDN-enabled Learning-based Incast Performance Inference Framework

value since we are in a supervised learning configuration. The features include the congestion control algorithm used (CC), the queuing discipline at the switch level (qdisc), the number of incast senders (N), the bottleneck bandwidth (C), the base round-trip-time (baseRTT), the server request unit (SRU) and the target attribute is the incast completion time (τ).

The trained model is then deployed (1) as the ML Agent. The model is deployed to be used for incast performance inference. This step provides accurate performance models for the optimization process. The online input (2), composed of (CC, qdisc, C, SRU, N, RTT, baseRTT), is got when an incast traffic is initiated by the client leveraging SDN fine-grained telemetry, In-band Network Telemetry (INT), and P4. Taking this input, a prediction of the incast traffic's performance is done (3). This prediction will then be leveraged by the Traffic Optimizer (4), which determines optimal runtime configurations to achieve efficient performance for the incast traffic.

Finally, when the incast traffic completes, its real observed performance metric is also collected, and the historical dataset could be updated (5). Having the database up-to-date is essential and will allow taking into account new dynamics from the data center. When the database significantly changed, the model needs to be re-constructed and re-deployed. Step (5), indirectly, gives also feedback for the optimizer.

IV. ML Incast Performance Prediction Model

We follow the typical workflow of Machine learning in networking [13]. We begin with the problem formulation. For the incast performance predictions, the target metric (completion time) being a continuous variable, its prediction is a regression problem. We are in the supervised learning case.

For this ML-based prediction mechanism to be relevant for our black-box optimization, it needs to be simple and easily generalizable [14]. Moreover, it is preferable if the model does not require a too high training set size to provide good predictions that capture well τ's dynamics.

A. Dataset and Analysis

We conducted intensive NS-3 simulations using the scenario topology in Fig. 1 and varying the parameters from TABLE I. For every simulation, we compute the corresponding completion time. We finally generate a dataset composed of 83200 observations, six parameters, and one target variable, the incast completion time. The variables include two categorical variables: the congestion control algorithm CC used (NewReno or Cubic) and the associated queuing discipline qdisc (FIFO or FQ_CoDel). The numerical variables are the bottleneck link bandwidth C, the base round trip time $baseRTT$, the switch buffer size B, and the number of simultaneous incast senders N. The server request unit SRU and RTO_{min} were fixed to 256000 bytes, and 10ms respectively, and were not part of the dataset features.

For the dataset preparation for the model training, we consider two possibilities: a single model taking six features (two categorical and four numerical variables) and the case where we elaborate individual models for the 4 categories (NewReno_FIFO, NewReno_FQ, Cubic_FIFO, Cubic_FQ). We will consider the individual models' case in this paper. Detailed information about the single model construction can be found in this previous work [9].

Each category is represented by a dataset of 20800 observations with only the numerical variables. In the data pre-processing step, the dataset is scaled by standardizing these numerical features. It consists of centering a feature's observations to the mean and scaled it to unit variance.

B. Model Training

The machine learning algorithm used for the predictions is Random Forest. Scikit-learn 0.23.2 [15] is used for the implementation.

Random Forest falls under machine learning averaging methods that combine the predictions of several base estimators here decision trees. The combined estimator is usually better since its variance is reduced. Decision trees are a non-parametric machine learning algorithm that predicts by learning simple decision rules inferred from the data features.

Hyper-parameter tuning analysis is conducted, and it reveals that default values perform pretty well for the predictions (e.g., the number of estimators (trees) used is n_estimators = 100).

From a minimalism perspective, for our ML model, we will not use all the samples from our dataset. A subset with n_samples (e.g., 5000, 10000, 15000, etc.) will be used after shuffling the entire dataset. Form this minimalist set, 70% will be used to train the model, while the 30% remaining will be used to validate the trained model.

V. Incast Performance Optimization

A. Preamble

The goal here is to find the optimal parameter setting for an incast request. An incast request involves the number of incast senders N, the total quantity of data to retrieve from the senders (related to the SRU), and the other related parameters like CC, qdisc, C, B, baseRTT, etc.

Depending on the context, not all of these parameters would be decision variables of the optimization problem. Non-decision variables (e.g., the CC) will be taken as inputs of the optimization model. The manageable parameters would generally be related to the switches, such as its buffer space or its queuing discipline. Adaptive buffer management [16] can be achieved easily with data-plane programmability powered by P4.

This paper will focus on efficiently choosing the switch buffer size B to maximize the incast performance by minimizing its completion time. This assumption turns our optimization problem into a switch buffer sizing one.

B. Switch Buffer Sizing

Before solving the optimization problem, it's worth recalling switch buffer sizing. Buffer is in the heart of the vast majority of performance issues [17]. Incast traffic is generally impacted by timeouts that are caused by packet losses from an overfilled buffer. Sizing it conveniently is of great importance.

In general, without making a focus on incast, buffer sizing was intensively studied in the literature. Indeed buffers allow internet routers to hold packets during congestion time. The rule-of-thumb states a buffer size equals the output bandwidth C times the round-trip-time RTT $B = C * RTT$ (the BDP, bandwidth-delay product) to keep high utilization at the bottleneck link [18]. Since link bandwidth was increasing and the number of flows carried by internet routers was growing exponentially, the BDP rule was very challenging for routers manufacturing. The BDP rule was then questioned by [19] that

Algorithm 1: ML-based Performance Optimization

```
Input: Parameters $P'$, maximum buffer space $B_{max}$
Output: $B*$, the optimal buffer space
1 model = Random Forest trained model
2 Generate a search space $\Omega_B$
3 for $B \in \Omega_B$ do
4     fct = model.predict[P', B]
5     FCTs[B] = fct
6 end
7 $B* = \underset{B}{\text{argmin}}$ FCTs
```

proposes a smaller buffer requirement $B = (C * RTT)/\sqrt{n}$, where n is the number of flows. This small buffer was proposed taking advantage of the desynchronization of the flows. It eases the design of high-rate routers at low cost and with almost no link utilization degradation.

For both the above propositions mainly defined for backbone routers, even if they present several advantages and have wide adoption, the RRT value to use could be tricky. Unfortunately, when it comes to data centers switches buffer requirements, as far as we know, there is no rule-of-thumb. A widely common recommendation is to use in data centers relatively small buffers to achieve high bandwidth, and low latency [20]–[22]. But due to the critical aspect of buffer size on performance, buffer sizing needs more attention.

C. Incast Performance Optimization Algorithm

With a focus on the switch buffer size B, the optimization problem consists of finding the best B* that minimizes τ for a given combination of the other parameters for an incast traffic denoted as $P' = P'_j$. The optimization procedure is done in three steps: i) Generate a set of candidates B, ii) Evaluate the resulting incast completion time of each of them, and iii) selects the one that satisfies the optimization objective.

The algorithm procedure is presented in Algorithm 1. For any incast traffic, the corresponding parameters are collected. The expected incast completion time is inferred with the Random forest model (constructed in the previous section) for various candidates B. After this evaluation, the best-fitted candidate B* is used by the SDN controller to dynamically adjust the switch buffer space to handle the incast traffic efficiently.

VI. Performance Evaluation

A. Setup and Evaluation Metrics

We generate the working dataset (presented in Section IV-A) from intensive NS-3 simulations. These simulations use the mixed incast-elephant traffic scenario from Fig. 1. A random forest incast performance model is constructed using this dataset. Its evaluation results and its effectiveness for the incast performance optimization are provided in the next sections.

We consider three evaluation metrics. The first is the prediction **score**. It represents which proportion of the variance in the dependent variable is predictable from the independent variables. The most precise regression model would be the one that has a relatively high R squared, close to 100, when expressed in percentage. We will represent the score in percentage. Secondly, we will use **NMAE** for Normalized Mean Absolute Error expressed in percentage. We want the NMAE to be as small as possible. And finally, we will consider the **relative prediction error**: $\frac{|y_i - \hat{y}_i|}{y_i}$.

B. ML Prediction Accuracy

For a subset with n_samples of 10000, where 7000 samples are used to train the random forest model, we obtain the scores and NMAE on the 3000 remaining set as presented in TABLE II. Using all the dataset (20800 samples) generated through NS-3 simulations gives scores around 99% with NMAE of around 1%. These results prove the ability of the ML-performance approach to provide accurate predictions.

TABLE II
ML PREDICTIONS ACCURACY

Categories	**Score (%)**	**NMAE (%)**
NewReno_FIFO	96.54	9.81
NewReno_FQ	96.95	9.07
Cubic_FIFO	96.90	9.26
Cubic_FQ	96.23	10.98

The prediction score and NMAE provide a good picture of the prediction accuracy. But they don't give detailed information on the model behavior. This information is provided in Fig. 3 which presents the CDF (Cumulative Distribution Function) of the relative errors over all the evaluation samples. This distribution of residuals shows that the prediction error is very low for most of the test data points for the 4 categories.

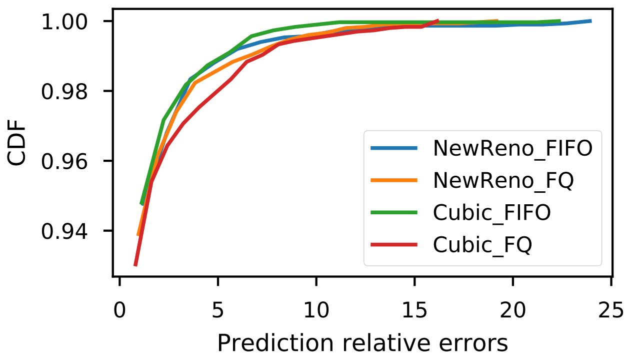

Fig. 3. CDF of predictions relative errors

In Fig. 4, we show For Cubic_FQ the real observed test points and their corresponding predictions with the random forest model. This regression plot confirms the significant relative errors observed from the CDF plot.

C. Optimization Evaluation Results

To evaluate the effectiveness of our optimization algorithm Algorithm 1, we compute B* for diverse values of N, with all the other parameters fixed. The algorithm output is presented in TABLE III. We will compare the performance observed

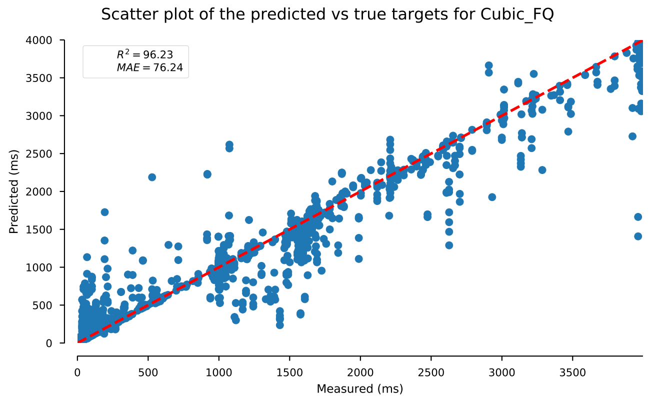

Fig. 4. Predictions vs. Real observations

using this optimal B versus when using the maximum available buffer space B_{max}.

TABLE III
OPTIMAL BUFFER SPACE B* FROM INCAST PERFORMANCE OPTIMIZATION ALGORITHM

N	1	4	8	16	20	32	48	64	100
B*	4	16	64	64	40	32	64	25	64

Fig. 5 shows the incast completion time τ when using Algorithm 1 against over-provisioning using B_{max}. We can observe that τ is almost the same with both B* and B_{max}. It follows from these results that, with our framework, we could achieve great incast performance while preventing buffer wastage that may occur when over-provisioning. Moreover, sometimes, using the maximum available buffer space could degrade the performance (N = 32 on Fig. 5).

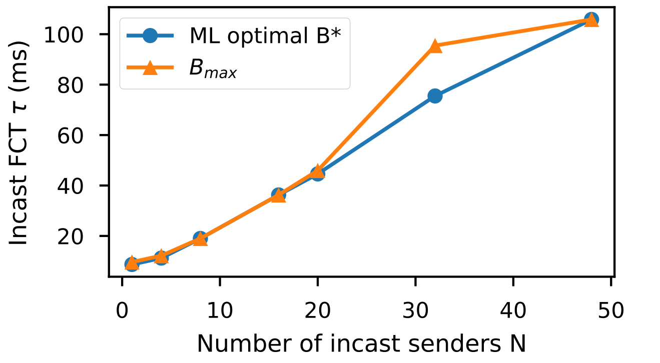

Fig. 5. Incast FCT Optimization for diverse number of incast senders

VII. RELATED WORKS

Prior works have explored the use of ML to provide models for network and performance optimization. The work in [23] proposes RouteNet that leverages the ability of Graph Neural Networks (GNN) for network modeling and optimization in SDN. Taking as input network topology information, routing schemes, and traffic matrix RouteNet, based on Generalized Linear Models, can provide accurate end-to-end performance metric predictions such as delay distribution (mean delay and jitter) and packet drop prediction. These predictions could then be leveraged by QoS-aware global performance optimization.

The work in [24] explores if Deep Reinforcement Learning (DRL) can be used for automatic traffic optimization in datacenters. Their preliminary study shows that The DRL approach's high latency is an obstacle to traffic optimization at the scale of current datacenters. Leveraging long-tail distribution of datacenter workload, they propose AuTO mimicking the Peripheral and Central Nervous Systems in animals to solve this scalability problem. Their work focuses on flow scheduling and load balancing. They adopt Multi-Level Feedback Queueing by optimizing its thresholds.

The study in [14] evaluates whether ML offers a general and straightforward approach to performance prediction. It assesses 6 ML performance prediction models across 13 real-world applications. The authors of [14] show that many applications exhibit a surprisingly high degree of irreducible prediction error. And they propose a more nuanced methodology for applying ML for performance prediction.

VIII. Conclusion And Future Research Directions

In this work, we propose an SDN-enabled machine-learning-based optimization framework for incast performance optimization in datacenter networks. It allows optimization on unknown or difficult to obtain analytical performance models by exploiting machine learning predictions. Evaluations based on intensive NS-3 simulations show that we can achieve accurate incast performance predictions. These predictions enable finding the efficient switch buffer space to achieve optimal incast completion time in different configurations. With the optimal switch buffer space, we achieve great incast performance while preventing buffer wastage that may occur when over-provisioning. We expect this framework to be a building block for autonomous data center management.

As future works, while this paper focuses on the use of ML in the optimization process of incast performance only, multi-objective optimization of both incast and elephant flows can be developed. We will also investigate other black-box optimization algorithms (e.g., Genetic Algorithms) in the optimization process.

References

[1] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 29–42.

[2] Y. Zhang and N. Ansari, "On architecture design, congestion notification, tcp incast and power consumption in data centers," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 39–64, 2012.

[3] "How to strike the right balance between application performance and cost," https://blogs.cisco.com/cloud/how-to-strike-the-right-balance-between-application-performance-and-cost?ccid=cc001268, 2020.

[4] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in sdn," in *Proceedings of the 2019 ACM Symposium on SDN Research*, 2019, pp. 140–151.

[5] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding tcp incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, 2009, pp. 73–82.

[6] Y. Li, R. Miao, M. Alizadeh, and M. Yu, "{DETER}: Deterministic {TCP} replay for performance diagnosis," in *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, 2019, pp. 437–452.

[7] J. Zhang, F. Ren, and C. Lin, "Modeling and understanding tcp incast in data center networks," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 1377–1385.

[8] W. Chen, F. Ren, J. Xie, C. Lin, K. Yin, and F. Baker, "Comprehensive understanding of tcp incast problem," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1688–1696.

[9] K. B. Nougnanke, Y. Labit, M. Bruyère, S. Ferlin, and U. Aivodji, "Learning-based incast performance inference in software-defined data centers," in *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2021, pp. 118–125.

[10] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of tcp throughput collapse in cluster-based storage systems." in *FAST*, vol. 8, 2008, pp. 1–14.

[11] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.

[12] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 393–430, 2018.

[13] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Network*, vol. 32, no. 2, pp. 92–99, 2017.

[14] S. Fu, S. Gupta, R. Mittal, and S. Ratnasamy, "On the use of ML for blackbox system performance prediction," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/fu

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[16] P. Chuprikov, S. Nikolenko, and K. Kogan, "Towards declarative self-adapting buffer management," *ACM SIGCOMM Computer Communication Review*, vol. 50, no. 3, pp. 30–37, 2020.

[17] "Sizing the buffer," https://blog.apnic.net/2019/12/12/sizing-the-buffer/.

[18] C. Villamizar and C. Song, "High performance tcp in ansnet," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 45–60, 1994.

[19] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 281–292, 2004.

[20] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 conference*, 2010, pp. 63–74.

[21] C. Raiciu and G. Antichi, "Ndp: rethinking datacenter networks and stacks two years after," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 5, pp. 112–114, 2019.

[22] A. Shpiner and E. Zahavi, "Race cars vs. trailer trucks: Switch buffers sizing vs. latency trade-offs in data center networks," in *2016 IEEE 24th Annual Symposium on High-Performance Interconnects (HOTI)*. IEEE, 2016, pp. 53–59.

[23] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet: Leveraging graph neural networks for network modeling and optimization in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2260–2270, 2020.

[24] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 conference of the ACM special interest group on data communication*, 2018, pp. 191–205.