Optimizing in the Dark: Learning Optimal Network Resource Reservation Through a Simple Request Interface

Qiao Xiang, *Member, IEEE*, Haitao Yu, James Aspnes, Franck Le, Chin Guok, Linghe Kong, *Senior Member, IEEE*, and Y. Richard Yang, *Senior Member, IEEE*

***Abstract*— Network resource reservation systems are being developed and deployed, driven by the demand and substantial benefits of providing performance predictability for modern distributed applications. However, existing systems suffer limitations: They either are inefficient in finding the optimal resource reservation, or cause private information (*e.g.*, from the network infrastructure) to be exposed (*e.g.*, to the user). In this paper, we design BoxOpt, a novel system that leverages efficient oracle construction techniques in optimization and learning theory to automatically, and swiftly learn the optimal resource reservations without exchanging any private information between the network and the user. In BoxOpt, we first model the simple reservation interface adopted in most reservation systems as a resource membership oracle. Second, we develop an efficient algorithm that constructs a resource separation oracle by a linear number of calls on resource membership oracle. Third, we develop a generic framework to construct a resource optimization oracle by iteratively calling the resource separation oracle, and then develop three novel, efficient algorithms under this generic framework, the best of which computes the optimal resource reservation by a linear number of calls on resource separation oracle. As such, BoxOpt can discover the optimal resource reservation with $O(n^2)$ calls on the resource membership oracle. We implement a prototype of BoxOpt with and demonstrate its efficiency and efficacy via extensive experiments using real network topology and a 7-day trace from a large operational federation network. Results show that (1) BoxOpt has a 100% correctness ratio by comparing with a state-of-the-art optimization solver, and (2) for 90% of requests, BoxOpt learns the optimal resource reservation within 10 seconds.**

***Index Terms*— Machine learning, network optimization, resource orchestration, bandwidth reservation.**

Manuscript received March 25, 2019; revised January 28, 2020 and July 31, 2020; accepted November 16, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor R. Elazouzi. Date of publication January 6, 2021; date of current version April 16, 2021. This work was supported in part by NSFC under Grant 61702373, Grant 61672385, Grant 61701347, Grant U190820096, and Grant 72061127001; in part by the NSF awards under Grant CCF-1637385, Grant CCF-1650596, and Grant OAC-1440745; in part by the Key-Area Research and Development Program of Guangdong Province (No.2019B121204009), Guangdong Institute of Chinese Engineering Development Strategies (No. 2019-GD-13), FANet: PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications (No. LZC0019), and the Guangdong Basic and Applied Basic Research Foundation (No. 2019B1515120031); in part by the Facebook Research Award; and in part by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Grant W911NF-16-3-0001. *(Corresponding author: Qiao Xiang.)*

Qiao Xiang is with the School of Informatics, Xiamen University, Xiamen 361005, China, and also with the Department of Computer Science, Yale University, New Haven, CT 06520 USA (e-mail: qiao.xiang@cs.yale.edu).

Haitao Yu is with the Peng Cheng Laboratory, Shenzhen 518066, China, and also with the Department of Computer Science and Technology, Tongji University, Shanghai 200092, China (e-mail: haitao.yu@tongji.edu.cn).

James Aspnes is with the Department of Computer Science, Yale University, New Haven, CT 06520 USA (e-mail: james.aspnes@yale.edu).

Franck Le is with the IBM T. J. Watson Research Center, New York, NY 10562 USA (e-mail: fle@us.ibm.com).

Chin Guok is with the Lawrence Berkeley National Laboratory, Berkeley, CA 94720 USA (e-mail: chin@es.net).

Linghe Kong is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: linghe.kong@sjtu.edu.cn).

Y. Richard Yang is with the Peng Cheng Laboratory, Shenzhen 518066, China, and also with Department of Computer Science, Yale University, New Haven, CT 06520 USA (e-mail: yry@cs.yale.edu).

Digital Object Identifier 10.1109/TNET.2020.3045595

I. INTRODUCTION

MANY modern distributed applications (*e.g.*, [1], [2]) construct complex data flows between end hosts, *e.g.*, in data center networks. The key to supporting these applications is the ability to provide guaranteed network resources (*i.e.*, bandwidth) for performance predictability [3]. As such, many network resource reservation systems are developed and deployed [4]–[9]. One representative network reservation system deployment, and the targeted deployment network of this paper, is the Large Hadron Collider (LHC) [10], where an On-Demand Secure Circuits and Advance Reservation System called OSCARS [11] is deployed to support the large-scale dataset transfer in the LHC experiments. Such reservation systems are usually inelastic, *i.e.*, once a network resource reservation is made, the resources are dedicated to the application/user till the end of the reservation period specified by user, unless it is terminated by the network earlier due to emergency (*e.g.*, link failure). Depending on specific applications, typical reservation periods range from hours to days [10], [12]. As such, applications/users prefer to reserve the optimal amount of network resources when they make the request.

However, because of the underlying networks' concern of revealing sensitive information, existing reservation systems do not provide applications with an interface to access information of the underlying network infrastructure (*e.g.*, topology, links' available bandwidth). Instead, networks only offer a *simple reservation interface* for applications to submit requests for reserving a specific amount of bandwidths for a set of flows: $\mathrm{request}(\mathrm{flow\_set}, \mathrm{bw\_values})$), and returns either success or failure. A major concern of this design is its inefficiency for the applications/users to find the optimal amount of network resources to reserve. To further illustrate the issues, consider the example in Figure 1, where a user (*e.g.*, application) wants to determine and reserve the maximum achievable bandwidth for two flows from S_1 to D_1, and S_2 to D_2, respectively.

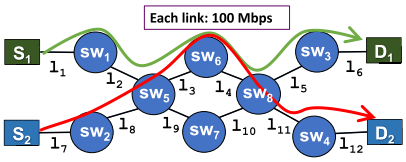


Fig. 1. An example network topology: the routes of two flows share bottleneck links, *i.e.*, l_3 and l_4, hence they can only collectively get a bandwidth of 100 Mbps.

Using existing solutions (*e.g.*, [4], [5]) that offer only a *simple reservation interface*, finding the constraint that both flows can collectively get only 100 Mbps of bandwidth is already an instance of the NP-hard membership-query based constraint acquisition problem [13], letting alone finding the optimal reservation for both flows (*e.g.*, 50 Mbps for each flow).

To address this problem, researchers have proposed several solutions, but all of them suffer limitations, and violate privacy requirements. For example, to determine optimal bandwidth reservations, recent proposals depart from the simple reservation interface, and require either networks to reveal sensitive information to users [12], [14]–[16], or vice versa [17]–[20]. These solutions are therefore limited to settings where the level of trust between the applications and the underlying network is high. These solutions cannot be deployed in general settings as malicious parties may use the exposed network information to identify vulnerable links and launch attacks (*e.g.*, DDoS).

In this paper, we explore the feasibility and benefits of learning the optimal network resource reservation for the user without exposing the private information of the network (*i.e.*, bandwidth capacity region) and the user (*e.g.*, resource orchestration policy) to each other. In particular, we tackle the following problem: *Given a user's objective function and the available resources in the network, how can a user learn the optimal network resource reservation using only the simple reservation interface?* In this problem, we use the term "learn" to refer to the process that given a user's objective function and the network's available resources, how to exploit and explore different reservation requests, based on the binary feedback (*i.e.*, success/failure) provided by the simple request interface, to find the optimal resource reservation.

This task is non-trivial due to the extremely limited feedback (*i.e.*, success/failure) provided by the simple reservation interface. Existing studies on optimization with unknown constraints are not suitable for addressing this problem. For example, Bayesian optimization based methods either require tens of thousands samplings to solve a problem with even a single unknown constraint (*e.g.*, [21]), or require additional knowledge on the number of constraints to scale (*e.g.*, [22]). Reinforcement learning based solutions (*e.g.*, [23]–[25]) are not appropriate candidates, either, for several reasons. First, the binary feedback provided by the simple reservation interface (*i.e.*, success/failure) makes it difficult to find a suitable reward function to ensure the convergence of the learning process. Second, the raw state space of the corresponding network resource reservation problem is exponential in terms of the number of flows the user wants to reserve resources for, resulting in complex, error-prone hyperparameter tuning process.

Our solution to this problem is BoxOpt, a novel learning system that automatically, and efficiently learns the optimal resource reservations for the user through the simple reservation interface, without exchanging any private information between the network and the user (*e.g.*, bandwidth feasible region of the network and the resource orchestration policy of the user). Specifically, BoxOpt allows users to include their resource reservation objectives as concave utility functions of the requested resources (*e.g.*, bandwidths) in the reservation requests. Upon receiving a reservation request, BoxOpt models the simple reservation interface of network resource reservation systems as a membership oracle over a polytope. It then expands oracle construction techniques [26], [27] from optimization and learning theory to construct a separation oracle through invoking the membership oracle in a linear number of iterations (n being the number of flows), which when called upon will accurately infer a search space in which the optimal reservation vector lies. With such a separation oracle, BoxOpt then constructs an optimization oracle, which can learn the optimal reservation vector through a polynomial number of calls on the separation oracle. The design of BoxOpt is modular so that different cutting-plane-based algorithms can be developed and plugged. Specifically, in this paper, we develop a generic optimization framework based on cutting plane methods, and develop three algorithms under this framework, each of which computes the optimal resource reservation by a linear or quadratic number of calls on resource separation oracle, respectively. Overall, given a user's objective function on network resource reservation and a network's bandwidth feasible region, BoxOpt not only can learn the optimal resource reservation efficiently (*i.e.*, with a polynomial total number of membership oracle calls), but also is privacy-preserving in that (1) the network does not know the objective function of the user, and (2) the user cannot reconstruct the bandwidth feasible region of the network because finding it via a membership oracle is NP-hard [13].

The **main contributions** of this paper are as follows:

- We study the important problem of learning the optimal network resource reservation through the simple reservation interface of network resource reservation systems. In particular, we design BoxOpt, a novel, fast, automatic, privacy-preserving learning system. To the best of our knowledge, BoxOpt is the first working system that solves this problem, can be extended to other optimization problems.
- In BoxOpt, we model the simple reservation interface as a membership oracle over a polytope, and expand oracle construction techniques from optimization and learning theory to develop an efficient algorithm that constructs a separation oracle in $O(n)$ calls on the membership oracle, and a series of algorithms, the best of which constructs an optimization oracle in $O(n)$ calls on the constructed separation oracle. As such, BoxOpt can discover the optimal resource reservation with $O(n^2)$ calls on the resource membership oracle.
- We implement a prototype of BoxOpt and demonstrate both its efficiency and efficacy through extensive experiments using real topologies and traces. Results show that (1) BoxOpt has a 100% correctness ratio by comparing with a state-of-the-art optimization solver (*i.e.*, CPLEX [28]), and (2) for 90% cases, it can learn the optimal reservation within 10 seconds.

The remaining of this paper is organized as follows. We present an overview of BoxOpt in Section III. We give

details on how BoxOpt efficiently learns the optimal network resource reservation only using the simple reservation interface in Section IV. We present the evaluation results of BoxOpt in Section V. We discuss related work in Section II and conclude the paper in Section VI.

II. Related Work

Network Resource Reservation Systems: Many network resource reservation systems have been developed and deployed [4]–[9], [12], [14]–[16]. Early systems mainly adopt a "blackbox-request" approach [4]–[9]. In particular, the user determine the amount of network resources (*e.g.*, bandwidth) needed for data flows based on their own constraints and objectives and submit requests for reserving the computed amount of resources through a simple interface (*i.e.*, $\mathrm{request}(\mathrm{flow}, \mathrm{bw\_req})$); and the network examines its own resource availability and returns success if the reservation request succeeds, or failure otherwise. This design is simple, but often makes it impossible for the user to find the optimal resources reservation.

One alternative to this design is called the "network-does-all" design. Specifically, systems using this design [12], [14]–[16] let the user submit their requirements, in terms of both constraints and objectives, on network resources to the network, and have the network compute and enforce the optimal network resource allocation for applications. One major concern of this design is, by sending all constraints and objectives to the network, the user expose her / his proprietary information (*e.g.*, internal data flow orchestration policy) to the network. For example, if a user submits a requirement that all data flows must have the same bandwidth, the network can infer that the user internally uses fairness policy. In addition, it requires the network to have the capability for solving the optimization problems specified by the user's requirements, which can be computationally expensive and time-consuming.

In addition, a third design called "resource-discovery" design was recently proposed [17], [19]. In this design, the user specify the endpoints of different data flows and submit to the network, and the latter returns the user with the resource availability and sharing information of these data flows using mathematic constraints as abstraction. As such, the user can perform their own resource orchestration algorithms to select the optimal amount of resources to reserve without revealing their internal constraints and objectives to the network. However, even though mathematic constraint is a compact abstraction, it still exposes the private information of the network (*e.g.*, the bandwidth feasible region) to the user.

Different from existing resource reservation systems, BoxOpt adopts a novel approach to efficiently learn the optimal resource reservation through the limited feedback from the simple interface provided by the "blackbox-request" reservation systems. BoxOpt enables the user to find the optimal amount of network resources to reserve without without exchanging any private information between the network and the user.

Machine Learning: Different machine learning techniques have been developed to solve optimization problems with unknown constraints [13], [21]–[25], [29]–[32]. However, they are not suitable for finding the optimal resource reservation via only the simple request interface. For example, Bayesian optimization based methods either require tens of thousands samplings to solve a problem with even a single unknown constraint (*e.g.*, [21]), or require additional knowledge on the number of constraints to scale (*e.g.*, [22]). Reinforcement learning based solutions (*e.g.*, [23]–[25]) are not appropriate candidates, either. This is because the binary feedback provided by the simple reservation interface (*i.e.*, success/failure) makes it difficult to find a suitable reward function to ensure the convergence of the learning process. In addition, the raw state space of the corresponding network resource reservation problem is exponential in terms of the number of flows the user wants to reserve resources for, resulting in complex, error-prone hyperparameter tuning process.

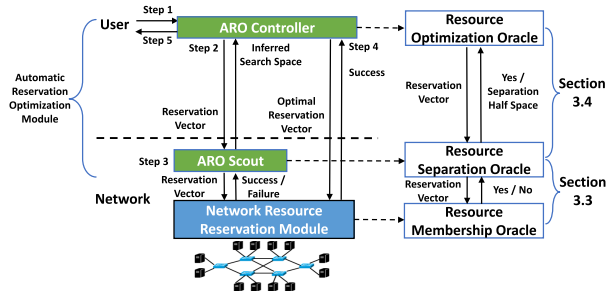


Fig. 2. The architecture and workflow of BoxOpt.

Another area in machine learning that is closely related to our problem is *constraint learning* [13], [29]–[32]. For example, [13] studies constraint acquisition via different types of membership queries, and show that this problem is in general NP-hard. [30], [32] study the membership problem of parameterized linear systems. We refer readers to [29] for a comprehensive survey. Instead of learning all linear inequalities that compose the bandwidth feasible region, as will be shown in the next few sections, BoxOpt leverages several powerful tools from optimization theory [26], [27], [33]. We expand the recent theoretical progress on efficient oracle constructions to a broader scenario. To the best of our knowledge, BoxOpt is the first working system that demonstrates the feasibility and benefits of learning the optimal solution of an optimization problem with only membership oracle. In addition to network resource reservation, it also sheds light for other areas such as multi-domain traffic engineering and collaborative data analytics.

III. Overview of BoxOpt

In this section, we first present the architecture and the workflow of BoxOpt. We then give a formal, mathematical formulation of the key technical challenge in BoxOpt: how to find the optimal network resource reservation through the simple reservation interface.

A. Architecture

BoxOpt is composed of two components: an automatic reservation optimization (ARO) module for the user, and a network resource reservation (NRR) module for the network (Figure 2). The two components interact with each other through the simple reservation interface commonly used in traditional network resource reservation systems.

Automatic Reservation Optimization Module: The ARO module is a private component belonging to the user, and is composed of two sub-components: an ARO controller, and an ARO scout.

The ARO controller is the main interface for the user to submit the resource reservation requests. A request consists of a set of n flows, $F = \{f_1, f_2, \ldots, f_n\}$, to reserve the resources for, and a concave utility function $util(\mathbf{x})$ to maximize, with $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and each x_i representing the available bandwidth that can be reserved for flow $f_i \in F$. Example utility functions include total throughput and priority-based total throughput. Given a user resource reservation request, the objective of the ARO controller is to infer the optimal resource reservation to maximize $util(\mathbf{x})$. The ARO controller achieves it with the assistance of the ARO scout. Specifically, the ARO controller iteratively selects a vector $\tilde{\mathbf{x}}$ of bandwidth values for F (called *reservation vector*) and sends it to the ARO scout. For each reservation vector, the ARO scout returns a search space where the optimal reservation vector lies in. With the inferred search spaces returned by the ARO scout, the ARO controller gradually converges to the optimal reservation vector that maximizes $util(\mathbf{x})$.

The ARO scout is the main user entity interacting with the NNR. For each $\tilde{\mathbf{x}}$ from the ARO controller, the ARO scout infers a search space where the optimal reservation vector lies in, and returns the inferred search space back to the ARO controller. To infer the search space where the the optimal reservation vector lies in, the ARO scout sends a sequence of reservation vectors to the NRR through the simple reservation interface. As further described in Section IV and Section V, for each reservation vector submitted from the ARO controller, the ARO scout might submit tens or hundreds of reservation vectors to the NNR to get an accurately-inferred search space, potentially, leading to a high overhead. As such, to reduce the total latency to find the optimal reservation vector, we design the ARO scout as a lightweight process that can be placed in the network (*e.g.*, a container instance), instead of with the user. This design decision reduces the user-network communication latency by over 220 times as demonstrated in the evaluation section. More importantly, this design does not expose the private information of the user (*i.e.*, $util(\mathbf{x})$) to the network, as the ARO controller does not send such information to the scout.

Network Resource Reservation Module: The NRR module is a private component belonging to the network. Its primary role is to verify whether the reservation vectors submitted by the ARO scout can be satisfied. Upon receiving a reservation vector from the ARO scout, the NRR extracts the relevant constraints from the network. The constraints include both physical network constraints (*e.g.*, if two flows share a same link, their allocated bandwidths cannot exceed the link's available bandwidth), and network policies (*e.g.*, rate limiting, etc.) The constraints are captured as an abstraction of linear inequalities [17]–[19]. For example, to capture the physical network constraints, the NNR first retrieves the routes (i.e., sequence of traversed links) for each flow. Then, for each link l in the network, the NNR generates the following linear inequality to ensure that the allocated bandwidths to the flows do not exceed the link's available bandwidth:

$$\sum x_i \leq w_l, \quad \forall f_i \text{ that uses } l \text{ in this route},$$

where w_l is the available bandwidth on link l. Considering the example in Figure 1, the NRR module generates the following linear inequalities:

$$\begin{aligned} x_1 &\leq 100 \quad \forall l_u \in \{l_1, l_2, l_5, l_6\}, \\ x_2 &\leq 100 \quad \forall l_u \in \{l_7, l_8, l_{11}, l_{12}\}, \\ x_1 + x_2 &\leq 100 \quad \forall l_u \in \{l_3, l_4\}, \\ x_1, x_2, x_3 &\geq 0. \end{aligned} \tag{1}$$

Then, the NRR generates additional linear inequalities to represent the network's internal traffic engineering policies, such as load-balancing and bandwidth limiting. For example, suppose the network wants to limit the total bandwidth of flows f_1 and f_2 to be no more than 80 Mbps even if there is no common link in their routes. Then a linear inequality $x_1 + x_2 + x_3 \leq 80$ is generated to represent this policy. Geometrically, the abstraction of linear inequalities represents the *bandwidth feasible region* of the network for providing bandwidths to a set of flows.

Finally, for each generated linear inequality, the NRR checks if it is satisfied by the bandwidth values specified in the reservation vector. If any inequality is violated, it returns a FAILURE signal. Otherwise, it returns SUCCESS.

To handle concurrent reservation requests, the NRR module instantiates multiple threads, each of which maintains a soft state of available network bandwidths and interacts with the ARO module of one request. When two requests want to reserve the same resources (*e.g.*, all bandwidth of a same link), only the earliest request will be satisfied. As such, when the network has high dynamics (*e.g.*, large number of simultaneous reservation requests and short reservation duration), BoxOpt may not be able to find the optimal resource reservations for concurrent requests. However, as we will show in Section V, given the long lasting time of reservations and the short time of BoxOpt to find the optimal reservation, the probability of many reservations competing the same resources is low (*e.g.*, none in the production trace used in the evaluation).

B. Workflow

Having presented the basic components of BoxOpt, we now briefly present its workflow to automatically compute and reserve the optimal network resources for a set of flows as follows (Figure 2):

- Step 1: The user submits a resource reservation request for a set of flows F to the ARO controller. The request also includes a concave utility function $util(\mathbf{x})$ of the bandwidths of F.
- Step 2: In an outer loop, the ARO controller iteratively selects reservation vectors to send to the ARO scout. The selection of the reservation vectors is described in Section IV-D. In return, for each reservation vector, the ARO scout determines and replies with an inferred search space.
- Step 3: In an inner loop, upon receiving a reservation vector from the controller, the ARO scout interacts with the NRR, according to Algorithm 1 from Section IV-C, to infer the next search space and send it back to the ARO controller.

The nested iteration of Step 2 and 3 stops when the ARO controller converges to the optimal reservation vector maximizing $util(\mathbf{x})$.

• Step 4: The ARO controller sends the optimal reservation vector to the NRR module to reserve the optimal resources for the user.

• Step 5: The ARO controller confirms with the user that the optimal network resource reservation has been successful.

C. Key Challenge

Through the introduction of its architecture and workflow, we show that the key challenge for BoxOpt lies in Step 2 and 3: *how can the ARO module interact with the NRR module through the simple reservation interface to compute the optimal network resource reservation?*. To address this challenge, we first give a formal, mathematical formulation.

Specifically, we first model the NRR module as a *resource membership oracle*. Without loss of generality, we use $\mathbf{Ax} \leq \mathbf{b}$ to denote the set of linear inequalities generated by the NRR module, and use $K : \{\mathbf{x}|\mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ to represent the bandwidth feasible region for a set of flows F. In this way, we give the definition of resource membership oracle:

Definition 1 (Reservation Membership Oracle (ReMEM)): Given a reservation vector $\check{\mathbf{x}}$, return YES if $\check{\mathbf{x}} \in K$, and return NO otherwise.

$ReMEM(\check{\mathbf{x}})$ accurately captures the interaction between the ARO scout and the NRR module. Next, we formally define the problem of network resource reservation optimization via simple reservation interface.

Problem 1 (Optimization via Membership Oracle): Find the optimal solution to the following optimization problem

$$\textit{maximize } util(\mathbf{x}), \tag{2}$$

subject to,

$$\mathbf{Ax} \leq \mathbf{b}, \tag{3}$$

$$\mathbf{x} \geq \mathbf{0}, \tag{4}$$

without the knowledge of $\mathbf{A}$ and $\mathbf{b}$, but only using ReMEM defined in Definition 1.

Maximizing $util(\mathbf{x})$ subject to $K : \{\mathbf{x}|\mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ is a classic convex optimization problem. There has been a rich body of literature on how to efficiently solve such problems [33]. However, most of the existing algorithms require the knowledge of the feasible region (in our case $\mathbf{Ax} \leq \mathbf{b}$). One may think of a strawman to learn K through the ReMEM oracle, and apply the standard optimization techniques to find the optimal $\mathbf{x}$. However, finding the feasible region through a membership oracle is NP-hard [13], making this strawman impractical.

In contrast, as we will present next, BoxOpt resorts to efficient oracle transformation techniques in optimization and learning theory [26], [27] to solve this problem efficiently, *i.e.*, learn the optimal resource reservation via a polynomial number of calls on the ReMEM oracle. In addition, we also show that given a user's objective function on network resource reservation and a network's bandwidth feasible region, BoxOpt is *privacy-preserving*: (1) the internal objective function of the user is not exposed to the network, and (2) with the NP-hardness of finding the feasible region through a membership oracle [13] and the polynomial total number of calls on the ReMEM oracle by BoxOpt, the user cannot reconstruct the bandwidth feasible region of the network.

IV. Optimizing Network Resource Reservation via Simple Reservation Interface

Having formally defined the key challenge for BoxOpt as a problem of optimization via membership oracle, this section discusses how we solve this problem. For presentation clarity, this section starts by reviewing some concepts in optimization theory. Then, it presents the basic idea of our solution, followed by its details.

A. Notations

Unless explicitly noted, we use v to denote a scalar and $\mathbf{v}$ to denote a vector of n dimensions, where n is the number of flows the user wants to reserve bandwidth for (see Section III-A). We use $\|\mathbf{v}\|_2 = \sqrt{\sum v_i^2}$ to denote the Euclidean norm of $\mathbf{v}$, and use $\|\mathbf{v}\|_\infty = \max |v_i|$ to denote the maximum norm of $\mathbf{v}$. We use $B_2^+(\mathbf{m}, \eta) = \{\mathbf{x}|\|\mathbf{x} - \mathbf{m}\|_2 \leq \eta, \mathbf{x} \geq \mathbf{0}\}$ to denote the set of all positive vectors whose Euclidean distance to $\mathbf{m}$ is at most η, and use $B_\infty^+(\mathbf{m}, \eta) = \{\mathbf{x}|\|\mathbf{x} - \mathbf{m}\|_\infty \leq \eta, \mathbf{x} \geq \mathbf{0}\}$ to denote the set of all positive vectors whose maximum norm distance to $\mathbf{m}$ is at most η.

B. Basic Idea

Our approach to solve Problem 1 utilizes the equivalence and polar relationships between different oracles in optimization theory [26]. In particular, we focus on the relationships between ReMEM with the following two oracles:

Definition 2 (Resource Separation Oracle (ReSEP)): Given a reservation vector $\check{\mathbf{x}}$, return YES if $\check{\mathbf{x}} \in K$, and otherwise return a half space $\{\mathbf{y}|\mathbf{p}^T(\mathbf{y} - \check{\mathbf{x}}) \leq \delta\}$ that contains K but not $\check{\mathbf{x}}$.

Definition 3 (Resource Optimization Oracle (ReOPT)): Given a reservation request for a set of flows F and the utility function $util(\mathbf{x})$, find $\mathbf{x}^* \in K$ that maximizes $util(\mathbf{x})$.

Given that there exist efficient algorithms (*e.g.*, ellipsoid method) that can construct an optimization oracle through invoking a separation oracle with a polynomial number of iterations, if we can construct a separation oracle through a polynomial number of calls to a membership oracle, we will be able to solve Problem 1.

One may think classic half space learning techniques can achieve such a construction of separation oracle via membership oracle. However, the problems are different. In half space learning, the goal is to compute a hyperplane to separate a set of given samples (in our case, the reservation vectors) from two predefined classes. In contrast, the goal of ReMEM to ReSEP construction is to compute a hyperplane separating K and a reservation vector not belonging to K by strategically choosing a minimal number of reservation vectors to send to ReMEM.

Specifically, we develop our solution to Problem 1 in two phases. First, we leverage recent progress on geometric algorithms [27] to develop an efficient algorithm that constructs ReSEP through invoking ReMEM for a polynomial number of times. Specifically, our algorithm expands the weak membership/separation oracle construction in [27] to strong membership/separation oracle construction. Second, we develop a generic framework, which constructs ReOPT through local feasibility checks and invoking ReSEP for a polynomial number of times, and develop three novel, efficient

Algorithm 1 Resource Separation Oracle $ReSEP(\check{\mathbf{x}})$

1 Select $\epsilon \in (0, r], \rho \in (0, 1)$;
2 $\kappa \leftarrow \frac{R}{r}$;
3 **if** *ReMEM returns YES for* $\check{\mathbf{x}}$ **then**
4 **return** $\check{\mathbf{x}} \in K$;
5 **else if** $\check{\mathbf{x}} \notin B_2(\mathbf{0}, R)$ **then**
6 **return** the half space $\{\mathbf{y}|\check{\mathbf{x}}^T(\mathbf{y} - \check{\mathbf{x}}) \leq 0\}$;
7 **else**
8 $r_1 \leftarrow rR^{-\frac{1}{3}}\epsilon^{\frac{1}{3}}$;
9 $\tilde{\mathbf{g}} \leftarrow Subgradient(\mathbf{0}, r_1, 4\epsilon, 3\kappa)$;
10 **return** the half space $\{y|\tilde{\mathbf{g}}^T(\mathbf{y} - \check{\mathbf{x}}) \leq (40n + 1)\rho^{-1}R^{\frac{2}{3}}\kappa\epsilon^{\frac{1}{3}}\}$;

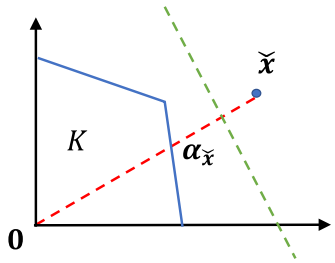


Fig. 3. An illustration of the auxiliary functions $\alpha_{\check{\mathbf{x}}}(\mathbf{d})$ and $h_{\check{\mathbf{x}}}(\mathbf{d})$.

algorithms under this framework, each of which, computes the optimal resource reservation by a linear or quadratic number of calls on resource separation oracle, respectively. Mapping these two phases to Step 2 and 3 in the workflow of BoxOpt, we see that the ARO scout is essentially the separation oracle ReSEP, and the ARO controller is the optimization oracle ReOPT (Figure 2). Next, we give the details of each phase.

C. From Resource Membership Oracle to Resource Separation Oracle

To construct ReSEP from ReMEM, we first define two auxiliary functions on vector $\mathbf{d} \in K$ given a reservation vector $\check{\mathbf{x}}$:

$$\alpha_{\check{\mathbf{x}}}(\mathbf{d}) \leftarrow \max_{\mathbf{d}+\alpha\check{\mathbf{x}} \in K} \alpha, \tag{5}$$

$$h_{\check{\mathbf{x}}}(\mathbf{d}) \leftarrow -\alpha_{\check{\mathbf{x}}}(\mathbf{d})\|\check{\mathbf{x}}\|_2 \tag{6}$$

As illustrated in Figure 3, given a vector $\mathbf{d} \in K$, $\mathbf{d} + \alpha_{\check{\mathbf{x}}}(\mathbf{d})$ is the last vector on the line from $\mathbf{d}$ to $\mathbf{d} + \check{\mathbf{x}}$ that is in K, and that $-h_{\check{\mathbf{x}}}(\mathbf{d})$ is the Euclidean distance from d to this point. Without loss of generality, we assume that $B_2^+(\mathbf{0}, r) \subset K \subset B_2^+(\mathbf{0}, R)$ for some positive numbers r, R and such an assumption can be trivially satisfied in practice. Extending the proof technique in [27] for an n-dimensional ball to only a partial ball on the first orthant, we get

Lemma 1: Given $\check{\mathbf{x}}$, $h_{\check{\mathbf{x}}}(\mathbf{d})$ *is convex on* K, *and is* $\frac{R+\theta}{R-\theta}$ *Lipschitz in* $B_2^+(\mathbf{0}, \theta)$ *for* $0 < \theta < r$.

With these auxiliary functions and a theorem that for any Lipschitz function, it is linear on a small ball [34], we can construct ReSEP by computing the subgradient of $h_{\check{\mathbf{x}}}(\mathbf{d})$ at $\mathbf{d} = \mathbf{0}$, which can be computed by binary search and invoking ReMEM. The constructed separation oracle is presented in Algorithm 1, and the computation of the subgradient of $h_{\check{\mathbf{x}}}(\mathbf{d})$ is presented in Algorithm 2.

Algorithm 2 Computing the Subgradient of $h_{\check{\mathbf{x}}}(\mathbf{d})$ $Subgradient(\mathbf{d}, r_1, \tau, L)$

1 $r_2 \leftarrow \sqrt{\frac{\tau r_1}{\sqrt{n}L}}$;
2 Randomly select $\mathbf{y}$ from $B_\infty(\mathbf{d}, r_1)$ following a uniform distribution;
3 Randomly select $\mathbf{z}$ from $B_\infty(\mathbf{y}, r_2)$ following a uniform distribution;
4 **for** $i \leftarrow 1, \ldots, n$ **do**
5 Define line segment $B_\infty(\mathbf{y}, r_2) \cap \mathbf{z} + s\mathbf{e_i}$, where $s \in R$ and $\mathbf{e_i}$ is a vector whose elements are all zeros except the ith one;
6 Denote the endpoints of this line segment as $\mathbf{s_i}$ and $\mathbf{t_i}$, respectively;
7 Evaluate $h_{\check{\mathbf{x}}}(\mathbf{t_i})$ and $h_{\check{\mathbf{x}}}(\mathbf{s_i})$ using binary search and ReMEM;
8 $\tilde{g}_i = \frac{h_{\check{\mathbf{x}}}(\mathbf{t_i}) - h_{\check{\mathbf{x}}}(\mathbf{s_i})}{2r_2}$;
9 **return** $\tilde{\mathbf{g}}$;

A key insight in Algorithm 1 is that it expands the applicability of similar construction process from weak membership/separation oracles to strong membership/separation oracles (*i.e.*, ReMEM and ReSEP). In particular, we have

Lemma 2: There is a random variable ϕ *with expectation* $E(\phi) \leq 2\ n\sqrt{\frac{\tau L}{r_1}}$ *such that* $\forall \mathbf{q} \in K$,

$$h_{\check{\mathbf{x}}}(\mathbf{q}) \geq h_{\check{\mathbf{x}}}(\mathbf{d}) + \tilde{\mathbf{g}}^T(\mathbf{q} - \mathbf{d}) - \phi\|\mathbf{q} - \mathbf{d}\|_\infty - 4nr_1L.$$

With this lemma, we show the correctness of Algorithm 1 in the following theorem.

Theorem 1: If $\check{\mathbf{x}} \notin K$, *Algorithm 1 yields a half space containing* K *but not* $\check{\mathbf{x}}$ *with probability* $1 - \rho$.

Proof: When $\check{\mathbf{x}} \notin B_2^+(\mathbf{0}, R)$, it is easy to see that the returned half space $\{\mathbf{y}|\check{\mathbf{x}}(\mathbf{y}) - \check{\mathbf{x}} \leq \mathbf{0}\}$ (Line 6 of Algorithm 1) contains K but not $\check{\mathbf{x}}$. When $\check{\mathbf{x}} \notin K$ but $\check{\mathbf{x}} \in B_2^+(\mathbf{0}, R)$, from Lemma 1, we know that $h_{\check{\mathbf{x}}}(\mathbf{d})$ has a Lipschitz constant of 3κ on $B_2^+(\mathbf{0}, \frac{r}{2})$. By selecting $\epsilon \in (0, r]$ and setting $r_1 = rR^{-\frac{1}{3}}\epsilon^{\frac{1}{3}}$ (Line 1 and 8 of Algorithm 1, respectively), we can get $B_\infty^+(\mathbf{0}, 2r_1) \subset B_2^+(\mathbf{0}, \frac{r}{2})$. As such, we can apply Lemma 2 and get that $\forall \mathbf{q} \in K$

$$h_{\check{\mathbf{x}}}(\mathbf{q}) \geq h_{\check{\mathbf{x}}}(\mathbf{0}) + \tilde{\mathbf{g}}^T \cdot \mathbf{q} - \phi\|\mathbf{q}\|_\infty - 12nr_1\kappa. \tag{7}$$

Next, because $\check{\mathbf{x}} \in B_2^+(\mathbf{0}, R)$, we have $-\frac{1}{\kappa}\check{\mathbf{x}} \in K$ and $h_{\check{\mathbf{x}}}(-\frac{1}{\kappa}\check{\mathbf{x}}) = h_{\check{\mathbf{x}}}(\mathbf{0}) - \frac{\|\check{\mathbf{x}}\|_2}{\kappa}$. Then we use Lemma 2 to get

$$h_{\check{\mathbf{x}}}(-\frac{1}{\kappa}\check{\mathbf{x}}) \geq h_{\check{\mathbf{x}}}(\mathbf{0}) + \tilde{\mathbf{g}}^T \cdot -\frac{1}{\kappa}\check{\mathbf{x}} - \frac{\phi}{\kappa}\|\check{\mathbf{x}}\|_\infty - 12nr_1\kappa, \tag{8}$$

As such, we then get

$$\tilde{\mathbf{g}}^T \cdot \check{\mathbf{x}} \geq \|\check{\mathbf{x}}\|_2 - \phi\|\check{\mathbf{x}}\|_\infty - 12nr_1\kappa^2. \tag{9}$$

Next, because $\epsilon \in (0, r]$, $\check{\mathbf{x}} \notin K$ and $B_2^+(\mathbf{0}, r) \subset K$, we have $(1 - \frac{\epsilon}{r})K \subset K$. By definition of $h_{\check{\mathbf{x}}}(\mathbf{d})$, we have

$$h_{\check{\mathbf{x}}}(\mathbf{0}) \geq -(1 - \frac{\epsilon}{r})\|\check{\mathbf{x}}\|_2 \geq -\|\check{\mathbf{x}}\|_2 + \epsilon\kappa. \tag{10}$$

Adding Equations (9) and (10) and then subtracting $2\epsilon\kappa$ on the right hand side, we get

$$h_{\check{\mathbf{x}}}(\mathbf{0}) + \tilde{\mathbf{g}}^T \cdot \check{\mathbf{x}} \geq -\phi\|\check{\mathbf{x}}\|_\infty - 12nr_1\kappa^2 - \epsilon\kappa. \tag{11}$$

Next, we add Equation (11) to Equation (7) and get that $\forall \mathbf{q} \in K$,

$$\begin{aligned} h_{\check{\mathbf{x}}}(\mathbf{q}) &\geq \tilde{\mathbf{g}}^T(\mathbf{q}-\check{\mathbf{x}}) - \phi\|\mathbf{q}\|_\infty - \phi\|\check{\mathbf{x}}\|_\infty \\ &\quad -12nr_1\kappa - 12nr_1\kappa^2 - \epsilon\kappa \\ &\geq \tilde{\mathbf{g}}^T(\mathbf{q}-\check{\mathbf{x}}) - 2\phi R - 24nr_1\kappa^2 - \epsilon\kappa. \end{aligned} \tag{12}$$

$\forall \mathbf{q} \in K$, we have $h_{\check{\mathbf{x}}}(\mathbf{q}) \leq 0$, and then we can have $\tilde{\phi} \geq \tilde{\mathbf{g}}(\mathbf{q}-\check{\mathbf{x}})$, where $\tilde{\phi}$ is a random scalar independent of $\mathbf{q}$ that satisfies

$$E(\tilde{\phi}) \leq 4\sqrt{\frac{12\epsilon\kappa}{r_1}}nR + 24nr_1\kappa^2 - \epsilon\kappa. \tag{13}$$

Putting $r_1 = rR^{-\frac{1}{3}}\epsilon^{\frac{1}{3}}$ into Equation (13), we get

$$E(\tilde{\phi}) \leq 40n\epsilon^{\frac{1}{3}}R^{\frac{2}{3}}\kappa + \epsilon\kappa. \tag{14}$$

Further leveraging $\epsilon \leq r \leq R$, we get

$$E(\tilde{\phi}) \leq (40n+1)\epsilon^{\frac{1}{3}}R^{\frac{2}{3}}\kappa. \tag{15}$$

Then we can finish the proof using Equation (15) and Markov inequality. □

In addition, observing Algorithm 1 and Algorithm 2, we see that the bottleneck to construct ReSEP is to compute $h_{\check{\mathbf{x}}}$ using binary search and ReMEM (Line 4-8 in Algorithm 2). As such, we give the following theorem on the complexity of Algorithm 1.

Theorem 2: Algorithm 1 constructs the reservation separation oracle (ReSEP) through an $O(n \log R)$ calls on the reservation membership oracle (ReMEM).

D. From Resource Separation Oracle to Resource Optimization Oracle

Having constructed ReSEP from ReMEM, we next focus on how to construct the resource optimization oracle ReOPT from the resource separation oracle ReSEP. To this end, we first develop a generic optimization framework, and then develop a series of algorithms to accomplish this goal under this framework.

Generic Framework: We build our generic ReOPT construction framework based on a cutting-plane-method based framework. The cutting-plane method is a family of optimization methods that iteratively refine the feasible region by means of a half space [26]. Specifically, the framework involves two main steps in each iteration:

1. Select a *center* from a *bounding box*;
2. Invoke a separation oracle to exam if the chosen center is a feasible solution, and find a new cutting plane based on the feedback from the separation oracle to refine the bounding box in the next iteration.

In each iteration, the bounding box is refined into a smaller one. And the iteration process stops when the bounding box becomes smaller than a predefined size (*i.e.*, predefined error range). Different cutting-plane methods have different ways to select the bounding box and the center, yet most of them use the same approach to find a new cutting plane. And we summarize it in the FINDCUTTINGPLANE algorithm (*i.e.*, Algorithm 3).

Specifically, this approach starts by sending the chosen center $\mathbf{p}$ to a separation oracle (*i.e.*, the ARO scout running the ReSEP developed in Section IV-C) (Line 1-2). If $\mathbf{p}$ is not in the feasible region K, it chooses the returned half space from ReSEP as the new cutting plane (Line 2). Otherwise, it update the current optimal value u_{best} by comparing it with $util(\mathbf{p})$, and computes the subgradient of the chosen center as the new cutting plane (Line 4-7).

Algorithm 3 FindCuttingPlane($\mathbf{p}, \mathbf{p}^*, u_{best}$)

1 **if** $ReSEP(\mathbf{p})$ *returns a half space* H' **then**
2 $H \leftarrow H'$;
3 **else**
4 **if** $util(\mathbf{p}) > u_{best}$ **then**
5 $\mathbf{p}^* \leftarrow \mathbf{p}$;
6 $u_{best} \leftarrow util(\mathbf{p}^*)$;
7 $H \leftarrow \{\mathbf{x} | (\nabla util(\mathbf{p}))^T(\mathbf{x}-\mathbf{p}) >= 0\}$;
8 return $H, \mathbf{p}^*, u_{best}$

Algorithm 4 ReOPT-EL: A Resource Optimization Oracle Using Ellipsoid as Bounding Box and Its Center to Find New Cutting Plane

1 Given an initial ellipsoid E_0 as $B_2^+(\mathbf{0}, R)$, the initial center of E_0 is $\mathbf{0}$;
2 $\mathbf{p}^* \leftarrow \mathbf{0}, u_{best} \leftarrow -\infty$;
3 $i \leftarrow 0$;
4 **while** $Vol(E_i) \geq V_\varepsilon$ **do**
5 $\mathbf{p}$ is the center of E_i;
6 $H, \mathbf{p}^*, u_{best} \leftarrow \text{FindCuttingPlane}(\mathbf{p}, \mathbf{p}^*, u_{best})$;
7 $E_{i+1} \leftarrow$ the minimum-volume ellipsoid containing $E_i \cap H$;
8 $i \leftarrow i+1$;
9 **return** $\mathbf{p}^*$;

Having presented the basic framework, we next design a series of different ReOPT algorithms.

ReOPT-EL: Our first ReOPT algorithm uses ellipsoid as the bounding box, and the center of the ellipsoid as the chosen center sent to the FINDCUTTINGPLANE algorithm, as shown in Algorithm 4. At the beginning, it initializes a large ellipsoid that contains the feasible region K (Line 1). Then it follows the basic framework to iteratively find a new cutting plane to shrink the bounding ellipsoid until the volume of the bounding ellipsoid becomes smaller than V_ϵ, the volume of a predefined error box (Line 4-8). Specifically, this is accomplished by constructing the smallest ellipsoid containing the intersection of the current bounding ellipsoid E_i and the cutting plane H in each iteration (Line 7). When the algorithm stops, it returns the vector $\mathbf{p}^*$ that gives the best utility value u_{best} as the optimal resource reservation vector (Line 9).

We present the following theorem on the optimality and efficiency of Algorithm 4.

Theorem 3: Algorithm 4 finds the optimal resource reservation vector $\mathbf{x}^*$ that maximizes $util(\mathbf{x})$ subject to K through an $O(n^2 \log \frac{R}{\varepsilon})$ calls on the reservation separation oracle (ReSEP).

Proof: The proof of this theorem can be derived by integrating the complexity of the classic ellipsoid method for feasibility problems [26] and Theorem 2. □

As stated in Theorem 3, the running time of ReOPT-EL increases nonlinearly as the number of flows in a request

Algorithm 5 ReOPT-RW: A Resource Optimization Oracle Using Polytope as Bounding Box and the Center of Samples Chosen by Random-Walk to Find the New Cutting Plane

1 given an initial polytope P_0 as $B_\infty^+(\mathbf{0}, R)$, pick N random points $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_N}$ in P_0;
2 $\mathbf{p}^* \leftarrow \mathbf{0}, u_{best} \leftarrow -\infty$;
3 $i \leftarrow 0$;
4 **while** $Vol(P_i) \geq V_\varepsilon$ **do**
5 $\quad \mathbf{p} \leftarrow \frac{1}{N}\sum_{i=1}^{N} \mathbf{v_i}$;
6 $\quad H, \mathbf{p}^*, u_{best} \leftarrow \text{FindCuttingPlane}(\mathbf{p}, \mathbf{p}^*, u_{best})$;
7 $\quad P_{i+1} \leftarrow P_i \cap H$;
8 $\quad$ randomly select a point $\mathbf{v}$ from $\{\mathbf{v_i} \in H, i = 1, 2, \ldots, N\}$ and random walk in P_{i+1} by hit-and-run method to generate another N random points $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_N}$;
9 $\quad i \leftarrow i + 1$;
10 **return** $\mathbf{p}^*$;

Algorithm 6 ReOPT-VC: A Resource Optimization Oracle Using Polytope as Bounding Box and Volumetric Center to Find the New Cutting Plane

1 given an initial polytope P_0 as $B_\infty^+(\mathbf{0}, R)$, the volumetric center of P_0 is $\mathbf{0}$;
2 $\mathbf{p}^* \leftarrow \mathbf{0}, u_{best} \leftarrow -\infty$;
3 $i \leftarrow 0$;
4 **while** $Vol(P_i) \geq V_\varepsilon$ **do**
5 $\quad \mathbf{p} \rightarrow$ the volumetric center of P_i computed using Newton method;
6 $\quad H, \mathbf{p}^*, u_{best} \leftarrow \text{FindCuttingPlane}(\mathbf{p}, \mathbf{p}^*, u_{best})$;
7 $\quad$ denote the H as $\{\mathbf{x}|\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq \beta\}$, choose β' such that Equations (16)(17) are satisfied;
8 $\quad P_{i+1} \leftarrow P_i \cap \{\mathbf{x}|\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq \beta'\}$;
9 $\quad i \leftarrow i + 1$;
10 **return** $\mathbf{p}^*$;

increases (*i.e.*, $O(n^2)$). As such, finding optimal resource reservation for a reservation request with a large number of flows using ReOPT-EL may be very slow. To improve the scalability of BoxOpt, we develop two other resource optimization oracle algorithms.

ReOPT-RW: Different from ReOPT-EL, the ReOPT-RW algorithm uses a polytope as the bounding box, and the center of a set of random points in the polytope as the chosen center sent to the FINDCUTTINGPLANE algorithm [35], as shown in Algorithm 5. It starts from a large polytope that contains the feasible region K (Line 1) and a set of random points in this polytope. Then it follows the basic framework in Algorithm 3 to iteratively find a new cutting plane to cut the polytope into a smaller one, until the volume of the bounding polytope becomes smaller than V_ϵ, the volume of a predefined error box (Line 4-9). In each iteration, the new polytope P_{i+1} is computed as the intersection of the current polytope P_i and the new cutting plane H (Line 7), and a set of points in P_{i+1} is chosen using hit-and-run method (Line 8) to compute the chosen center in the next iteration (Line 5). When the algorithm stops, it returns the vector $\mathbf{p}^*$ that gives the best utility value u_{best} as the optimal resource reservation vector (Line 9).

The ReOPT-RW algorithm finds the optimal resource reservation vector much faster than the ReOPT-EL algorithm. This is because comparing with an ellipsoid bounding box, a polytope bounding box approaches to the optimal solution faster. Formally, we present its complexity and optimality as follows.

Theorem 4: Algorithm 5 finds the optimal resource reservation vector $\mathbf{x}^*$ *that maximizes* $util(\mathbf{x})$ *subject to* K *through an* $O(n \log \frac{R}{\varepsilon})$ *calls on the reservation separation oracle (ReSEP).*

Proof: The proof of this theorem can be derived by integrating the complexity of the random walk method for feasibility problems [35] and Theorem 2. □

From Theorem 4, we observe that the running time of ReOPT-RW increases linearly as the number of flows in a request increases (*i.e.*, $O(n)$), which is much lower than the $O(n^2)$ complexity of the ReOPT-EL algorithm. As we will show in Section V using real-world topology and trace, the ReOPT-RW algorithm outperforms the ReOPT-EL algorithm by reducing the number of ReSEP calls by 64% on average, and by 95% in the best case.

In addition to ReOPT-RW, we also develop another resource optimization oracle algorithm in the following, which also uses polytopes as bounding boxes, but with a different mechanisms to find the cutting plane in each iteration.

ReOPT-VC: Similar to ReOPT-RW, the ReOPT-VC algorithm also uses a polytope as the bounding box. However, instead of the center of a set of random points in the bounding box, the ReOTP-VC algorithm chooses to send the *volumetric center* of the bounding box to the FINDCUTTINGPLANE algorithm to get the new cutting plane [36]. More specifically, given a bounded polytope $P = \{\mathbf{x}|\mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^n\}$, its logarithmic barrier is $-\sum_{i=1}^{m} \ln((\mathbf{a}_i^{\mathrm{T}}\mathbf{x} - b_i)^2)$. The Hessian of this barrier is computed as $\Phi(\mathbf{x}) = \sum_{i=1}^{m} \frac{\mathbf{a}_i\mathbf{a}_i^{\mathrm{T}}}{(\mathbf{a}_i^{\mathrm{T}}\mathbf{x}-b_i)^2}$ where $\mathbf{a}_i^{\mathrm{T}}$ denotes the ith row of $\mathbf{A}$. Denote $F(\mathbf{x}) = \frac{1}{2}\ln(\det(\Phi(\mathbf{x})))$. The volumetric center of P is defined as the point that minimizes $F(\mathbf{x})$ over P.

Algorithm 6 describes the ReOPT-VC algorithm. It follows the same basic framework as the ReOPT-EL and the ReOPT-RW algorithms, with two key improvements. First, in each iteration, the algorithm computes the volumetric center of a polytope P_i computed using Newton method (Line 5). Second, to avoid the subgradient of $F(\mathbf{x})$ at the $\mathbf{p}$ in the next iteration becomes infinite, it adjusts the cutting plane $\{\mathbf{x}|\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq \beta\}$ computed from the FINDCUTTINGPLANE algorithm to $\{\mathbf{x}|\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq \beta'\}$ such that

$$\mathbf{c}^{\mathrm{T}}\mathbf{p} \geq \beta', \tag{16}$$

and

$$\frac{\mathbf{c}^{\mathrm{T}}\Phi(\mathbf{p})^{-1}\mathbf{c}}{(\mathbf{c}^{\mathrm{T}}\mathbf{p} - \beta')^2} = \frac{\varepsilon}{2}. \tag{17}$$

Similar as the ReOPT-RW algorithm, the ReOPT-VC algorithm also finds the optimal resource reservation vector much faster than the ReOPT-EL algorithm. We present its complexity and optimality in the following theorem.

Theorem 5: Algorithm 6 finds the optimal resource reservation vector $\mathbf{x}^*$ *that maximizes* $util(\mathbf{x})$ *subject to* K *through an* $O(n \log \frac{R}{\varepsilon})$ *calls on the reservation separation oracle (ReSEP).*

Proof: The proof of this theorem can be derived by integrating the complexity of the volumetric center based method for feasibility problems [36] and Theorem 2. □

Though the ReOPT-VC algorithm has the same linear complexity as the ReOTP-RW algorithm, as we will show in the next section using real-world topology and trace, the ReOPT-VC algorithm outperforms the ReOPT-RW algorithm by reducing the number of ReSEP calls by 30% on average, and by 76% in the best case.

Putting Theorems 2, 3, 4 and 5 together, we get the following theorem on the optimality and efficiency of BoxOpt.

Theorem 6: When using the ReOPT-RW or ReOPT-VC algorithm, BoxOpt finds $\mathbf{x}^*$ *that maximizes* $util(\mathbf{x})$ *subject to* K *through* $O(n^2 \log R \log \frac{R}{\varepsilon})$ *calls on ReMEM.*

E. Discussion

Privacy-Preserving: We first discuss the privacy-preserving property of BoxOpt. During the operation of BoxOpt, the user's objective function is never exposed to the network. In addition, Theorem 6 states that given a user's objective function on network resource reservation and a network's bandwidth feasible region, BoxOpt learns the optimal resource reservation efficiently with a polynomial total number of membership oracle calls. With this polynomial result and the NP-hardness of finding the feasible region through a membership oracle [13], it is straightforward to see that the user cannot reconstruct the bandwidth feasible region of the network. Formally, we give the following theorem:

Theorem 7: Given a user's objective function and a network's bandwidth feasible region, BoxOpt learns the optimal resource reservation without exposing the private information of the network (i.e., the bandwidth feasible region) and the user (i.e., the resource optimization objective) to each other.

As indicated in the theorem, the privacy-preserving of BoxOpt for a given pair of the user's objective function and the network's bandwidth feasible region. When a user adjusts its objective functions during the interaction with the simple interface (*e.g.*, testing the resource availability of different flows), whether BoxOpt can maintain privacy-preserving under such a differential query scenario is an open question, and we plan to investigate it as future work.

Supporting Convex Feasible Region: The problem formulation of BoxOpt assumes typical cases in network resource reservation systems, where the resource availability is represented as a set of linear inequalities [17]–[19]. However, the algorithms presented in Section IV-C and IV-D also apply to the cases where the feasible region is a convex set instead of a polytope. As such, BoxOpt can be extended to other application scenarios, such as resource allocation in wireless networks [37] and electric car charging scheduling in smart grid [38].

Supporting Objective Functions From Both Network and User: The problem formulation of BoxOpt assumes representative network resource reservation systems that do not have an internal utility, but only provide a simple request interface for users to reserve resources [5]–[9]. When a network internally has a utility function it wants to maximize, the actual bandwidth feasible region NRR allows for the user may become smaller, potentially improving the efficiency of BoxOpt. We plan to systematically investigate the extension of BoxOpt to this scenario in future work.

Deploying BoxOpt in Large-Scale Networks: Deploying BoxOpt in a large network may present several potential challenges. We discuss them and provide preliminary solutions one by one. The first challenge is how BoxOpt can *quickly collect the network information* (*e.g.*, routing, bandwidth and policies) of a large network and abstract them as linear inequalities. This procedure requires communications between the NRR module of BoxOpt and all routers/controllers of network, which may cause long latency and hence is not scalable. One design we have proposed to address this issue is precomputation and projection [17], [19]. To use this design in BoxOpt, the NRR module periodically collects network information from routers and controllers to discover the abstraction of linear inequalities for a set of flows between every pair of source and destination routers. For example, in a network with only 10,000 pairs of source-destination routers, one precomputed linear inequality may be $x_1 + x_2 + \ldots, x_{10000} \leq 100$. When a user wants to reserve resources for a set of flows, the NRR module does not need to communicate with routers and controllers for the information. Instead, it locally performs a projection on the precomputed linear inequalities by only keeping the variables representing the flows in the user request, and uses the projected inequalities to interact with the ARO module for computing the optimal resource reservation for user. In the same example, if the user request only involves two pairs of source-destination routers, say x_1 and x_3. The NRR module can project the precomputed inequality to get $x_1 + x_3 \leq 100$. In our early evaluation [17], [19], we find that this design improves the abstraction discovery delay in response to user request by at least 2 times, and the periodic precomputation latency is within seconds in a collaborative network with 200 member networks.

The second challenge is how BoxOpt copes with *the large number of linear inequalities* to represent the network information of a large network, since this affects the efficiency of the membership oracle (ReMEM) in BoxOpt. One design we have proposed to address this issue is to compress the large set of linear inequalities into a smaller set representing the same feasible region [17]–[19]. In particular, we develop a polynomial-time redundant inequality removal algorithm that given a large set of linear inequalities, computes the minimal set of linear inequalities with the same feasible region. With this algorithm, the NRR module can compress the projected set of inequalities to a minimal set, so that the efficiency of the membership oracle can be significantly improved. We evaluate the performance of this algorithm using various real topologies in TopologyZoo [39], and we find that it can remove over 40% redundant linear inequalities in a large WAN network (*e.g.*, $\sim$70 routers), even when the number of flows in a request is large (*e.g.*, 100) [40].

The third challenge is how BoxOpt *efficiently finds optimal resource reservation for user requests when there are a large number of requests in the network*. Treating each request as an independent one and using BoxOpt to find the optimal resource reservation may be inefficient. To this end, we give a couple of initial designs that use machine learning techniques (*e.g.*, neural network, Bayesian optimization and reinforce-

ment learning) to improve the efficiency of finding optimal resource reservations of a new request by harvesting the similarities between different requests. Although traditional machine learning techniques are not suitable for finding the optimal resource reservation for a given objective-resources pair via the simple request interface (as discussed in Section I), they have been proved useful for finding similarities between samples and make accurate predictions [41]. First, BoxOpt can incorporate an optimal resource prediction (ORR) module on the network side of BoxOpt to extract features of past requests (*e.g.*, the source-destination pair, the users of requests, network state, and the times of requests) and their optimal solutions as a training dataset to train a deep neural network based classifier. As such, when a new request comes, BoxOpt can redirect the request to the ORR module, which uses the trained classifier to predict the optimal resource reservation, and returns the prediction to the user. The user can then decide whether to accept and use this reservation or use it as a starting point for the ARO module to search for a better reservation solution. Second, BoxOpt can introduce a reinforcement learning based search accelerator (SA) module on the user side. In particular, it models the search history of the user's previous requests (*e.g.*, the sequence of sampled requests before finding an optimal resource reservation) as different state spaces, takes an action to decide the next sample request so as to avoid repeating similar searches for the new request, and observe the efficiency change of finding the optimal resource reservation as reward to further adjust the action. A systematic investigation of applying these two machine-learning based modules to improve the efficiency of BoxOpt is left as future work.

To summarize, in the above we discussed a series of issues of related with deploying BoxOpt in large-scale networks (*i.e.*, network information collection latency, large amounts of network information and large number of user requests), and provided preliminary solutions to address them (*i.e.*, precomputation and projection, minimal, equivalent linear inequality compression and the use of machine learning techniques to accelerate the processing of new requests based on request history). We plan to investigate these designs in more detail and integrate them into BoxOpt in future work. Moreover, as we will show in the next section, our evaluation results also demonstrate the potentials of BoxOpt being applied in large-scale networks (*i.e.*, seconds to find optimal resource reservation vs. hours or days of large-scale dataset transfer).

V. Performance Evaluation

We implement a prototype of BoxOpt and evaluate its performance on an operational federation network supporting large-scale distributed science collaborations, and using real traffic traces from recent science experiments. We first describe our setup, followed by the detailed results.

A. Methodology

We implement three versions of BoxOpt, all of which use the same ReSEP oracle presented in Algorithm 1 in the ARO scout. The first version uses the ReOPT-EL algorithm in the ARO controller, denoted as BoxOpt-ReOPT-EL, while the other two use the ReOPT-RW and ReOPT-VC algorithms, respectively, and denoted as BoxOpt-ReOPT-RW and BoxOpt-ReOPT-VC.

We evaluate the performance of BoxOpt on the topology from LHC Open Network Environment (LHCONE), a global science network consisting of 62 institute and transit networks [42]. We randomly select a topology for each network from the Topology Zoo [39]. Specifically, the topology chosen for transit networks varies from 31 switches/routers with 33 links to 49 switches/routers with 85 links. And the topology chosen for institute networks (*e.g.*, campus science networks) ranges from 7 switches/routers with 6 links to 21 switches/routers with 44 links. We then assemble the connections and topologies from each network to form a unified large science network, which reflects the current practice in science data analytics federation network.

We evaluate the performance of BoxOpt using the real trace from the CMS experiment [43], a main source of data traffic in LHCONE. In particular, we focus on a 7-day trace from September 30, 2018, to October 6, 2018. This trace consists of a total of 2215 resource reservation requests. The number of flows in each request varies between 1 and 15. Because the CMS experiment is one of the largest ongoing distributed scientific experiments with complex, distributed analytics across tens of geographically distributed locations, we believe the trace is representative of complex data flow of modern distributed applications.

B. Results

In our experiments, we set R as the maximum of link bandwidth in the network topology (*i.e.*, 10 Gbps) and $r = 0.0001R$. We run extensive experiments by choosing different values of ϵ and ρ and different utility functions. In what follows, we present the results of one setting: maximizing total throughput when $\rho = 0.001$ and $\epsilon = \frac{1}{5}r$. Results of other settings are highly similar as this setting, hence are omitted due to page limit.

Correctness of BoxOpt: For each reservation request, we compare the optimal resource reservation computed by BoxOpt with the optimal solution to the problem $util(\mathbf{x})$ subject to K computed by a state-of-the-art optimization solver (*i.e.*, CPLEX [28]). We find that in all 2215 requests, all three versions of BoxOpt output the same optimal solution as the CPLEX solver does, *i.e.*, we verify that BoxOpt has a 100% correctness ratio.

Efficiency of BoxOpt: As illustrated in Figure 2 (Section III), two critical bottlenecks deciding the efficiency of BoxOpt are communication latency between the optimization oracle at the ARO controller invoking the separation oracle at the ARO scout, and the large number of invocations on the membership oracle at the NRR module. In our experiment, we find that even though the membership oracle is invoked for a large number of times, its total invocation latency is negligible when compared with the communication latency. As such, we use the total communication latency to find the optimal resource reservation for each request to represent the efficiency of BoxOpt, and study the invocations times of ReMEM in later experiments. Specifically, we assume the user is located at New York and the network is in Los Angeles. For each invocation of ReSEP at the ARO scout from the ARO controller, we assign it a round trip time (RTT) randomly chosen from the statistic RTT data collected in [44]. Then the communication latency to find the

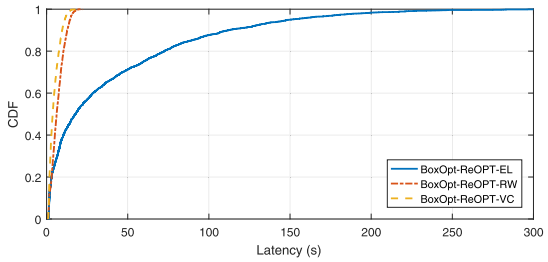


Fig. 4. CDF of the communication latency of BoxOpt.

optimal resource reservation for a given request is the sum of all RTTs incurred by corresponding ReSEP invocations.

Figure 4 plots the CDF of communication latency of all requests in the experiment. We observe that for around 90% of the requests, BoxOpt-ReOPT-EL is able to learn the optimal resource reservation within 120 seconds. This is due to the quadratic complexity of the ReOPT-EL algorithm.

In contrast, for 90% of the requests, BoxOpt-ReOPT-RW is able to learn the optimal resource reservation within 13 seconds, and its worst latency is less than 22 seconds. And BoxOpt-ReOPT-VC further has an efficiency of learning the optimal resource reservation within 10 seconds for 90% of the request and a worst latency of only 12 seconds. This demonstrates the efficiency of BoxOpt to swiftly learn the optimal resource reservation via the simple reservation interface. Considering the lasting time of network resource reservation (*e.g.*, hours and days) and the amount of data being transmitted (*e.g.*, TBs), we draw the conclusion that when using ReOPT-VC or ReOPT-RW, BoxOpt is highly efficient for finding the optimal resource reservation (90% 13 seconds and 90% 10 seconds, respectively). If applications relax the requirement of optimal resource reservation to accept sub-optimal reservations, the efficiency of BoxOpt can be further improved. We leave the investigation of this optimality-efficiency trade-off as future work.

Figure 5 plots the detailed statistics on the efficiency of different versions of BoxOpt to learn the optimal resource reservation for requests with different number of flows, with Figure 5d compares the average latency of different versions of BoxOpt. The increasing trend of the latency is consistent with our complexity analysis in Theorems 3, 4 and 5, *i.e.*, the latency increases quadratically for BoxOpt-ReOPT-EL, and linearly with BoxOpt-ReOPT-RW and BoxOpt-ReOPT-VC. We also note that in Figure 5d, we do not include the average latency of BoxOpt-ReOPT-EL for requests with more than 10 flows for the purpose of illustrating the performance difference between BoxOpt-ReOPT-RW and BoxOpt-ReOPT-VC: BoxOpt-ReOPT-VC has the lowest average latency among all three versions of BoxOpt.

Efficiency of ReOPT: We next study the efficiency of the ReOPT oracle to learn the optimal resource reservation. To this end, we count the number of ReSEP invocations (*i.e.*, the bottleneck operation of the ReOPT oracle) for each request.

Figure 6 shows the CDF of the number of ReSEP invocations in different versions of BoxOpt. We observe that for BoxOpt-ReOPT-EL, about 90% requests can be finished within 1200 ReSEP calls, but the remaining requests can take as many as 3000 ReSEP calls. This indicates that BoxOpt-ReOPT-EL is practical for request with a small number of flows.

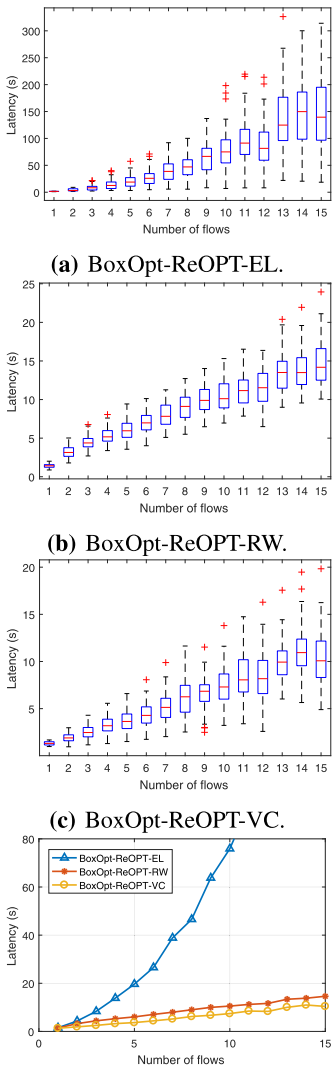


Fig. 5. Efficiency of BoxOpt: total communication latency to compute the optimal resource reservation.

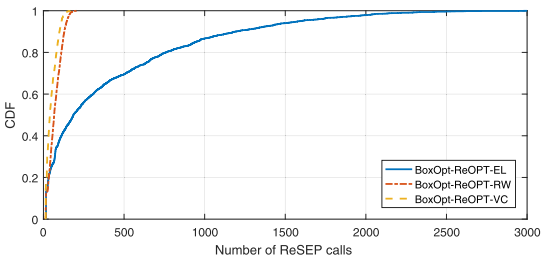
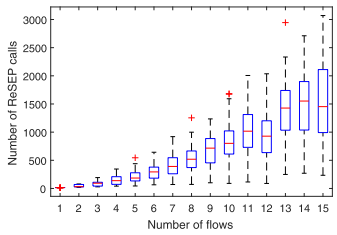
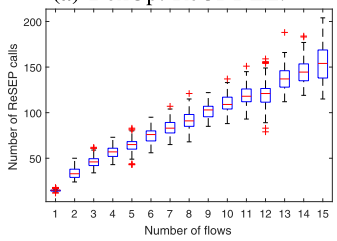


Fig. 6. CDF of the number of ReSEP invocations.

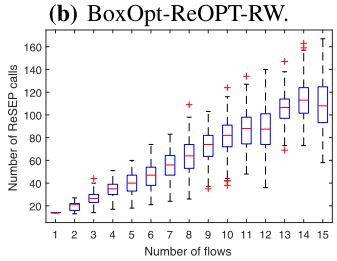
In contrast, for 90% requests, BoxOpt-ReOPT-RW can learn the optimal resource reservation within 130 calls on ReSEP, and BoxOpt-ReOPT-VC further reduces this number to only 100 ReSEP calls. This indicates that BoxOpt-ReOPT-RW and BoxOpt-ReOPT-VC is highly efficient and scalable even for finding the optimal resource reservation for requests with a large number of flows.



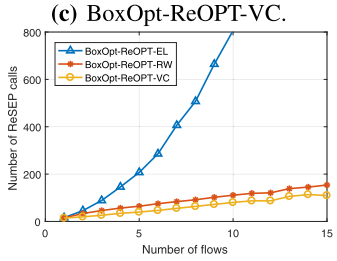
(a) BoxOpt-ReOPT-EL.



(b) BoxOpt-ReOPT-RW.



(c) BoxOpt-ReOPT-VC.



(d) Average number of ReSEP invocations.

Fig. 7. Efficiency of ReOPT: number of ReSEP invocations to learn the optimal resource reservation.

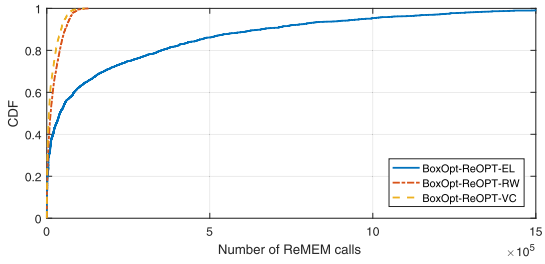
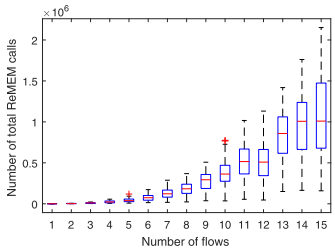
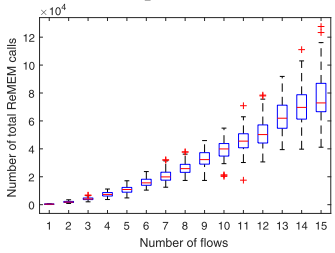


Fig. 8. CDF of the number of ReMEM invocations.

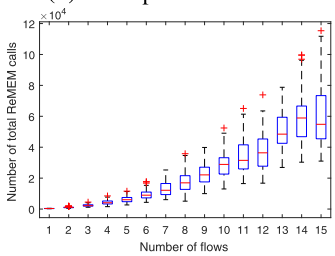
Figure 7a, Figure 7b and Figure 7c present the box plot of the number of ReSEP invocations in three versions of BoxOpt. To compare the difference, Figure 7d further plots the change of average number of ReSEP invocations of three systems over the number of flows in requests. We observe that the number of ReSEP invocations increases quadratically in BoxOpt-ReOPT-EL, and linearly in BoxOpt-ReOPT-RW and BoxOpt-ReOPT-VC, which is consistent with



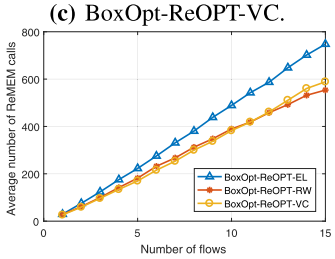
(a) BoxOpt-ReOPT-EL.



(b) BoxOpt-ReOPT-RW.



(c) BoxOpt-ReOPT-VC.



(d) Average number of ReMEM invocations in ReSEP request.

Fig. 9. Efficiency of ReSEP: number of ReMEM invocations to learn the optimal resource reservation.

the complexity analysis in Section IV-D and the statistics of Figure 5d. We also note that in Figure 7d, we do not include the average latency of BoxOpt-ReOPT-EL for requests with more than 10 flows for the purpose of illustrating the performance difference between BoxOpt-ReOPT-RW and BoxOpt-ReOPT-VC: BoxOpt-ReOPT-VC has the lowest average number of ReSEP calls among all three versions of BoxOpt.

In addition, we observe that although BoxOpt-ReOPT-VC provides the best overall average performance over BoxOpt-ReOPT-EL and BoxOpt-ReOPT-RW, for requests with the same number of flows, its variances of ReSEP calls is higher than that of BoxOpt-ReOPT-RW. This shows that BoxOpt-ReOPT-RW has a higher performance predictability, which is important for large data analytics networks.

Efficiency of ReSEP: In the end, we study the efficiency of the ReSEP oracle to infer the search space for ReOPT. To this end, we count the number of ReMEM invocations (*i.e.*, the

bottleneck operation of the ReSEP oracle) for each request. Figure 8 plots the CDF of the number of ReMEM invocations for all requests. We observe that for 90% requests, the total ReMEM invocations required is within 40000 with ReOPT-VC algorithm and within 50000 with ReOPT-RW algorithm, and even 550000 with ReOPT-EL algorithm. This large number demonstrates the necessity and benefits of putting ReSEP (*i.e.*, the ARO scout) with the network. Although it does not reduce the total number of ReMEM invocations, considering the applications' need to find the optimal resource reservation, the lasting time of network resource reservations (*e.g.*, hours and days) and the short time BoxOpt takes to find the optimal reservation (*e.g.*, seconds), the high invoking number of the NRR module is an acceptable overhead. In addition, given the long lasting time of reservations and the short time of BoxOpt to find the optimal reservation, the probability of many reservations competing to invoke the NRR module is low (*e.g.*, none in the production trace used in the evaluation).

Integrating this observation on the large number of ReMEM calls with the observation on the low latency and small number of ReSEP calls from Figure 6 and Figure 4, respectively, we can conclude that the design to put ReSEP on the network side improves the efficiency of BoxOpt (*i.e.*, the communication latency) by an average over 220 times, computed by dividing the total number of ReMEM calls by the total number of ReSEP calls of all experiments using ReOPT-VC algorithms.

Figure 9a, Figure 9b and Figure 9c further break down the statistics based on the size of requests and give the box plot of the number of ReMEM invocations with three versions of BoxOpt. Figure 9d gives the average number of ReMEM invocations in ReSEP request. We observe that the almost linear increase of ReMEM invocations is consistent with Theorem 2. This demonstrates that asymptotically the ReSEP algorithm (Algorithm 1) is highly scalable. And we leave how to further reduce the increasing slope of the number of ReMEM calls as future work.

VI. Conclusion

We investigate the feasibility and benefits for the user to learn the optimal network resource reservation for the user without exposing the private information of the network (*i.e.*, bandwidth capacity region) and the user (*e.g.*, resource orchestration policy) to each other. To this end, we design BoxOpt, a novel, automatic learning system to efficiently learn the optimal resource reservations through the simple reservation interface of network resource reservation systems, without exposing the private information of network or user. We demonstrate its efficiency and efficacy through extensive evaluation using real network topology and trace.

Acknowledgment

The authors thank Christian Bessiere, Haizhou Du, Kai Gao, Max Del Giudice, Chris Harshaw, Amin Karbasi, Yin Tat Lee, Geng Li, Yang Liu, John MacAuley, Harvey Newman, Lam M. Nguyen, Salvatore Ruggieri, Mudhakar Srivatsa, Xin Wang, Jingxuan Zhang, and Yan Zhu for their help during the preparation of this paper.

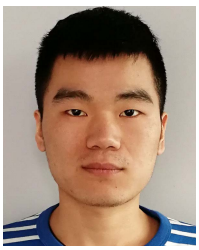
Qiao Xiang dedicates this paper to Wei Liang, a close friend from high school who recently passed away.

References

[1] M. Zaharia *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. NSDI*. Berkeley, CA, USA: USENIX Association, 2012, p. 2.
[2] T. White, *Hadoop: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly Media, 2012.
[3] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 5, pp. 44–48, Sep. 2012.
[4] M. Campanella *et al.*, "Bandwidth on demand services for European research and education networks," in *Proc. IEEE 1st Int. Workshop Bandwidth Demand*, Nov. 2006, pp. 65–72.
[5] C. Guok and D. Robertson, "Esnet on-demand secure circuits and advance reservation system (OSCARS)," in *Proc. GridNets*, vol. 92, 2006, [Online]. Available: https://www.es.net/assets/ESnet-Research/OSCARS/oscars-gridnets-20061001.pdf
[6] W. Johnston, C. Guok, and E. Chaniotakis, "Motivation, design, deployment and evolution of a guaranteed bandwidth network service," in *Proc. TERENA Netw. Conf.*, 2011, pp. 1–14.
[7] B. Riddle, "Bruw: A bandwidth reservation system to support end-user work," in *Proc. TERENA Netw. Conf.*, Poznan, Poland, 2005.
[8] X. Zheng, M. Veeraraghavan, N. S. V. Rao, Q. Wu, and M. Zhu, "CHEETAH: Circuit-switched high-speed end-to-end transport architecture testbed," *IEEE Commun. Mag.*, vol. 43, no. 8, pp. S11–S17, Aug. 2005.
[9] J. Sobieski, T. Lehman, and B. Jabbari, "Dragon: Dynamic resource allocation via gmpls optical networks," in *Proc. MCNC Opt. Control Planes Workshop*, Chicago, IL, USA, 2004.
[10] *The Large Hadron Collider*. Accessed: Dec. 18, 2020. [Online]. Available: https://home.cern/topics/large-hadron-collider
[11] *Oscars: On-Demand Secure Circuits and Advance Reservation System*. Accessed: Dec. 18, 2020. [Online]. Available: https://www.es.net/engineering-services/oscars/
[12] J. Lee *et al.*, "Application-driven bandwidth guarantees in datacenters," in *Proc. SIGCOMM*, 2014, pp. 467–478.
[13] C. Bessiere, F. Koriche, N. Lazaar, and B. O'Sullivan, "Constraint acquisition," *Artif. Intell.*, vol. 244, pp. 315–342, Mar. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370215001162
[14] R. Soulé *et al.*, "Merlin: A language for provisioning network resources," in *Proc. CoNEXT*, 2014, pp. 213–226.
[15] K. Subramanian, L. D'Antoni, and A. Akella, "Genesis: Synthesizing forwarding tables in multi-tenant networks," in *Proc. 44th ACM SIGPLAN Symp. Princ. Program. Lang. (POPL)*, 2017, pp. 572–585.
[16] V. Heorhiadi, M. K. Reiter, and V. Sekar, "Simplifying software-defined network optimization using sol," in *Proc. NSDI*, 2016, pp. 223–237.
[17] Q. Xiang *et al.*, "Toward fine-grained, privacy-preserving, efficient multi-domain network resource discovery," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 8, pp. 1924–1940, Aug. 2019.
[18] K. Gao, Q. Xiang, X. Wang, Y. R. Yang, and J. Bi, "An objective-driven on-demand network abstraction for adaptive applications," *IEEE/ACM Trans. Netw.*, vol. 27, no. 2, pp. 805–818, Apr. 2019.
[19] Q. Xiang *et al.*, "Fine-grained, multi-domain network resource abstraction as a fundamental primitive to enable high-performance, collaborative data sciences," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage, Anal. (SC)*, 2018, pp. 5:1–5:13.
[20] Q. Xiang *et al.*, "Toward optimal software-defined interdomain routing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Jul. 2020, pp. 1529–1538.
[21] M. A. Gelbart, J. Snoek, and R. P. Adams, "Bayesian optimization with unknown constraints," 2014, *arXiv:1403.5607*. [Online]. Available: http://arxiv.org/abs/1403.5607
[22] S. Ariafar, J. Coll-Font, D. Brooks, and J. Dy, "ADMMBO: Bayesian optimization with unknown constraints using ADMM," *J. Mach. Learn. Res.*, vol. 20, no. 123, pp. 1–26, 2019.
[23] R. S. Sutton *et al.*, *Introduction to Reinforcement Learning*, vol. 135. Cambridge, MA, USA: MIT Press, 1998.
[24] S. Miryoosefi, K. Brantley, H. Daume, III, M. Dudik, and R. E. Schapire, "Reinforcement learning with convex constraints," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 14070–14079.
[25] P. Englert and M. Toussaint, "Combined optimization and reinforcement learning for manipulation skills," in *Robotics: Science and Systems*. Los Altos Hills, CA, USA: The RSS Foundation, 2016.
[26] L. Lovasz, M. Grotschel, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, 2nd ed. New York, NY, USA: Springer, 1993.
[27] Y. T. Lee, A. Sidford, and S. S. Vempala, "Efficient convex optimization with membership oracles," 2017, *arXiv:1706.07357*. [Online]. Available: http://arxiv.org/abs/1706.07357

[28] (2018). *ILOG CPLEX*. [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer
[29] L. De Raedt, A. Passerini, and S. Teso, "Learning constraints from examples," in *Proc. 32nd AAAI Conf. Artif. Intell. (AAAI)*, New Orleans, LA, USA, 2018, pp. 2–7.
[30] S. Ruggieri, "Deciding membership in a class of polyhedra," in *Proc. ECAI*, 2012, pp. 702–707.
[31] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh, "Disjoint, partition and intersection constraints for set and multiset variables," in *Proc. Int. Conf. Princ. Pract. Constraint Program.* Berlin, Germany: Springer, 2004, pp. 138–152.
[32] S. Ruggieri, "Learning from polyhedral sets," in *Proc. 23rd Int. Joint Conf. Artif. Intell. (IJCAI)*, 2013, pp. 1069–1075.
[33] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[34] S. Bubeck and R. Eldan, "Multi-scale exploration of convex functions and bandit convex optimization," in *Proc. Conf. Learn. Theory*, 2016, pp. 583–589.
[35] D. Bertsimas and S. Vempala, "Solving convex programs by random walks," in *Proc. 34th Annu. ACM Symp. Theory Comput. (STOC)*, 2002, pp. 109–115.
[36] P. M. Vaidya, "A new algorithm for minimizing convex functions over convex sets," *Math. Program.*, vol. 73, no. 3, pp. 291–341, Jun. 1996.
[37] D. Julian, M. Chiang, D. O'Neill, and S. Boyd, "QoS and fairness constrained convex optimization of resource allocation for wireless cellular and ad hoc networks," in *Proc. 21st Annu. Joint Conf. IEEE Comput. Commun. Societies*, vol. 2, Jun. 2002, pp. 477–486.
[38] J. Rivera, C. Goebel, and H.-A. Jacobsen, "Distributed convex optimization for electric vehicle aggregators," *IEEE Trans. Smart Grid*, vol. 8, no. 4, pp. 1852–1863, Jul. 2017.
[39] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.
[40] K. Gao, C. Gu, Q. Xiang, X. Wang, Y. R. Yang, and J. Bi, "ORSAP: Abstracting routing state on demand," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Singapore, Nov. 2016, pp. 1–2.
[41] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[42] E. Martelli and S. Stancu, "LHCOPN and LHCONE: Status and future evolution," *J. Phys., Conf. Ser.*, vol. 664, no. 5, 2015, Art. no. 052025.
[43] *CMS Task Monitoring*. Accessed: Sep. 26, 2017. [Online]. Available: http://dashb-cms-job.cern.ch/
[44] (2018). *Global Ping Statistics—WonderNetwork*. [Online]. Available: https://wondernetwork.com/pings/.
[45] Q. Xiang, H. Yu, J. Aspnes, F. Le, L. Kong, and Y. R. Yang, "Optimizing in the dark: Learning an optimal solution through a simple request interface," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 1674–1681.

Qiao Xiang (Member, IEEE) received the bachelor's degree in information security and the bachelor's degree in economics from Nankai University in 2007, and the master's and Ph.D. degrees in computer science from Wayne State University in 2012 and 2014, respectively. He is a faculty member with Xiamen University. He was previously an Associate Research Scientist with the Department of Computer Science, Yale University. His research interests include software-defined networking, resource discovery and orchestration in collaborative data sciences, interdomain routing, and wireless cyber-physical systems. From 2016 to 2016, he was a Post-Doctoral Fellow with the Department of Computer Science, Yale University. From 2014 to 2015, he was a Post-Doctoral Fellow with the School of Computer Science, McGill University.



Haitao Yu received the bachelor's degree in computer science in 2017. He is currently pursuing the master's degree with the Department of Computer Science and Technology, Tongji University, China. His research interests include software defined networking, programming language, and optimization theory.

James Aspnes received the B.S. degree in mathematics, the M.S. degree in computer science and engineering from MIT, and the Ph.D. degree in computer science from Carnegie-Mellon University. He is currently a Professor of computer science with Yale University. His research emphasizes the use of randomization to solve difficult core problems in the theory of distributed algorithms. His recent work has concentrated on tools for managing large-scale loosely structured systems as found in peer-to-peer networks and wireless sensor networks. These tools include novel distributed data structures supporting efficient range queries over large data sets scattered across many machines, new models of distributed computation that capture the limited resources of individual nodes in sensor systems, and mechanisms for providing security and fault-tolerance in large-scale systems with no restrictions on the arrival of new and possibly malevolent participants. His interests also include related problems in biology, economics, and learning theory.

Franck Le received the Diplome d'Ingenieur degree from the Ecole Nationale Superieure des Telecommunications de Bretagne in 2000, and the Ph.D. degree from Carnegie Mellon University in 2010. He is currently a Research Staff Member with the IBM T. J. Watson Research Center. His current research interests lie at the intersection of the Internet of Things, artificial intelligence, and distributed systems and networks.

Chin Guok received the B.S. degree in computer science from the University of Pacific in 1991, and the M.S. degree in computer science from The University of Arizona in 1997. He joined ESnet in 1997 as a Network Engineer, focusing primarily on network statistics. He was a Core Engineer in the testing and production deployment of MPLS and QoS (Scavenger Service) within ESnet. He is the Technical Lead of the ESnet On-demand Secure Circuits and Advanced Reservation System (OSCARS) project which enables end-users to provision guaranteed bandwidth virtual circuits within ESnet. He also serves as a Co-Chair of the Open Grid Forum On-Demand Infrastructure Service Provisioning Working Group. His research interests include high-performance networking and network protocols; dynamic network resource provisioning; network tuning issues; and hybrid network traffic engineering.

Linghe Kong (Senior Member, IEEE) received the B.Eng. degree in automation from Xidian University in 2005, the master's degree in telecommunication from TELECOM SudParis in 2007, and the Ph.D. degree in computer science from Shanghai Jiao Tong University in 2012. He is currently a Research Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. Before that, he was a Post-Doctoral Researcher with Columbia University, McGill University, and Singapore University of Technology and Design. His research interests include wireless networks, big data, mobile computing, and the Internet of Things.

Y. Richard Yang (Senior Member, IEEE) received the B.E. degree in computer science and technology from Tsinghua University in 1993, and the M.S. and Ph.D. degrees in computer science from the University of Texas at Austin in 1998 and 2001, respectively. He is currently a Professor of computer science and electrical engineering with Yale University. His research is supported by both U.S. government funding agencies and leading industrial corporations, and spans areas including computer networks, mobile computing, wireless networking, and network security. His work has been implemented/adopted in products/systems of major companies (e.g., AT&T, Alcatel-Lucent, Cisco, Google, Microsoft, and Youku), and featured in mainstream media including Economist, Forbes, Guardian, Chronicle of Higher Education, Information Week, MIT Technology Review, Science Daily, USA Today, Washington Post, and Wired, among others. His awards include a CAREER Award from the National Science Foundation and a Google Faculty Research Award.