Multi-Resource Load Balancing for Virtual Network Functions

Tao Wang[1] Hong Xu[2] Fangming Liu*[1]
[1]Key Laboratory of Services Computing Technology and System, Ministry of Education,
School of Computer Science and Technology, Huazhong University of Science and Technology
[2]NetX Lab @ City University of Hong Kong

Abstract—**Middleboxes are widely deployed to perform various network functions to ensure security and improve performance. The recent trend of Network Function Virtualization (NFV) makes it easy for operators to deploy software implementations of these network functions on commodity servers. However, virtual network functions consume different amounts of resources when processing packets. Thus a multi-resource load balancing (MRLB) mechanism is needed to efficiently utilize server resources. MRLB problem in the context of NFV is fundamentally different from multi-resource allocation problems, as well as traditional single-resource load balancing and multi-resource load balancing problems in task scheduling. In this paper, we tackle the MRLB problem in NFV by first proposing *dominant load*—the load of the most stressed resource on a server—as the load balancing metric. We then formulate the MRLB problem as an optimization to minimize the maximum dominant load of all NFV servers given the demand. Based on proximal Jacobian ADMM, we propose an efficient algorithm to solve the problem in large scale settings. Through extensive trace-driven simulations and prototype experiments on a testbed, we show that our MRLB algorithm with dominant load performs significantly better and faster than benchmarking algorithms.**

I. Introduction

Middleboxes are prevalent in networks. A recent study [29] indicates that the number of middleboxes is comparable to that of forwarding devices. Middleboxes perform a wide range of critical network functions, e.g. intrusion detection systems (IDS), firewalls, VPN, WAN optimization, etc.

There is a growing trend of replacing dedicated hardware middleboxes with software instances of network functions on commodity servers, which is known as Network Function Virtualization (NFV). Just like other virtualization technologies, NFV enables elastic resource provisioning: Network functions can be deployed on a server cluster; an operator can dynamically increase the scale for a specific function when traffic hikes, or shut down idle virtual instances during demand valleys. Therefore, together with Software Defined Networking (SDN) [32], NFV holds great promises to reduce the network management cost and facilitate rapid deployment of new network functions [18].

For scalability, performance, and fault-tolerance, an operator usually deploys many virtual instances for the same network function (NF) across multiple servers [18]. Thus how to steer the traffic among these servers, that is load balancing, becomes a fundamental task for operating the NFV cluster. Load balancing is important because high utilization often leads to unstable packet processing performance and makes the NFV server prone to faults [20], [26]. As we show in Sec. II-A, some software implementations use 100% CPU with only $\sim$700Mbps incoming traffic, and suffer from more than 20% packet loss.

Traditionally, load balancing just distributes traffic evenly among the middleboxes running the same NF. It becomes more challenging in the context of NFV. Different NFs are now consolidated on the same physical server. Further, they entail different resource demands: IDS and security encryption bottleneck on CPU, while software routers bottleneck on memory or network I/O depending on the packet size [19]. We also verify the heterogeneity of resource consumptions through measurements in Sec. II-B. Simply balancing the load of one resource may lead to poor efficiency of other hardware resources, which limits the processing capacity of the entire NFV cluster. Thus the problem is inherently *multi-resource* and fundamentally different from traditional load balancing with a single resource (most commonly bandwidth) [10], [31].

An immediate challenge in multi-resource load balancing is, how shall we define the problem? Clearly the objective is to balance the utilization of the servers. However, with multiple resources now, the definition of a server's utilization, or load, is unclear. One may use the average load across all resources as a solution. This does not capture the actual resource limitation of the server. For example, for a server with two resources—CPU and bandwidth, consider two cases: (1) its CPU and bandwidth loads are 0.1 and 0.9, respectively; and (2) the loads are 0.5 and 0.5, respectively. The average load is the same, yet the server is clearly more stressed in the first case as it is running out of bandwidth.

Thus we propose to use a server's *dominant load*—the maximum load of all resource types—for multi-resource load balancing. The concept is similar to dominant resource share in multi-resource allocation problem [19]. In the above example, the server's dominant load is 0.9 in the first case and 0.5 in the second, indicating that it is more stressed with less capability in the first case. A desirable property of *dominant load* is that servers naturally prefer traffic whose demands help balance their load, which is instrumental to our problem. With this

*This work was supported in part by the National Key Research & Development (R&D) Plan under grant 2016YFB1000500, the National 973 Basic Research Program under Grant 2014CB347800, NSFC under Grant 61520106005, and by Hong Kong RGC GRF-11202315 and CRF-C7036-15G. (Corresponding author: Fangming Liu)

definition, the multi-resource load balancing problem is then to distribute traffic of different NFs among servers in order to minimize the maximum *dominant load*.

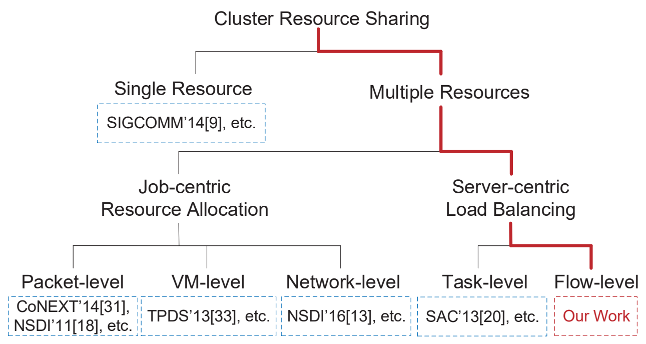


Fig. 1. Design space of cluster resource sharing.

Fig. 1 surveys the design space of cluster resource sharing and the thick line places our work in the context of multi-resource load balancing. Note that though related, our problem is different in nature from multi-resource allocation problem of various granularities (i.e. packet-level [19], [22], [33], VM-level [35] and flow-level [14]), which emerges from job scheduling in large-scale clusters. These need to assign workload (jobs or traffic of NFs) to servers. The allocation problem focuses on jobs, and it is to decide the amounts of resources for each job in a fair and efficient way. As long as the allocation is fair across jobs, which server ends up running them is irrelevant. On the contrary, load balancing focuses on servers, and strives to balance their loads given the total traffic volume. That is, the total workload is exogenous, mainly driven by applications, in load balancing, whereas in allocation it is the decision variable. Also, inherently different from the multi-resource load balancing problem in the context of task scheduling where the tasks are atomic and unsplittable [21], we assume that the aggregate flows in network can be arbitrarily split among servers, which is commonly used in the literature [36].

We make the following contributions towards solving the multi-resource load balancing problem in NFV.

- Based on *dominant load*, we present a simple yet general formulation of the problem in the offline setting as the first step. We consider a time period of minutes (e.g. 10 minutes), for which the total traffic demand of the NFV cluster can be predicted accurately as supported by a recent measurement of Facebook's internal data center network traffic [28].
- Given the increasing variety of NFs, the number of flows and their volume, and the number of servers involved, a fast and efficient solution is required to derive a load balancing scheme for a large-scale NFV deployment. Rather than using standard centralized solvers, we blend the advantages of the auxiliary variable method and proximal Jacobian Alternating Direction Method of Multipliers (ADMM) to devise an algorithm that addresses the scalability challenge.
- Our third contribution is an extensive performance evaluation using trace-driven simulations and testbed experiments. We show that our approach reduces the maximum *dominant load* of the NFV cluster by up to 62.7%. Our algorithm is able to converge within 34 iterations for large problem sizes, demonstrating its potential for practical use. We also develop a prototype of our algorithm using the Floodlight controller [3]. Experimental results show that our algorithm outperforms alternative schemes by up to 1.22×.

TABLE I
VIRTUAL NETWORK FUNCTIONS USED IN MEASUREMENT

VNF	Description
`Bro` [9]	A comprehensive traffic analysis platform.
`Snort` [7]	A network intrusion prevention system, capable of performing real-time traffic analysis.
`Squid` [8]	A web cache proxy, supporting HTTP, HTTPS, FTP, etc.
`PRADS` [6]	A passive packet sniffer to collect traffic statistics.

II. Motivation

In this section, we first motivate multi-resource load balancing using an empirical measurement study of common virtual network functions (VNFs). Our measurement involves four widely used software NFs in the literature [18]—`Bro` [9], `Snort` [7], `Squid` [8], and `PRADS` [6], as summarized in Table I. Then, we present examples that describes challenges of multi-resource load balancing in detail.

A. Why Load Balancing?

The first motivation question we must answer is, why bother considering load balancing? Can we just pack VNFs to achieve high utilizations of resources and reduce operating costs for operators? We show through measurements that high utilization severely degrades performance in the context of NFV, and thus it is necessary to consider load balancing.

We use a small testbed consisting of two commodity servers each with an Intel quad-core Xeon E5620 2.4GHz CPU and two 1GbE NICs. The measurements here use `Bro` and `Snort` only. We deploy them on one server as two Intrusion Detection Systems (IDS) that perform different packet filtering functions. Specifically, we configure `Bro` to filter every packet with a particular source IP address, and configure `Snort` to log any packet that matches a predefined list of source IP addresses. We generate traffic at different rates using another server to see how the VNFs perform. Each VNF is pinned to a unique core and measurements are carried out for 10 independent runs. Results are shown with error bars in Figs. 2 and 3.

We make the following observations. Fig. 2 shows that the CPU usage of `Bro` increases steadily with traffic. It reaches 100% when the traffic is over 700Mbps, which results in 20.9%–73.7% traffic loss as depicted in Fig. 3. For `Snort`, its CPU usage stabilizes at about 28% as shown in Fig. 2.

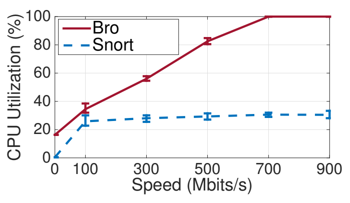


Fig. 2. CPU utilization comparison of Bro and Snort.

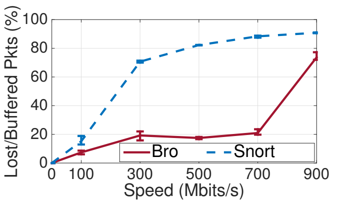


Fig. 3. Dropped/outstanding packets of Bro and Snort.

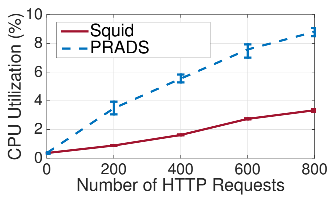


Fig. 4. CPU utilization comparison of Squid and PRADS.

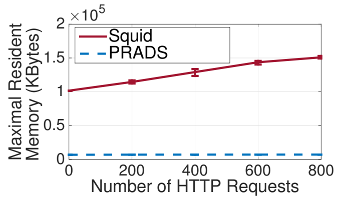


Fig. 5. Maximal resident memory of Squid and PRADS.

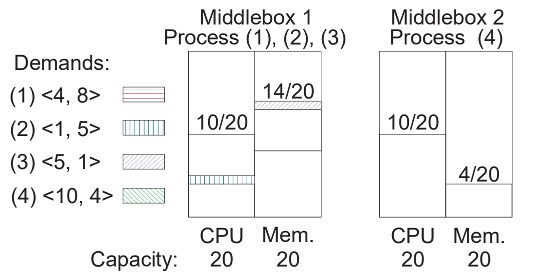


Fig. 6. A motivating example of multi-resource load balancing, where we only consider load balancing on CPU.

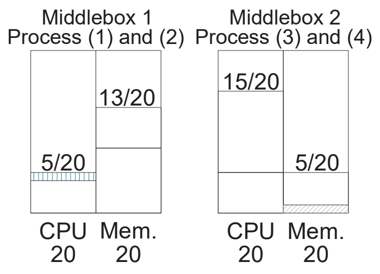


Fig. 7. Considering average load still leads to utilization imbalance within a VNF.

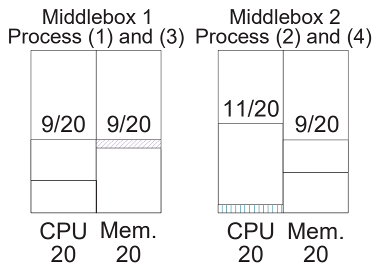


Fig. 8. Considering *dominant load* gives the best load balancing performance.

We believe this is due to its implementation that caps the CPU usage to avoid hogging the resource and the packet logging process indeed consumes less CPU compared with packet filtering which needs to search and match strings. As shown in Fig. 3, though Bro's percentage of buffered packets is stable when traffic ranges from 300Mbps to 700Mbps, buffered packets may be lost and result in long-tail flow completion time, inflating latency for real-time processing. Further, as traffic increases, the percentage of outstanding packets buffered for processing increases to over 70%.

The measurements show that, in data centers where traffic exhibits significant spatial and temporal variations [11], high utilization easily leads to overload and performance degradations. Our finding is consistent with some existing work that also reports unstable packet processing performance when server resource is heavily used [26]. It is important to load balance the VNFs and servers when operating the NFV cluster.

B. Why Multi-Resource Load Balancing?

Next we motivate the need of considering multiple resources for load balancing, again using empirical measurements. We focus on assessing the CPU and memory usages of Squid and PRADS now. We configure Squid as a web cache proxy to cache and reuse web pages for a Firefox browser. For PRADS, we configure it to passively listen on the NIC and record the TCP SYN packets received. We use HttpClient [5] to generate HTTP requests for different websites. For Squid, we direct all the HTTP requests through the Squid proxy, while for PRADS we directly send the HTTP requests to the external websites. We vary the number of HTTP requests to see their performances under different traffic loads. Consistent with previous section, each VNF is pinned to a unique core and measurements are carried out independently for 10 runs. The results are depicted with error bars in Figs. 4 and 5.

TABLE II
CONFIGURATIONS OF MACHINES IN ONE OF THE GOOGLE CLUSTERS [4], [27], THE CPUS AND MEMORY ARE NORMALIZED TO THE MAXIMUM MACHINE

CPUs	Memory	Number of Machines
0.5	0.5	11616
0.5	0.25	6152
0.5	0.75	1919
1.0	**1.0**	1411
0.25	0.25	224
0.5	0.124	74
0.5	0.03	5
0.5	0.96	5
1.0	0.5	3
0.5	0.06	2

We observe that the multi-resource usage of NFs exhibits salient heterogeneity. CPU and memory usages of both Squid and PRADS increase proportionally with the number of requests.[1] PRADS is more CPU-bound: it uses ∼9% CPU when the number of TCP requests is 800 while Squid's CPU usage is only ∼3.4%. Squid, on the other hand, is memory-bound: its memory consumption is an order of magnitude larger than PRADS. Given the fact that the NFV cluster hosts different types of NFs [18] which may consume significantly heterogeneous amounts of different resources, one must consider multiple resources for NFV load balancing.

[1]The memory usage of PRADS does increase linearly. As it uses much less memory than Squid, this trend is not seen clearly in Fig. 5.

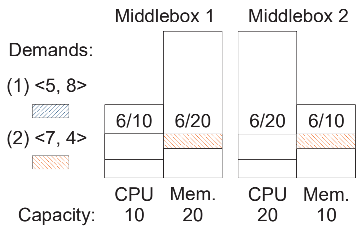


Fig. 9. Evenly splitting requests among two servers.

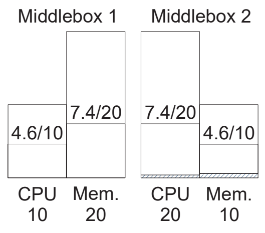


Fig. 10. Splitting requests considering dominant load.

C. Challenges of Multi-Resource Load Balancing

We now discuss the unique challenge of multi-resource load balancing, and why using the concept of *dominant load* can resolve it. Consider two identical middleboxes, each with two resources, CPU and memory, as shown in Fig. 6. The capacity is 20 units for each resource. There are 4 flows with equal bandwidth demand, and the per-packet processing demands of each flow are $\langle 4, 8\rangle$, $\langle 1, 5\rangle$, $\langle 5, 1\rangle$, and $\langle 10, 4\rangle$ as in $\langle \text{CPU}, \text{memory}\rangle$, respectively. For simplicity here the flows are unsplittable, while in general they are splittable, since in practice we consider a flow as an aggregate of individual flows between the same end-points.

We first show that traditional single resource load balancing does not apply to the multi-resource setting. Suppose load balancing only concerns CPU. This results in the solution shown in Fig. 6, where flows 1, 2, and 3 are routed to middlebox 1. The CPU utilization of both is the same. However, memory usage of middlebox 2 is much lower than middlebox 1. Only considering one resource clearly cannot balance the loads of multiple resources.

To consider multiple resources, one must scalarize the multi-dimensional utilization vector [22]. A natural idea is to use the average utilization of all resources, and equalize this metric across the middleboxes. Applying average load to the same example yields the solution depicted in Fig. 7, where flows 1 and 2 are directed to middlebox 1 and other two are directed to middlebox 2. This roughly equalizes the average load of both middleboxes. Yet for each middlebox, different resources are not utilized evenly. For middlebox 1, its CPU is well under-utilized while for middlebox 2, its memory is under-utilized. Thus average load cannot balance the loads of multiple resource well, either.

Finally, let us consider *dominant load*, i.e. the load of the most utilized resource on the middlebox, as the load balancing metric. This leads to flows 1 and 3 being directed to middlebox 1, as shown in Fig. 8. For middlebox 1 its CPU and memory are equally utilized, and for middlebox 2 its CPU load is just slightly higher than memory, significantly better than the two solutions before. Further, the dominant loads of the two (45% and 55%, respectively) are also balanced, and are much lower than the dominant loads in Fig. 6 and Fig. 7. Thus we rely on *dominant load* as the load balancing metric in this paper.

Further, in the splittable case, one may consider distributing the flows proportional to each server's processing capacity as an effective strategy. Note that the heterogeneity of servers exists in the cluster, as illustrated in Table II which lists the configurations of machines in one of Google clusters [4], [27]. We consider an example as shown in Figs. 9 and 10 where each middlebox has a total capacity of 30 units and those two have different capacities of $\langle 10, 20\rangle$ and $\langle 20, 10\rangle$ units for CPU and memory, respectively. Evenly splitting the requests results in 60% dominant load on both servers as shown in Fig. 9. However, the optimal solution is depicted in Fig. 10 with only 46% dominant load, which demonstrates that equally splitting requests is not desirable in the multi-resource context.

III. Multi-resource Load Balancing Problem

We now introduce our model and optimization framework for **M**ulti-**R**esource **L**oad **B**alancing (MRLB) in NFV.

A. Model

We consider a provider that deploys its NFs on a cluster. To overcome the limits of single-point failure and scalability, the provider deploys an NF across M commodity servers, and one server may host multiple NFs as shown in Fig. 11. An NF can have multiple instances of VNFs deployed on different commodity servers, performing the same processing for different flows. The cluster consists of P distinct NFs. Since the VNF placement problem has been studied [24], we assume that these M servers and P NFs are static. Each server j hosts a subset of VNFs denoted as $\mathcal{S}_j$ and has R different hardware resources.

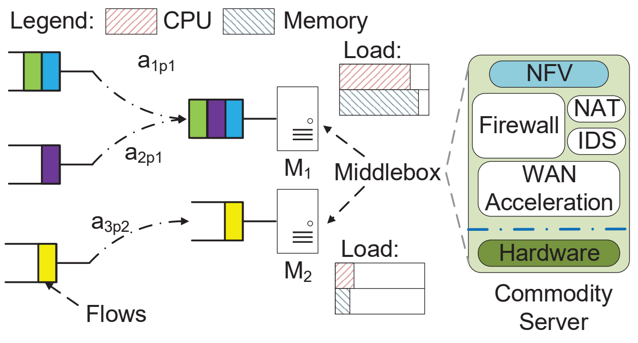


Fig. 11. An NFV cluster where flows are directed through a network to the virtual network functions (VNFs) deployed on the commodity servers.

Flows are directed to the NFV cluster to undergo packet processing which consumes multiple resources. The set of VNFs that a flow i needs to hit is $\mathcal{F}_i$ [17]. Flows have different per-packet processing demands. For the purpose of load balancing, we define a flow as an aggregate of traffic that has the same processing demands. The provider is able to collect statistics and enforce policies of network functions [29]. Thus, rather than arbitrarily changing the NFs processing online, which may violate the flow affinity requirement, we consider an offline setting where the traffic demand (in packet size or Gb) and $\mathcal{F}_i$ (set by the operator) of an aggregate flow are known, and the load balancing decision needs to be

TABLE III
KEY NOTATIONS

Symbol	Semantics
$i \in [1, N]$	flow i
$j \in [1, M]$	server j
$k \in [1, R]$	resource k
$\mathcal{F}_i$	set of NFs flow i needs to be processed by
F_{ip}	whether NF p is in $\mathcal{F}_F(i)$
$\mathcal{S}_j$	set of deployed NFs on server j
S_{jp}	whether NF p is in $\mathcal{F}_S(j)$
D_{ipk}	flow i's demand of p type NF on resource k
C_{jk}	server j's capacity of resource k
a_{ipj}	fraction of flow i's traffic routed to j for NF p to process
l_j	dominant load of server j

made periodically say every 10 minutes. Recent measurements of production data center traffic also confirm that traffic is quite stable in the time scale of minutes to days [28] and recent technique achieves accurate traffic prediction [30]. We assume that full-bisection bandwidth network is available in the cluster, and do not consider the bandwidth footprint. Since a flow is an aggregate of many TCP flows, we assume it can be arbitrarily split among the servers, which is commonly used in the literature [36].

B. Problem Formulation

We summarize some useful notations in Table III before formulating our problem. Flow i's resource demand vector for NF p, obtained by multiplying its bandwidth demand (in packets/s) by per-packet processing demands (in resource/packet) is denoted as $\langle D_{ip1}, D_{ip2}, \ldots, D_{ipR} \rangle$, and server j's capacity vector is $\langle C_{j1}, C_{j2}, \ldots, C_{jR} \rangle$. Let a_{ipj} denote the fraction of flow i's traffic routed to server j for NF p to process.

The *dominant load* of server j is defined as:

$$l_j = \max_k \left\{ \sum_{i=1}^{N} \sum_{p=1}^{P} \frac{a_{ipj} D_{ipk} \cdot F_{ip} S_{jp}}{C_{jk}} \right\}. \tag{1}$$

Now, we rigorously formulate the MRLB problem. Our objective is to balance the loads across all servers while keeping the utilization of each resource low:

$$\begin{aligned}
\min \quad & \max_j \ l_j && (2)\\
\text{s.t.} \quad & (1), \\
& \sum_{j=1}^{M} a_{ipj} = 1, \ \forall\, i, p : F_{ip} \neq 0, && (3)\\
& \sum_{i=1}^{N} \sum_{p=1}^{P} a_{ipj} D_{ipk} \cdot F_{ip} S_{jp} \le C_{jk}, \ \forall\, j, k, && (4)\\
& a_{ipj} \in [0, 1], \ \forall\, i, p, j, && (5)\\
\text{var.} \quad & \{a_{ipj}\}.
\end{aligned}$$

The objective function is intuitive: We have to minimize the maximum *dominant load* of M servers, which is similar to single resource load balancing that minimizes the maximum link congestion for example. Constraint (3) is the usual flow conservation constraint which ensures all the flows are processed by the necessary NFs. The inequality (4) constrains the usages of resources in all dimensions of a server by its capacities.

Problem (2) is a classical minimax optimization, and can be solved by the classical minimax solver. Though the classical solver has polynomial time complexity, the number of iterations it requires is directly related to the size of problem [15]. In our problem, the number of flows (with identical processing demands) and number of servers can be $O(10^2)$–$O(10^3)$, and the number of NFs can be $O(10)$–$O(10^2)$. Thus problem (2) is a large-scale problem for which the conventional solver incurs a large number of iterations. Yet the problem needs to be solved within minutes in practice. Motivated by this observation, in the following we develop an efficient distributed solution algorithm.

IV. ADMM-based Algorithm Design

In this section, we present a new distributed algorithm to solve large-scale MRLB problems efficiently. The algorithm is based on the alternating direction method of multipliers (ADMM), a simple yet effective method for solving large-scale convex optimization problems [12]. We give a detailed overview of ADMM is in our online appendix [1] and our method falls into the category of proximal Jacobian ADMM [16].

A. Transforming to the ADMM Form

Before we transform the MRLB problem to the ADMM form, we simplify the problem formulation in (2). If a flow i does not need NF p, the variables $\{a_{ipj} \mid \forall\, j\}$ and the constants $\{D_{ipk} \mid \forall\, k\}$ are immaterial to the problem. Thus we use $\hat{a}_{qj}$ and $\hat{D}_{qk}$ to substitute a_{iqj} and D_{ipk} where q stands for a unique pair of flow i and NF p that flow i must be processed by (i.e. $q \to i, p$). The row dimension of the newly introduced variables is Q. Then, problem (2) can be expressed as:

$$\begin{aligned}
\min \quad & \max_j \ l_j && (6)\\
\text{s.t.} \quad & l_j = \max_k \left\{ \sum_{q=1}^{Q} \frac{\hat{a}_{qj} \hat{D}_{qk}}{C_{jk}} \right\}, \ \forall\, j, && (7)\\
& \sum_{j=1}^{M} \hat{a}_{qj} = 1, \ \forall\, q, && (8)\\
& \sum_{q=1}^{Q} \hat{a}_{qj} \hat{D}_{qk} \le C_{jk}, \ \forall\, j, k, && (9)\\
& \hat{a}_{qj} \in [0, 1], \ \forall\, q, j, && (10)\\
& \hat{a}_{qj} = 0, \ \forall\, q, j : F^{S}_{jq \to p} = 0, && (11)\\
\text{var.} \quad & \{\hat{a}_{qj}\}.
\end{aligned}$$

where the new constraint (11) indicates that if the server j does not run NF p, we cannot route flows that need NF p processing to this server.

To handle the minimax optimization (6), we introduce an auxiliary variable L and reformulate the problem:

$$\begin{aligned} \min \quad & L && (12)\\ \text{s.t.} \quad & L \geq \sum_{q=1}^{Q} \frac{\hat{a}_{qj}\hat{D}_{qk}}{C_{jk}}, \ \forall\ j,k, && (13)\\ & (8),\ (9),\ (10),\ (11), \\ \text{var.} \quad & L,\ \{\hat{a}_{qj}\}. \end{aligned}$$

Is ADMM directly applicable? Our problem (12) cannot be direclty solved using ADMM. First, the constraints (13) are inequalities, rather than the equality constraints required by ADMM. Second, constraints (8) and (9) couple all variables together, while in ADMM problems constraints are separable for each set of variables. Moreover, this coupling happens on two orthogonal dimensions simultaneously: The per-flow demand conservation constraint (8) couples $\hat{a}$ across all servers while the per-resource capacity constraint (9) couples $\hat{a}$ across all flows.

The inequality constraints can be transformed to equality constraints by introducing slack variables. For the second challenge, the coupling prevents (12) from being solved distributively. To this end, we introduce another set of auxiliary variables $\gamma_{qj} = \hat{a}_{qj},\ \forall\ q,j$, and reformulate the problem as follows:

$$\begin{aligned} \min \quad & L && (14)\\ \text{s.t.} \quad & L - \sum_{q=1}^{Q} \frac{\hat{a}_{qj}\hat{D}_{qk}}{C_{jk}} - \delta_{jk} = 0,\ \forall\ j,k && (15)\\ & \sum_{j=1}^{M} \gamma_{qj} = 1,\ \forall\ q, && (16)\\ & \gamma_{qj} = \hat{a}_{qj},\ \forall\ q,j, && (17)\\ & \gamma_{qj} \in [0,1]\,,\ \forall\ q,j, && (18)\\ & \delta_{jk} \geq 0, \forall\ i,j, && (19)\\ & (9),\ (10),\ (11), \\ \text{var.} \quad & L,\ \{\hat{a}_{qj}\},\ \{\gamma_{qj}\},\ \{\delta_{jk}\}. \end{aligned}$$

Remark: Problem (14) is obviously equivalent to the original problem (2). In (16), γ_{qj} guarantees that the traffic demand of a flow is fully processed while $\hat{a}_{qj}$ ensures that all the servers are not overloaded in any of the resources.

The augmented Lagrangian $\mathcal{L}_\rho$ of (14) can be obtained as follows:

$$\begin{aligned} \mathcal{L}_\rho = L + \sum_{j=1}^{M}\sum_{k=1}^{R} \vartheta_{jk}(L - \sum_{q=1}^{Q} \frac{\hat{a}_{qj}\hat{D}_{qk}}{C_{jk}} - \delta_{jk}) \\ + \frac{\rho}{2}\sum_{j=1}^{M}\sum_{k=1}^{R}(L - \sum_{q=1}^{Q} \frac{\hat{a}_{qj}\hat{D}_{qk}}{C_{jk}} - \delta_{jk})^2 \\ + \sum_{q=1}^{Q}\sum_{j=1}^{M} \varphi_{qj}(\gamma_{qj} - \hat{a}_{qj}) + \sum_{q=1}^{Q}\sum_{j=1}^{M} \frac{\rho}{2}(\gamma_{qj} - \hat{a}_{qj})^2 \qquad (20) \end{aligned}$$

where ϑ_{jk}, φ_{qj} are dual variables for constraints (15) and (17), respectively.

B. A Distributed ADMM Algorithm

Basically, ADMM solves the dual problem iteratively: $x_u^{t+1} = \arg\min_{x_u} \mathcal{L}_\rho(x_1^t, \cdots, x_u, \cdots, x_m^t, \lambda^t)$ for each variable x_u's update in iteration t, where λ is the dual variable. However, applying this classical Gauss-Seidel update method [12] is not desirable, since it is only convergent for 2 blocks of variables while our transformed formulation (14) has 4 blocks of variables. By splitting variables, m-block ($m \geq 3$) ADMM can be converted to the 2-block form [34]. However, this method introduces too many auxiliary variables which results in high computation time. The sequential updates of Gauss-Seidel are also hard to implement in a distributed way as discussed in our online appendix [1].

In order to develop an efficient algorithm with provable convergence, we choose to rely on proximal Jacobian ADMM [16]. Proximal Jacobian ADMM introduces a proximal term $\frac{1}{2}\left\|x_u - x_u^t\right\|_{H_u}^2$ of variable x_u to the augmented Lagrangian $\mathcal{L}_\rho$, where H_u is a symmetric and positive semi-definite matrix and we let $\|x_u\|_{H_u}^2 := x_u^{\mathrm{T}} H_u x_u$. The proximal term $\frac{1}{2}\left\|x_u - x_u^t\right\|_{H_u}^2$ makes the subproblem strongly convex and more stable.

Now we present our algorithm design. First, we initialize the variables L, δ, $\hat{a}$, γ and the multipliers ϑ, φ to 0. We set the matrices $H_u = \omega \mathrm{I}, \forall u$ ($\omega > 0$) in the proximal term. In each iteration $t+1$, we perform five updates distributively.

1. L**-update**: The update of L can be obtained by solving the following subproblem[2]:

$$\begin{aligned} \min \quad & L + \frac{\rho}{2}\sum_{j=1}^{M}\sum_{k=1}^{R}(L - \sum_{q=1}^{Q} \frac{\hat{a}_{qj}^t\hat{D}_{qk}}{C_{jk}} - \delta_{jk}^t)^2 \\ & + \sum_{j=1}^{M}\sum_{k=1}^{R} \vartheta_{jk}^t L + \frac{\omega}{2}(L - L^t)^2 \qquad (21)\\ \text{s.t.} \quad & 0 \leq L \leq 1. \end{aligned}$$

This problem has only one variable L and can be quickly solved with standard solvers.

2. δ**-update**: Each server j solves the following subproblem for obtaining $\delta_j^{t+1} := (\delta_{j1}^{t+1}, \delta_{j2}^{t+1}, \cdots, \delta_{jR}^{t+1})$:

$$\min \quad \frac{\rho}{2}\sum_{k=1}^{R}\left(L^t - \sum_{q=1}^{Q} \frac{\hat{a}_{qj}^t\hat{D}_{qk}}{C_{jk}} - \delta_{jk}\right)^2$$

[2]This can be easily derived from $\mathcal{L}_\rho$ as in (20) by directly regarding irrelevant variables as constants.

$$+\sum_{k=1}^{R}\vartheta_{jk}^{t}\delta_{jk}+\frac{\omega}{2}\left(\delta_j-\delta_j^t\right)^2 \quad (22)$$
$$\text{s.t.} \quad (19).$$

This problem has only R variables and can be easily solved by any solvers for the quadratic program.

3. $\hat{a}$-update: Each server j solves the following subproblem for $\hat{a}_j^{t+1} := (\hat{a}_{1j}^{t+1}, \hat{a}_{2j}^{t+1}, \cdots, \hat{a}_{Qj}^{t+1})^{\mathrm{T}}$:

$$\begin{aligned}\min \quad & -\sum_{k=1}^{R}\vartheta_{jk}^{t}\sum_{q=1}^{Q}\frac{\hat{a}_{qj}\hat{D}_{qk}}{C_{jk}}+\frac{\omega}{2}(\hat{a}_j-\hat{a}_j^t)^2\\ & +\frac{\rho}{2}\sum_{k=1}^{R}\left(L^t-\sum_{q=1}^{Q}\frac{\hat{a}_{qj}\hat{D}_{qk}}{C_{jk}}-\delta_{jk}^t\right)^2\\ & +\sum_{q=1}^{Q}\left(-\varphi_{qj}^t\hat{a}_{qj}+\frac{\rho}{2}(\gamma_{qj}^t-\hat{a}_{qj})^2\right) \quad (23)\\ \text{s.t.} \quad & (10).\end{aligned}$$

This per-server subproblem, just like (22), is again a small-scale quadratic problem which can be solved efficiently.

4. γ-update: Each generated q of the corresponding flow i assigned to NF p solves the following subproblem for $\gamma_q^{t+1} := (\gamma_{q1}^{t+1}, \gamma_{q2}^{t+1}, \cdots, \gamma_{qM}^{t+1})$:

$$\begin{aligned}\min \quad & \sum_{j=1}^{M}\left(\varphi_{ij}^t\gamma_{qj}+\frac{\rho}{2}(\gamma_{qj}-\hat{a}_{qj}^t)^2\right)\\ & +\frac{\omega}{2}\left(\gamma_q-\gamma_q^t\right)^2 \quad (24)\\ \text{s.t.} \quad & (16),\ (18).\end{aligned}$$

The γ-update is performed for each flow assigned to the specific NF, whereas the $\hat{a}$-update is done for each server. This demonstrates the necessity to introduce auxiliary variables γ to break the coupling between variables.

5. Dual updates: Each server j updates ϑ for the constraint (15):

$$\vartheta_{jk}^{t+1}=\vartheta_{jk}^t+\tau\rho\left(L^{t+1}-\sum_{q=1}^{Q}\frac{\hat{a}_{qj}^{t+1}\hat{D}_{qk}}{C_{jk}}-\delta_{jk}^{t+1}\right) \quad (25)$$

while each flow i assigned to NF p updates φ for the constraint (17):

$$\varphi_{qj}^{t+1}=\varphi_{qj}^t+\tau\rho(\gamma_{qj}^{t+1}-\hat{a}_{qj}^{t+1}) \quad (26)$$

Remark: As discussed above, $\{\hat{a}_{qj}\}$ decides the fractions of flow i that needs NF p processing routed to j^{th} server. The use of auxiliary variables $\{\gamma_{qj}\}$ enables the separation of per-flow and per-server constraints. The L-update proceeds towards the lower maximal *dominant load*. The $\hat{a}$-update guarantees the distribution of all loads does not exceed the servers' capacities across all resources as in (9), while the γ-update ensures flow conservation constraints in (8).

The distributed nature of proximal Jacobian ADMM allows for a parallel implementation. In step 1, some machines can be tasked with performing L-, δ-, and $\hat{a}$-updates for each NFV server, and some for γ-update for each flow at the same time, which can be done in parallel. In step 2, the new results of step 1 are broadcast for the dual variable updates before the next iteration starts.

C. Optimality Analysis

We now analyze the optimality of our proposed ADMM-based algorithm. Lemma 1 shows that our algorithm solves the problem (2) optimally.

Lemma 1. *Our proposed algorithm based on proximal Jacobian ADMM converges to an optimal solution at the rate of $o(1/t)$ for (2).*

The proof can be constructed along the same line as the proof in [16], and is omitted here.

V. Evaluation

We conduct trace-driven simulations to extensively evaluate our algorithms for solving MRLB. According to [29], we set the number of servers M to 100, the number of aggregated flows N to 500, and the number of NFs P to 3.

A. Simulation Setup

Traces and Topology: Due to the lack of real-world data about the processing demands of different network functions, we use the Google cluster workload traces [27] which record the resource consumptions of different jobs in a Google data center to emulate the multi-resource demands of traffic. The traces contain job demands for three resources, CPU, memory, and local disk space, whose CDFs are depicted in Fig. 12. In our simulation, we just use the CPU and memory demands, which are more relevant to packet processing than disk I/O. Though not ideal, this is arguably the only feasible evaluation methodology and existing work [33] resorts to the same approach.

We consider two settings of server configuration: (1) Homogeneous setting. The capacity of each resource on every server is set to 1, which is the maximum normalized capacity in Fig. 12 and Table II. (2) Heterogeneous setting. The configuration of servers is generated according to the capacity distribution shown in Table II. The experiments are conducted in discrete time slots. In each slot, we generate the demand vectors for each flow according to the distributions in Fig. 12. The set of VNFs flow i needs to hit $\mathcal{F}_i$ and the set of VNFs hosted by server j, $\mathcal{S}_j$, are randomly chosen.

Scheme compared: We evaluate the following schemes.

- **MRLBA:** Our proposed ADMM-based multi-resource load balancing algorithm. The parameters settings are discussed below.
- **Backfill Balance (BB):** A commonly used greedy heuristic for task scheduling in the multi-resource setting [23]. Here we apply BB with dominant load, referred to as

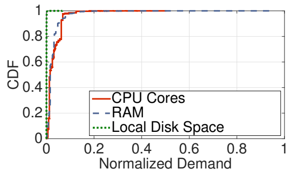


Fig. 12. The resource demands in Google's cluster [27], which are normalized to the maximal capacities.

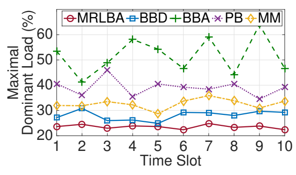


Fig. 13. The comparison of maximum dominant load of all resources on all servers in the homogeneous setting.

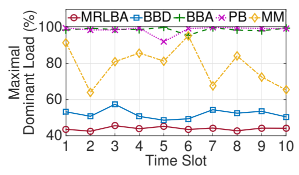


Fig. 14. The comparison of maximum dominant load of all resources on all servers in the heterogeneous setting.

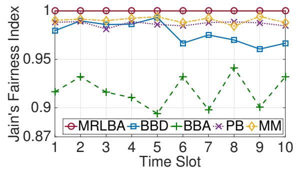


Fig. 15. The comparison of Jain's fairness index of dominant loads of all servers in the homogeneous setting.

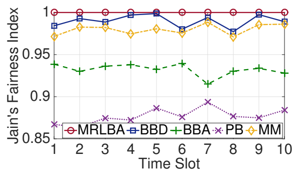


Fig. 16. The comparison of Jain's fairness index of dominant loads of all servers in the heterogeneous setting.

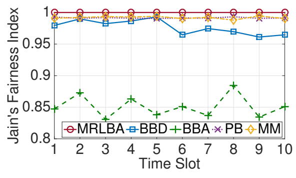


Fig. 17. The comparison of Jain's fairness index of CPU loads of all servers in the homogeneous setting.

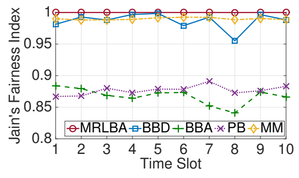


Fig. 18. The comparison of Jain's fairness index of CPU loads of all servers in the heterogeneous setting.

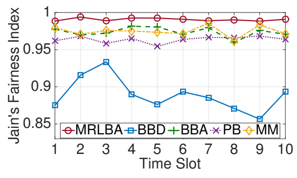


Fig. 19. The comparison of Jain's fairness index of memory loads of all servers in the homogeneous setting.

BBD, and also with average load as discussed in Sec. II, denoted as **BBA**.

- **Priority Based (PB):** Different weights are given to the individual resources based on their load distributions. The load of a server is defined as the weighted average of each resource's load [21]. PB aims at minimizing the standard deviation of all the servers' loads.
- **Market Mechanism (MM):** A market mechanism policy is introduced in [37] to balance multi-resource loads. In MM, job i selects the server j with lowest cost which is defined as $\frac{\sum_k^R J_{ik} \cdot L_{jk}}{\sum_k^R J_{ik}}$, where J_{ik} is task i's demand on resource k and L_{jk} is server j's load on resource k.
- `fminimax`**:** A function provided by MATLAB which internally implements the classical sequential quadratic programming (SQP) solver [2]. This is used for comparison of the convergence time.

ADMM Parameters: In ADMM algorithm, three parameters need to be tuned: the penalty parameter ρ, the damping parameter τ, and ω in the quadratic form in (21), (22), (23), (24), (25), and (26). According to the empirical study in [16], proximal Jacobian ADMM performs better with $\rho = 0.1$, $\tau = 1$, and $\omega = 0.1(m-1)\rho$ where m is the number of blocks (i.e. $m = 4$). We follow these guidelines here. We also adaptively update parameters according to [16]. The stopping criterion is that the relative error of the objective value between two consecutive iterations is less than 10^{-3}.

Performance metrics: The main performance metric is the maximum dominant load of servers, i.e. the objective of (2) in Sec. III. It shows the worst-case performance in the system. We also use the Jain's fairness index to evaluate the load balancing performance across servers. The Jain's fairness index,

$$\mathcal{J}(x_1, x_2, \cdots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \cdot \sum_{i=1}^n x_i^2},$$

is a value between $\frac{1}{n}$ and 1, where n is the number of elements. A value closer to 1 means the elements are more similar, i.e. all servers have roughly the same dominant load. We calculate this index by treating the dominant loads of each server as $\{x_i\}$. We also investigate Jain's index of individual resource usage.

B. Performance Evaluation

Load balancing among servers. Fig. 13 and 14 depict the maximum dominant load comparison of all schemes in both server configuration settings, which reflects the worst-case load among servers. We observe the following: (1) MRLBA offers the best performance with the lowest maximum load. It outperforms BBA, PB, and MM by 2.19×, 1.66×, and 1.38×, respectively on average in homogeneous setting, by 2.25×, 2.24×, and 1.79×, respectively on average in heterogeneous setting, which demonstrates its ability to balance load in multi-resource environments. (2) Compared with BBA, PB, and MM, the performance variation of MBLRA is smaller, indicating that MRLBA better adapts to traffic variations.

Further, Fig. 15 and 16 leverage the Jain's fairness index to show MRLBA's load balancing performance among the servers. On average, MRLBA is 1.04×, 1.18× and 1.04× better. This shows that MRLBA results in a more balanced load distribution among all the servers. As observed in Fig. 15, though PB and MM can sometimes outperform BBD considering Jain's fairness index, BBD indeed obtain a distribution with lower maximum dominant load in Fig. 13, especially in heterogeneous setting as shown in Fig. 14. The reason behind is that PB and MM are designed to achieve balanced load distribution while BBD is devised for minimizing the maximum dominant load, which is a more important metric as we have shown in the Sec. II. Moreover, with respect to the individual resource, Figs. 17, 18, 19 and 20 plot the Jain's fairness index of CPU and memory in both settings. It can be seen that MRLBA outperforms other algorithms.

Impact of traffic load. Next we investigate the performance of our algorithm in different traffic load conditions. We introduce a traffic load factor ϕ to scale the total resource demands of all flows relative to the baseline setting.

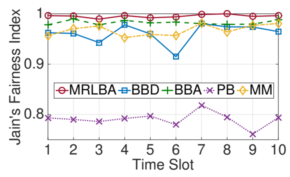


Fig. 20. The comparison of Jain's fairness index of memory loads of all servers in the heterogeneous setting.

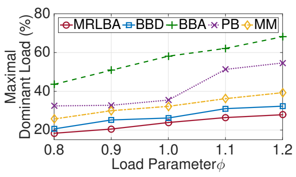


Fig. 21. The comparison of maximum dominant load with varying traffic load factors in homogeneous setting.

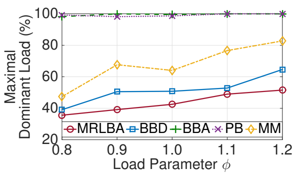


Fig. 22. The comparison of maximum dominant load with varying traffic load factors in heterogeneous setting.

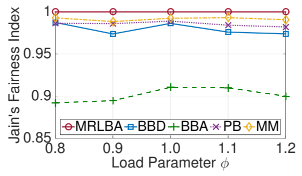


Fig. 23. The comparison of Jain's fairness index of dominant load with varying traffic load factors in homogeneous setting.

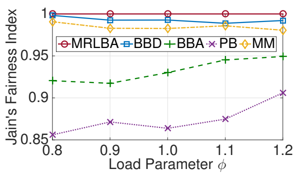


Fig. 24. The comparison of Jain's fairness index of dominant load with varying traffic load factors in heterogeneous setting.

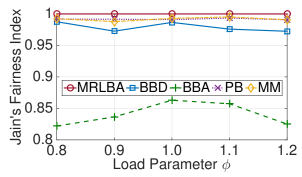


Fig. 25. The comparison of Jain's fairness index of CPU with varying traffic load factors in homogeneous setting.

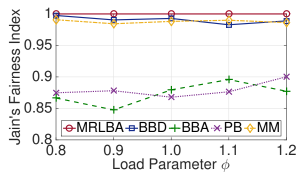


Fig. 26. The comparison of Jain's fairness index of CPU with varying traffic load factors in heterogeneous setting.

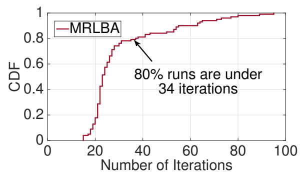


Fig. 27. Through over 300 runs, the MRLBA is able to converge within 34 iterations in most of the time.

The higher the ϕ is, the higher the resource contention is. We look at both the maximum dominant load and the Jain's fairness index. Fig. 21 and 22 show that the utilization of each resource on each server increases when the traffic load increases since each server needs to handle more traffic. Moreover, we can observe that the slope of the MRLBA curve is smaller compared with other schemes, which demonstrates the scalability of MRLBA to handle increasing traffic. Furthermore, in Fig. 23, 24, 25, 26, compared with BBA, PB, and MM, MRLBA results in a more balanced load distribution. Specifically, it brings 1.09×, 1.17× and 1.01× improvements in Jain's fairness index, respectively, as shown in Fig. 24.

Comparison of load balancing objectives. Recall that BBD uses dominant load as the criteria while BBA uses average load. From Fig. 13, 14, 15, 16, 21, 22, 23 and 24, it can be observed that in most runs, BBD outperforms BBA in both maximum dominant load and Jain's index considering dominant loads of all servers. This confirms that dominant load is a better load balancing objective than average load. However, as shown in Fig. 19, BBA results in a more balanced situation considering memory resource. This may be due to the demand distribution in Fig. 12 where the demand for CPU is higher compared with the demand for memory. Thus, BBD tries to lower the CPU usage which is relatively more stressed in the homogeneous setting. From Fig. 20, we can observe that this load imbalance problem of individual memory resource is mitigated. The reason behind is that the memory capacity of the NFV cluster in the heterogeneous setting is lower compared with it in the homogeneous setting, as shown in Table II. Therefore, in the heterogeneous setting, the memory may become the more stressed resource.

Convergence of our algorithm. At last, we experimentally examine the actual convergence speed of our MRLBA. Fig. 27 plots the CDF of the number of iterations that our algorithm needs to converge for over 100 runs with the same scale. Our algorithm is able to converge within 34 iterations for 80% of the runs. The fastest run only takes 15 iterations, and the slowest takes 96 iterations. Further, we examine the runtime of MRLBA as well as `fminimax`. The experiments are done on a server with Intel quad-core Xeon E5620 2.4GHz CPU and are carried out in 10 unique runs. The average runtime under different scales is shown in Table IV. Compared with `fminimax` solver, MRLBA converges in much less time, especially for large-scale problems. Note `fminimax` does not converge after one day for large-scale problems. This demonstrates that MRLBA is suitable for the time scale of load balancing in production networks.

TABLE IV
AVERAGE RUNTIME OF MRLBA AND `fminimax`

Runtime \ Scale / Method	MRLBA	`fminimax`
N=50, M=20	0.066s	5.996s
N=100, M=25	0.126s	57.501s
N=200, M=50	0.449s	8971.113s
N=500, M=100	5.594s	-
N=1000, M=200	31.544s	-

VI. IMPLEMENTATION

Besides simulations, we develop a prototype of our ADMM-based MRLB solver and conduct testbed experiments. As shown in Fig. 28, the prototype has four main components implemented in the controller: policy enforcer, MRLB solver, flow entries installer, and demand collector. The policy enforcer is used by the operator to specify which VNFs are needed for different flows. The demand collector is responsible for collecting and analyzing multi-resource usages for deciding the demand input to the MRLB solver. We use the `atop` tool to log the resource usages on each VNF server and send these logs to the controller. The MRLB solver takes input from the policy enforcer and demand collector to generate the load

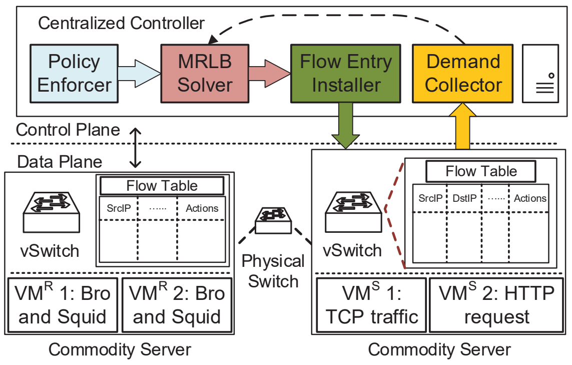
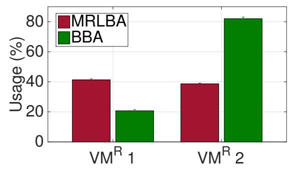
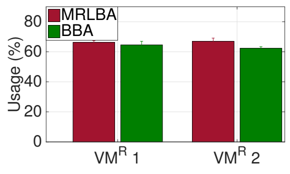


Fig. 28. The design of our MRLB solver in practice.



(a) CPU



(b) Resident memory

Fig. 29. Resource usages on the two receiver VMs.

balancing decisions based on Sec. IV, which is distributively implemented. Given the load balancing results, the flow entries installer enforces them by inserting QoS rules to host server vSwitch and physical switches.

We also conduct experiments on the testbed in a setting similar to that in Sec. II. On one server, we set 2 VMs each with 3 logical cores and 1GB memory as the traffic generator. One VM generates TCP traffic at 500Mbps which needs packet filtering by `Bro`, and the other VM sends 800 HTTP requests which are directed to the `Squid`. On the other machine, we set up another 2 VMs with the same configuration that host both `Bro` and `Squid`. The two VMs are pinned to a unique logical core. All VMs run OVS which connect to the Floodlight controller [3] on the host server.

Here we take the resource usages measured in Sec. II as the input to the MRLB solver. Fig. 29 plots the results. We observe that MRLBA achieves a more balanced load distribution between 2 VMs and outperforms BBA by 1.22× in terms of dominant load.

VII. Related Work

In this section, we briefly review existing work on multi-resource allocation and multi-resource load balancing problem in the context of task scheduling, as well as max-min fairness and ADMM.

Multi-resource allocation. Multi-resource fair allocation is first discussed in [19] in the context of cluster job allocation. Ghodsi et al. propose the notation of Dominant Resource Fairness (DRF) and allocation algorithms [19], and soon extend to considering multi-resource packet scheduling on middleboxes. The follow-up work (e.g. [25]) proposes methods for reducing the complexity of such a multi-resource scheduler. Wang et al. [33] explore the trade-off between fairness and throughput and propose a new DRF-based scheduler. As discussed in Sec. I, allocation problems (including VM-level [35] and flow-level [14] scheduling) are related to but different from load balancing: the former focuses on jobs and the fairness among them, while the latter focuses on servers given resources demands.

Multi-resource load balancing. Load balancing problem in the multi-resource setting has been studied in task scheduling, cache system and cloud system [21], [23], [37]. William et al. define average load of all resources as the server's load [23], while in [21], a server's load is defined as the weighted average of different resources where the weights are based on the runtime loads. And a market mechanism is proposed to schedule jobs in the multi-resource setting [37]. Those works are inherently a variant of multi-dimensional bin packing problem [13], where the task (job or request) is atomic and unsplittable. As disscused in Sec. I, our work distinguishes from multi-resource task scheduling: flow is splittable among servers and our objective is to minimize the dominant load rather than achieving balanced load distribution.

Max-min fairness. The MRLB problem formulation is closely related to max-min fairness, which has been extensively studied in the literature. To solve max-min problems, classical methods try to maximize the lowest allocation iteratively by leveraging LP solvers in each iteration, which has been proved inefficient and slow in [15]. Addressing the issue, Danna et al. use the idea of binary search to find saturated vertices and edges, which is shown experimentally to be faster [15]. In this paper, we resort to another technique, ADMM, to deal with the inefficiency.

ADMM. Firstly proposed in the 1970s, great advances in ADMM have been made in the field of statistics, machine learning and related areas [12]. In the field of networking, Xu et al. [36] derive the routing decisions for geo-distributed cloud services by using ADMM. Several recent works [16], [34], [38] extend the classical 2-Block ADMM [12] to the general m-Block ($m \geq 3$) case with convergence guarantees. In this paper, we introduce auxiliary variables based on [16], which falls into the category of proximal Jacobian ADMM.

VIII. Conclusion

In this paper, we presented the study on NFV load balancing in the multi-resource setting and motivated it through empirical measurements. Based on the notion of dominant load, we formally presented our multi-resource load balancing problem which considers both resource and server dimensions. Leveraging proximal Jacobian ADMM, we devised a distributed algorithm by introducing auxiliary variables to decompose the origin problem into a series of small-scale subproblems which can be solved efficiently. Extensive trace-driven simulations of both homogeneous and heterogeneous configuration settings show that our proposed algorithm outperforms greedy heuristics by ∼2.23× on average. Further, our proposed algorithm performs significantly faster than the benchmarking algorithm and it converges within 34 iterations for 80% of the time. Experimental results on testbed also demonstrated our algorithm's effectiveness.

References

[1] https://www.dropbox.com/s/mlvdtqvgwet0dxi/p-appendix.pdf?dl=0.

[2] http://www.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html#f26684.

[3] Floodlight Project. http://www.projectfloodlight.org/floodlight/.

[4] Google cluster workload trace. https://github.com/google/cluster-data.

[5] HttpComponents Client. http://hc.apache.org/httpcomponents-client-ga/index.html.

[6] PRADS. http://gamelinux.github.io/prads/.

[7] Snort Network Intrusion Prevention System. http://www.snort.org/.

[8] Squid: Optimising Web Delivery. http://www.squid-cache.org/.

[9] The Bro Network Security Monitor. http://www.bro.org/.

[10] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. In *Proc. of ACM SIGCOMM*, 2014.

[11] T. Benson, A. Akella, and D. A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. of ACM IMC*, 2010.

[12] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 2011.

[13] C. Chekuri and S. Khanna. On multidimensional packing problems. *SIAM Journal on Computing*, 2004.

[14] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica. HUG: Multi-Resource Fairness for Correlated and Elastic Demands. In *Proc. of USENIX NSDI*, 2016.

[15] E. Danna, S. Mandal, and A. Singh. A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering. In *Proc. of IEEE INFOCOM*, 2012.

[16] W. Deng, M.-J. Lai, Z. Peng, and W. Yin. Parallel multi-block ADMM with O (1/k) convergence. *arXiv preprint arXiv:1312.3040*, 2013.

[17] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *Proc. of USENIX NSDI*, 2014.

[18] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling Innovation in Network Function Control. In *Proc. of ACM SIGCOMM*, 2014.

[19] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proc. of USENIX NSDI*, 2011.

[20] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *Proc. of ACM SIGCOMM*, 2011.

[21] Y. Jia, I. Brondino, R. J. Peris, M. P. Martínez, and D. Ma. A multi-resource load balancing algorithm for cloud cache systems. In *Proc. of ACM SAC*, 2013.

[22] C. Joe-Wang et al. Multi-resource allocation: Fairness-efficiency trade-offs in a unifying framework. In *Proc. of IEEE INFOCOM*, 2012.

[23] W. Leinberger, G. Karypis, and V. Kumar. Job scheduling in the presence of multiple resource requirements. In *Proc. of ACM/IEEE SC*. ACM, 1999.

[24] L. Lewin-Eytan, J. Naor, R. Cohen, and D. Raz. Near Optimal Placement of Virtual Network Functions. In *Proc. of IEEE INFOCOM*, 2015.

[25] X. Li and C. Qian. Low-complexity multi-resource packet scheduling for network function virtualization. In *Proc. of IEEE INFOCOM*, 2015.

[26] R. Potharaju and N. Jain. Demystifying the Dark Side of the Middle: A Field Study of Middlebox Failures in Datacenters. In *Proc. of ACM IMC*, 2013.

[27] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proc. of ACM SoCC*, 2012.

[28] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the Social Network's (Datacenter) Network. In *Proc. of ACM SIGCOMM*, 2015.

[29] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service. In *Proc. of ACM SIGCOMM*, 2012.

[30] H. Wang, L. Chen, K. Chen, Z. Li, Y. Zhang, H. Guan, Z. Qi, D. Li, and Y. Geng. Flowprophet: Generic and accurate traffic prediction for data-parallel cluster computing. In *Proc. of IEEE ICDCS*, 2015.

[31] P. Wang, H. Xu, Z. Niu, D. Han, and Y. Xiong. Expeditus: Congestion-aware load balancing in clos data center networks. In *Proc. of ACM SoCC*, 2016.

[32] T. Wang, F. Liu, J. Guo, and H. Xu. Dynamic sdn controller assignment in data center networks: Stable matching with transfers. In *Proc. of IEEE INFOCOM*, 2016.

[33] W. Wang, C. Feng, B. Li, and B. Liang. On the fairness-efficiency tradeoff for packet processing with multiple resources. In *Proc. of ACM CoNEXT*, 2014.

[34] X. Wang, M. Hong, S. Ma, and Z.-Q. Luo. Solving multiple-block separable convex minimization problems using two-block alternating direction method of multipliers. *arXiv preprint arXiv:1308.5294*, 2013.

[35] Z. Xiao, W. Song, and Q. Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 2013.

[36] H. Xu and B. Li. Joint request mapping and response routing for geo-distributed cloud services. In *Proc. of IEEE INFOCOM*, 2013.

[37] C. C. Yang, K. T. Chen, C. Chen, and J. Y. Chen. Market-based load balancing for distributed heterogeneous multi-resource servers. In *Proc. of IEEE ICPADS*, 2009.

[38] Z. Zhou, F. Liu, B. Li, B. Li, H. Jin, R. Zou, and Z. Liu. Fuel cell generation in geo-distributed cloud services: A quantitative study. In *Proc. of IEEE ICDCS*, 2014.