An Alternating Direction Method Approach to Cloud Traffic Management

Chen Feng
Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada
cfeng@eecg.toronto.edu

Hong Xu
Department of Computer Science
City University of Hong Kong
Kowloon, Hong Kong
henry.xu@cityu.edu.hk

Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada
bli@eecg.toronto.edu

ABSTRACT

In this paper, we introduce a unified framework for studying various cloud traffic management problems, ranging from geographical load balancing to backbone traffic engineering. We abstract these real-world problems as a multi-facility resource allocation problem, and develop two distributed optimization algorithms that are amenable to parallel implementation. Our algorithms not only overcome the major difficulties of the standard dual-decomposition method, but also enjoy low computational complexity and low message-passing overhead. We prove the rate of convergence of our algorithms by utilizing several very recent results on alternating direction method of multipliers. As a by-product of our analysis, we provide a simple yet rigorous stopping rule. Simulation results not only confirm our analysis, but also highlight several additional advantages of our algorithms, such as scalability and fault-tolerance.

1. INTRODUCTION

In this paper, we consider problems of the following form:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{N} f_i(x_{i1},\ldots,x_{in}) - \sum_{j=1}^{n} g_j(y_j) \qquad (1) \\ \text{subject to} \quad & \forall j : \sum_{i=1}^{N} x_{ij} = y_j \\ & \forall i : x_i = (x_{i1},\ldots,x_{in})^T \in \mathcal{X}_i \subseteq \mathbb{R}^n \\ & \forall j : y_j \in \mathcal{Y}_j \subseteq \mathbb{R}. \end{aligned}$$

As we will describe in Sec. 2, optimization problems of this form appear in many cloud traffic management scenarios, including geographical load balancing and backbone traffic engineering. Generically, the problem (1) amounts to allocating resources from n facilities to N users such that the "social welfare" (i.e., utility minus cost) is maximized. The utility function $f_i(x_i)$ represents the performance, or the level of satisfaction, of user i when she receives an amount x_{ij} of resources from each facility j, where $x_i = (x_{i1},\ldots,x_{in})^T$. In practice, this performance measure can be in terms of revenue, throughput, or average latency, depending on the problem setup. We assume throughout the paper that $f_i(\cdot)$ are concave. The cost function $g_j(y_j)$ represents the operational expense or congestion cost when facility j allocates an amount y_j of resources to all the users. Note that y_j is the sum of x_{ij} (over i), since each facility only cares about the total amount of allocated resources. We assume that $g_j(\cdot)$ are convex. The constraint sets $\{\mathcal{X}_i\}$ and $\{\mathcal{Y}_j\}$ are used to model the additional constraints, which are assumed to be convex sets.

We refer to problem (1) as the *multi-facility resource allocation problem*. In this paper, we are interested in solutions that are amenable to *parallel* implementations, since a cloud provider usually has abundant servers for parallel computing. As we will soon see, (1) is inherently a large-scale convex optimization problem, with millions of variables, or even more, for a production cloud. The standard approach to constructing parallel algorithms is dual decomposition with (sub)gradient methods. However, it suffers from several difficulties for problem (1). First, dual decomposition usually requires delicate adjustments of step sizes, leading to slow convergence especially for large-scale problems. Second, dual decomposition generally requires the utility functions $f_i(\cdot)$ to be strictly concave *and* the cost functions $g_j(\cdot)$ to be strictly convex. However, these requirements cannot be met in many problem settings of (1), as demonstrated in Sec. 2.

To overcome these difficulties, we develop new distributed algorithms for the multi-facility resource allocation problem. Our solutions achieve faster convergence under weaker technical assumptions. In particular, our algorithms achieve $\mathcal{O}(1/k)$ rate of convergence for *general* utility and cost functions (where k is the number of iterations), and achieve $\mathcal{O}(1/a^k)$ (for some $a > 1$) rate of convergence when *either* the utility functions are strictly concave *or* the cost functions are strictly convex. More importantly, compared to dual decomposition, our solutions enjoy lower computational complexity and lower message-passing overhead.

Our distributed algorithms are based on *alternating direction method of multipliers* (ADMM), a simple yet powerful method that has recently found practical use in many large-scale convex optimization problems [7]. Although ADMM has been widely applied to areas of statistics, machine learning, and signal processing, its application to networking research is still in an early stage. To the best of our knowledge, the work [41–43] represents one of the first such applications. Compared to these previous algorithms, the algorithms proposed in this paper require much weaker technical assumptions to ensure convergence, and at the same time, enjoy much lower computational complexity and message-passing overhead.

We prove the rate of convergence of our algorithms by utilizing several very recent results on ADMM. Although the convergence of ADMM is well known in the literature (see, e.g., [5, 7]), its rate of

convergence has only been established very recently [12,21]. These new results provide a solid theoretical foundation for our analysis. Based on these results, we are not only able to establish the convergence rates of our algorithms, but also able to give a simple yet rigorous stopping rule compared to the conventional stopping rule proposed in [7].

Finally, we present an extensive empirical study on our algorithms. Our simulation results not only confirm our theoretical analysis, but also highlight some other important advantages of our algorithms, including their scalability to a large number of users and their fault-tolerance with respect to updating failures.

The main contributions of this paper are as follows:

1. We identify several cloud traffic management problems as instances of the multi-facility resource allocation problem.
2. We develop new distributed algorithms for the multi-facility resource allocation problem, which have a number of unique advantages compared to dual decomposition and previous algorithms.
3. We prove convergence rates for our algorithms by using several very recent results. We also provide a simple yet rigorous stopping rule for our algorithms.
4. We present extensive simulation results, which not only confirm our analysis, but also demonstrate the scalability and fault-tolerance of our algorithms.

2. APPLICATIONS TO CLOUD TRAFFIC MANAGEMENT

Before developing distributed algorithms to the multi-facility resource allocation problem, we first give a few examples from the recent literature in the context of cloud traffic management, where optimization problems of the form (1) naturally appear. We also illustrate the large scale of these problems for a production system, which motivates our quest for efficient distributed algorithms.

2.1 Geographical Load Balancing

2.1.1 Background

Cloud services, such as search, social networking, etc., are often deployed on a geographically distributed infrastructure, i.e. data centers located in different regions as shown in Fig. 1, for better performance and reliability. A natural question is then how to direct the workload from users among the set of geo-distributed data centers in order to achieve a desired trade-off between performance and cost, since the energy price exhibits a significant degree of geographical diversity as seminally pointed out by [37]. This question has attracted much attention recently [17, 29, 30, 37, 41–43], and is generally referred to as geographical load balancing.

2.1.2 Basic Model

We now introduce a formulation for the basic geographical load balancing problem, which captures the essential performance-cost trade-off and covers many existing works [17,30,37,41–43]. Here, we define a user to be an group of customers aggregated from a common geographical region sharing a unique IP prefix, as is often done in practice to reduce complexity [35]. We use x_{ij} to denote the amount of workload coming from user i and directed to data center j. We use t_i to denote the total workload of each user that can be fairly easily predicted using machine learning. We use $f_i(\cdot)$ to represent the utility of user i, and use $g_j(\cdot)$ to represent the cost of data center j. These functions can take various forms depending on the scenario as we will elaborate soon.

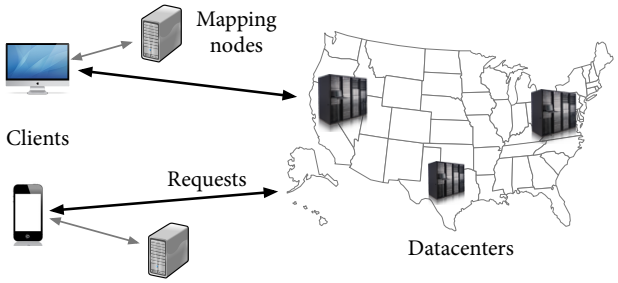

Figure 1: A cloud service running on geographically distributed data centers.

With these notations, we formulate the basic geographical load balancing problem:

$$\text{maximize} \quad \sum_i f_i(x_i) - \sum_j g_j\left(y_j\right) \tag{2}$$

$$\text{subject to} \quad \forall i: \sum_j x_{ij} = t_i, x_i \in \mathbb{R}_+^n, \tag{3}$$

$$\forall j: \; y_j = \sum_i x_{ij} \le c_j, \tag{4}$$

where (3) describes the workload conservation and non-negativity constraint, and (4) is the capacity constraint at data centers. Since the constraint (3) can be rewritten as $\forall i : x_i \in \mathcal{X}_i$, where $\mathcal{X}_i$ is a convex set, problem (2) is an instance of problem (1).

Now, let us consider the utility function $f_i(\cdot)$. Latency is arguably the most important performance metric for most interactive services: A small increase in the user-perceived latency can cause substantial utility loss for the users [27]. The user-perceived latency largely depends on the end-to-end propagation latency [16, 34], which can be obtained through active measurements. Let l_{ij} denote the end-to-end propagation latency between user i and data center j. The following utility function f_i has been used in [41,42]

$$f_i(x_i) = -qt_i \left(\sum_j x_{ij} l_{ij} / t_i \right)^2. \tag{5}$$

Here, q is the weight factor that captures the relative importance of performance compared to cost in monetary terms. Clearly, the utility function $f_i(\cdot)$ achieves its maximum value when latency is zero. Also, the function $f_i(\cdot)$ depends on the *average latency* $\sum_j x_{ij} l_{ij} / t_i$. For different applications, f_i may depend on other aggregate statistics of the latency, such as the maximum latency or the 99-th percentile latency, which may be modeled after a norm function.

For the cost function $g_j(\cdot)$, many existing works consider the following [17,30,37,43]

$$g_j(y_j) = P_j^E \cdot \text{PUE} \cdot E(y_j). \tag{6}$$

Here, P_j^E denotes the energy price in terms of \$/KWh at data center j. PUE, power usage effectiveness, is the ratio between total infrastructure power and server power. Since total infrastructure power mainly consists of server power and cooling power, PUE is commonly used as a measure of data center energy efficiency. Finally, $E(y_j)$ represents the server power at data center j, which is a function of the total workload y_j and can be obtained empirically. A commonly used server power function is from a measurement study of Google [14]:

$$E(y_j) = c_j P_{\text{idle}} + (P_{\text{peak}} - P_{\text{idle}})\, y_j, \tag{7}$$

where P_{idle} is server idle power and P_{peak} peak power.

2.1.3 Problem Scale

The geographical load balancing problem (2) would be easy to solve, if its scale is small with, say, hundreds of variables. However, for a production cloud, (2) is inherently an extremely large-scale optimization. In practice, the number of users N (unique IP prefixes) is on the order of $\mathcal{O}(10^5)$ [35]. Thus the number of variables $\{x_{ij}\}$ is $\mathcal{O}(10^6)$. The load balancing decision usually needs to be updated on a hourly basis, or even more frequently, as demand varies dynamically. The conventional dual decomposition approach suffers from many performance issues for solving such large-scale problems, as we argued in Sec. 1. Thus we are motivated to consider new distributed optimization algorithms.

2.1.4 Extensions

In this section, we provide some additional extensions of the basic model (2) from the literature to demonstrate its importance and generality.

Minimizing Carbon Footprint. In (2), the monetary cost of energy is modeled. The environmental cost of energy, i.e., the carbon footprint of energy can also be taken into account. Carbon footprint also has geographical diversity due to different sources of electricity generation in different locations [17]. Hence, it can be readily modeled by having an additional carbon cost P_j^C in terms of average carbon emission per KWh in the objective function of (2) following [17, 30].

Joint Optimization with Batch Workloads. There are also efforts [29,41,42] that consider the delay-tolerant batch workloads in addition to interactive requests, and the integrated workload management problem. Examples of batch workloads include MapReduce jobs, data mining tasks, etc. Batch workloads provides additional flexibility for geographical load balancing: Since their resource allocation is elastic, when the demand spikes we can allocate more capacity to run interactive workloads by reducing the resources for batch workloads.

To incorporate batch workloads, we introduce n "virtual" users, where user j generates batch workloads running on data center j. Let w_j be the amount of resource used for batch workloads on data center j, and let $\tilde{f}_j(w_j)$ be the utility of these batch workloads. Then the joint optimization can be formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_i f_i(x_i) + \sum_j \tilde{f}_j(w_j) - \sum_j g_j(y_j) \\ \text{subject to} \quad & \forall i: \sum_j x_{ij} = t_i,\ x_i \in \mathbb{R}_+^n; w \in \mathbb{R}_+^n \\ & \forall j: \ y_j = \sum_i x_{ij} + w_j \le c_j. \end{aligned}$$

The utility function $\tilde{f}_j(\cdot)$ depends only on w_j but not on latency, due to its elastic nature. In general, $\tilde{f}_j(\cdot)$ is an increasing and concave function, such as the log function used in [41, 42]. Clearly, this is still an instance of (1).

2.2 Backbone Traffic Engineering

2.2.1 Background

Large cloud service providers, such as Google and Microsoft, usually interconnect their geo-distributed data centers with a private backbone wide-area networks (WANs). Compared to ISP WANs, data center backbone WANs exhibit unique characteristics [18,25]. First, they are increasingly taking advantage of the software-defined networking (SDN) architecture, where a logically centralized controller has global knowledge and coordinates all transmissions [8, 19]. SDN paves the way for implementing logically centralized traffic engineering. In addition, the majority of the backbone traffic, such as copying user data to remote data centers and synchronizing large data sets across data centers, is elastic. Thus, since the cloud service provider controls both the applications at the edge and the routers in the network, in addition to routing, it can perform application rate control, i.e., allocate the aggregated sending rate of each application, according to the current network state. These characteristics open up the opportunity to perform joint rate control and traffic engineering in backbone WANs, which is starting to receive attention in the networking community [18, 23, 25].

2.2.2 Basic Model

We model the backbone WAN as a set $\mathcal{J}$ of interconnecting links. Conceptually, each cloud application generates a *flow* between a source-destination pair of data centers. We index the flows by i, and denote by $\mathcal{I}$ the set of all flows. We assume that each flow can use multiple paths from its source to destination. This is because multi-path routing is relatively easy to implement (e.g., using MPLS [13, 23, 25]) and offers many benefits. For each flow i, we denote by $\mathcal{P}_i$ the set of its available paths and define a *topology matrix* A_i of size $|\mathcal{J}| \times |\mathcal{P}_i|$ as follows:

$$A_i[j,p] = \begin{cases} 1, & \text{if link } j \text{ lies on path } p \\ 0, & \text{otherwise.} \end{cases}$$

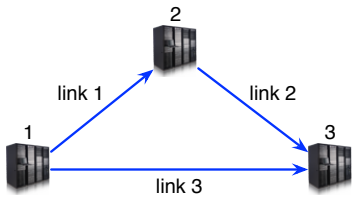

Figure 2: An illustration of three data centers with 3 links.

For example, consider a network with three data centers and 3 links as illustrated in Fig. 2. A flow (say, flow 1) from data center 1 to data center 3 has two paths: {link 1, link 2} and {link 3}. In this case, $|\mathcal{J}| = 3$, $|\mathcal{P}_1| = 2$, and the topology matrix A_1 is

$$A_1 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Clearly, the topology matrix A_i provides a mapping from paths to links. Let w_{ip} denote the amount of traffic of flow i on path p, and let x_{ij} denote the amount of traffic of flow i on link j. Then we have $x_i = A_i w_i$, where $w_i = (w_{i1}, \ldots, w_{i|\mathcal{P}_i|})^T$. Since A_i is always full column-rank (otherwise some path must be redundant), A_i has a left-inverse A_i^{-1} such that $w_i = A_i^{-1} x_i$. For instance, a left-inverse of A_1 in the previous example is

$$A_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Note that w_i models the rate control decision for each application flow. A flow corresponds to potentially many TCP connections between a particular source-destination pair of data centers, carrying traffic for this particular application. We choose to model rate control at the application flow level because the latest data center backbone architectures [23, 25] are designed to control the aggregated sending rates of applications across data centers. The aggre-

gated rate can be readily apportioned among different connections following some notion of fairness, and rate control can be enforced by adding a shim layer in the servers' operating system and using a per-destination token bucket [2].

We use $f_i(w_i)$ to represent the utility of flow i, and $g_j(y_j)$ to represent the congestion cost of link j, where $y_j = \sum_i x_{ij}$ is the total traffic on link j. The joint rate control and traffic engineering problem can be formulated as

$$\text{maximize} \quad \sum_i f_i(A_i^{-1}x_i) - \sum_j g_j(y_j) \tag{8}$$

$$\text{subject to} \quad \forall i: \ x_i \in \mathbb{R}_+^n, \tag{9}$$

$$\forall j: \ y_j = \sum_i x_{ij} \le c_j, \tag{10}$$

where (9) describes the non-negativity constraint, and (10) says that the total traffic on link j cannot exceed the capacity c_j. Clearly, problem (8) is again an instance of problem (1).

The utility function $f_i(w_i)$ should be concave, such as the log function $f_i(w_i) = \log(\sum_p w_{ip})$, or a more general "rate-fairness" function used for Internet TCP congestion control [33]. It is worth noting that even if $f_i(w_i)$ is strictly concave (with respect to w_i), $f_i(A_i^{-1}x_i)$ is *not* strictly concave (with respect to x_i) in general. This important fact will be used in Sec. 3.4. The cost function $g_j(y_j)$ is convex and non-decreasing. For example, the function can be a piece-wise linear function with increasing slopes, which is used in [18].

Finally, note that the topology matrix A_i only depends on the source-destination pair. Hence, for a given source data center, the number of all possible topology matrices is bounded by the number of all other data centers. In other words, the topology matrices are easy to store and maintain in practice.

2.2.3 Problem Scale

Similar to the geographical load balancing problem, backbone traffic engineering is also a large-scale optimization problem for a production data center backbone WAN. In practice, a provider runs hundreds to thousands of applications with around ten data centers [23, 25]. Thus the number of application flows is $\mathcal{O}(10^5)$ to $\mathcal{O}(10^6)$. For a WAN with tens of links, we potentially have tens of millions of variables $\{x_{ij}\}$. Compared to geographical load balancing, the traffic engineering decisions need to be updated over a very small time window (say, every 5 or 10 minutes as in [23,25]) to cope with traffic dynamics. This further motivates us to derive a fast distributed solution.

2.2.4 Extensions

We present some possible extensions of the basic model.

Minimizing Bandwidth Costs. Unlike big players like Google and Microsoft, small cloud providers often rely on ISPs to interconnect their data centers. In this case, bandwidth costs become one of the most important operating expenses. Although many ISPs adopt the 95-percentile charging scheme in reality, the link bandwidth cost is often assumed to be linear with the link traffic, because optimizing a linear cost in each interval can reduce the monthly 95-percentile bill [44]. Hence, the bandwidth cost can be easily incorporated by adding these linear functions to (8).

Incrementally Deployed SDN. Instead of upgrading all routers to be SDN-capable with a daunting bill, cloud providers could deploy SDN incrementally [1]. In such a scenario, some routers still use standard routing protocols such as OSPF, while other routers have the flexibility to choose the next hop. This scenario can be easily modeled by imposing additional constraints on the set $\mathcal{P}_i$ of available paths such that $\mathcal{P}_i$ only contains *admissible* paths. (See Definition 1 in [1] for details.)

3. DISTRIBUTED ALGORITHMS

In this section, we first review some basics of ADMM. We then apply ADMM to design our distributed algorithms.

3.1 Standard ADMM Algorithm

The standard ADMM algorithm solves convex optimization problems in the form

$$\begin{aligned} \text{minimize} \quad & f(x) + g(y) \\ \text{subject to} \quad & Ax + By = c, \\ & x \in \mathcal{X}, \ y \in \mathcal{Y}, \end{aligned} \tag{11}$$

with variables $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$, where $f: \mathbb{R}^n \to \mathbb{R}$ and $g: \mathbb{R}^m \to \mathbb{R}$ are convex functions, $A \in \mathbb{R}^{p\times n}$ and $B \in \mathbb{R}^{p\times m}$ are matrices, $\mathcal{X}$ and $\mathcal{Y}$ are nonempty closed convex subsets of $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively. The objective function in (11) is *separable* over *two* sets of variable x and y, which are coupled through a linear equality constraint $Ax + By = c$.

The *augmented Lagrangian* [22] for problem (11) is

$$L_\rho(x, y, \lambda) = f(x) + g(y) + \lambda^T(Ax + By - c) + (\rho/2)\|Ax + By - c\|_2^2,$$

where $\lambda \in \mathbb{R}^p$ is the Lagrange multiplier (or the dual variable) for the equality constraint, and $\rho > 0$ is the *penalty parameter*. Clearly, L_0 is the (standard) Lagrangian for (11), and L_ρ is the sum of L_0 and a *penalty term* $(\rho/2)\|Ax + By - c\|_2^2$. Introducing the penalty term leads to improved numerical stability and faster convergence [7].

The standard ADMM algorithm solves problem (11) with the iterations:

$$\begin{aligned} x^{k+1} &:= \operatorname*{argmin}_{x\in\mathcal{X}} L_\rho(x, y^k, \lambda^k), \\ y^{k+1} &:= \operatorname*{argmin}_{y\in\mathcal{Y}} L_\rho(x^{k+1}, y, \lambda^k), \\ \lambda^{k+1} &:= \lambda^k + \rho(Ax^{k+1} + By^{k+1} - c), \end{aligned}$$

where the penalty parameter ρ is the step size for the update of the dual variable λ. Note that the primal variables x and y are updated in an alternating fashion, which accounts for the term *alternating direction*.

The standard ADMM algorithm takes advantage of the separable structure of problem (11) and decomposes (11) over primal variables x and y. This is particularly useful in applications where the x-update and y-update admit simple solutions or can be implemented in a distributed manner.

The standard ADMM algorithm has a *scaled form*, which is often more convenient (and will be used in our algorithm design). Introducing $u = (1/\rho)\lambda$ and combining the linear and quadratic terms in the augmented Lagrangian, we can express the ADMM algorithm as

$$\begin{aligned} x^{k+1} &:= \operatorname*{argmin}_{x\in\mathcal{X}} \left(f(x) + (\rho/2)\|Ax + By^k - c + u^k\|_2^2\right), \\ y^{k+1} &:= \operatorname*{argmin}_{y\in\mathcal{Y}} \left(g(y) + (\rho/2)\|Ax^{k+1} + By - c + u^k\|_2^2\right), \\ u^{k+1} &:= u^k + Ax^{k+1} + By^{k+1} - c. \end{aligned}$$

The optimality and convergence of ADMM can be guaranteed under very mild technical assumptions [5,7]. In practice, it is often

the case that ADMM converges to modest accuracy within a few tens of iterations [7].

3.2 New Distributed Algorithms

Before we introduce our new distributed algorithms, we explain why a direct application of ADMM fails to provide a distributed solution.

For the simplicity of notations, we let $x = (x_1^T, \ldots, x_N^T)^T$, $f(x) = -\sum_{i=1}^N f_i(x_i)$, $y = (y_1, \ldots, y_n)^T$, and $g(y) = \sum_{j=1}^n g_j(y_j)$. Then problem (1) can be rewritten as:

$$\begin{array}{ll} \text{minimize} & f(x) + g(y) \\ \text{subject to} & Ax = y \\ & x \in \mathcal{X},\ y \in \mathcal{Y}, \end{array} \tag{12}$$

where the matrix $A = [I, \ldots, I]$ (I is the $n \times n$ identity matrix). Clearly, problem (12) is in ADMM form.

Observe that the penalty term is $(\rho/2)\| \sum_{i=1}^N x_i - y\|_2^2$, and so the x-update in the standard ADMM algorithm cannot be further decomposed across the users.

To address this difficulty, we introduce a set of auxiliary variables $z_i = x_i$, and reformulate problem (1) as:

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^N f_i(x_i) - g(\sum_{i=1}^N z_i) \\ \text{subject to} & \forall i : x_i = z_i \\ & \forall i : x_i \in \mathcal{X}_i;\ \sum_{i=1}^N z_i \in \mathcal{Y}. \end{array} \tag{13}$$

Now, the new penalty term is $(\rho/2)\sum_{i=1}^N \|x_i - z_i\|_2^2$, which is decomposable across the users.

Applying the scaled form of ADMM to problem (13), we obtain the following iterations:

$$x_i^{k+1} := \operatorname*{argmin}_{x_i \in \mathcal{X}_i} \left(-f_i(x_i) + (\rho/2)\|x_i - z_i^k + u_i^k\|_2^2 \right)$$

$$z^{k+1} := \operatorname*{argmin}_{(\sum_i z_i) \in \mathcal{Y}} \left(g(\sum_{i=1}^N z_i) + (\rho/2)\sum_{i=1}^N \|z_i - x_i^{k+1} - u_i^k\|_2^2 \right)$$

$$u_i^{k+1} := u_i^k + x_i^{k+1} - z_i^{k+1}.$$

Clearly, the first and last steps can be implemented independently in parallel for each user i. The second step (z-update) seems to require solving a problem in Nn variables, but we will soon see that it is equivalent to solving n single-variable problems.

Following the steps in [7], we rewrite the z-update as

$$\begin{array}{ll} \text{minimize} & g(N\bar{z}) + (\rho/2)\sum_{i=1}^N \|z_i - x_i^{k+1} - u_i^k\|_2^2 \\ \text{subject to} & \bar{z} = (1/N)\sum_i z_i, \\ & N\bar{z} \in \mathcal{Y}, \end{array}$$

where $\bar{z} \in \mathbb{R}^n$ is an additional variable denoting the average of $\{z_i\}$. Minimizing over $z_1, \ldots, z_N$ with $\bar{z}$ fixed has the solution

$$z_i = x_i^{k+1} + u_i^k + \bar{z} - \bar{x}^{k+1} - \bar{u}^k, \tag{14}$$

so the z-update can be implemented by solving the problem

$$\begin{array}{ll} \text{minimize} & g(N\bar{z}) + (N\rho/2)\|\bar{z} - \bar{x}^{k+1} - \bar{u}^k\|_2^2 \\ \text{subject to} & N\bar{z} \in \mathcal{Y}, \end{array} \tag{15}$$

for $\bar{z} \in \mathbb{R}^n$ and then applying (14). Substituting (14) for z_i^{k+1} in the u-update gives

$$u_i^{k+1} := \bar{u}^k + \bar{x}^{k+1} - \bar{z}^{k+1},$$

which shows that the dual variables u_i^k are equal for all the users. Substituting (14) for z_i^k in the x-update, we obtain the final version of the first algorithm.

Distributed ADMM Algorithm 1. Initialize $\{x_i^0\}$, $\{\bar{z}_j^0\}$, $\{\bar{u}_j^0\}$. For $k = 0, 1, \ldots,$ repeat

1. x**-update:** Each user i solves the following sub-problem for x_i^{k+1}:

$$\begin{array}{ll} \text{min} & -f_i(x_i) + (\rho/2)\|x_i - x_i^k + \bar{x}^k - \bar{z}^k + \bar{u}^k\|_2^2 \\ \text{s.t.} & x_i \in \mathcal{X}_i. \end{array}$$

2. z**-update:** Each facility j solves the following sub-problem for $\bar{z}_j^{k+1}$:

$$\begin{array}{ll} \text{min} & g_j(N\bar{z}_j) + (N\rho/2)(\bar{z}_j - \bar{x}_j^{k+1} - \bar{u}_j^k)^2 \\ \text{s.t.} & N\bar{z}_j \in \mathcal{Y}_j. \end{array}$$

3. **Dual update:** Each facility j updates $\bar{u}_j^{k+1}$:

$$\bar{u}_j^{k+1} := \bar{u}_j^k + \bar{x}_j^{k+1} - \bar{z}_j^{k+1}.$$

Next, we switch the order of the x-update and z-update in the scaled form of ADMM, obtaining our second algorithm:

$$z^{k+1} := \operatorname*{argmin}_{(\sum_i z_i) \in \mathcal{Y}} \left(g(\sum_{i=1}^N z_i) + (\rho/2)\sum_{i=1}^N \|z_i - x_i^k - u_i^k\|_2^2 \right)$$

$$x_i^{k+1} := \operatorname*{argmin}_{x_i \in \mathcal{X}_i} \left(-f_i(x_i) + (\rho/2)\|x_i - z_i^{k+1} + u_i^k\|_2^2 \right)$$

$$u_i^{k+1} := u_i^k + x_i^{k+1} - z_i^{k+1}.$$

Similarly, the second and last steps can be implemented in parallel, and the z-update can be handled in the same way as before. The final version of the second algorithm is as follows.

Distributed ADMM Algorithm 2. Initialize $\{x_i^0\}$, $\{\bar{z}_j^0\}$, $\{\bar{u}_j^0\}$. For $k = 0, 1, \ldots,$ repeat

1. z**-update:** Each facility j solves the following sub-problem for $\bar{z}_j^{k+1}$:

$$\begin{array}{ll} \text{min} & g_j(N\bar{z}_j) + (N\rho/2)(\bar{z}_j - \bar{x}_j^k - \bar{u}_j^k)^2 \\ \text{s.t.} & N\bar{z}_j \in \mathcal{Y}_j. \end{array}$$

2. x**-update:** Each user i solves the following sub-problem for x_i^{k+1}:

$$\begin{array}{ll} \text{min} & -f_i(x_i) + (\rho/2)\|x_i - x_i^k + \bar{x}^k - \bar{z}^{k+1} + \bar{u}^k\|_2^2 \\ \text{s.t.} & x_i \in \mathcal{X}_i. \end{array}$$

3. **Dual update:** Each facility j updates $\bar{u}_j^{k+1}$:

$$\bar{u}_j^{k+1} := \bar{u}_j^k + \bar{x}_j^{k+1} - \bar{z}_j^{k+1}.$$

Here, we would like to point out that these two algorithms have different strengths and naturally complement each other, which will be made clear in Section 4.

3.3 Parallel Implementation

The distributed nature of the above algorithms allows for efficient parallel implementation in the cloud that has abundant server resources. Here, we briefly discuss several issues pertaining to such implementations. We focus on the first algorithm, since the same discussion applies to the second algorithm.

We associate each user a type-1 processor, which stores and maintains two states $(x_i^k, \bar{x}^k - \bar{z}^k + \bar{u}^k)$. Similarly, we associate each facility a type-2 processor, which stores and maintains $(\bar{u}_j^k, \bar{x}_j^{k+1})$. At the kth iteration, each type-1 processor solves a small convex problem (in n variables), and then reports the updated x_{ij}^{k+1} to facility j. Each facility j collects x_{ij}^{k+1} from all type-1 processors, and then computes the average $\bar{x}_j^{k+1}$. This is called a *reduce* step in parallel computing [11]. After the reduce step, each type-2 processor solves a single-variable convex problem and updates $\bar{u}_j^{k+1}$. Then, each type-2 processor sends the value of $\bar{x}^{k+1} - \bar{z}^{k+1} + \bar{u}^{k+1}$ to all type-1 processors, which is called a *broadcast* step. In actual implementation, a server can host multiple processors of the same type. For example, one can use one server to host all type-2 processors and a number of other servers each hosting multiple type-1 processors. This enables us to further reduce the message-passing overhead, since a server hosting type-1 processors can report the *local sum* to the server hosting type-2 processors, and the server hosting type-2 processors needs only to send one copy of $\bar{x}^{k+1} - \bar{z}^{k+1} + \bar{u}^{k+1}$ to every other servers.

An alternative and perhaps much simpler method to implement Algorithm 1 is based on the MPI *Allreduce* operation [38], which computes the global sum over all processors and distributes the result to every processor. Although the Allreduce operation can be achieved by a reduce step followed by a broadcast step, an efficient implementation (for example, via butterfly mixing) often leads to much better performance. With the help of Allreduce, we only need N processors of the same type, with each storing and maintaining three states $(x_i^k, \bar{u}^k, \bar{x}^k)$. At the kth iteration, each processor solves a small convex problem and updates x_i^{k+1}. Then, all the processors perform an Allreduce operation so that all of them (redundantly) obtain $\bar{x}^{k+1}$. After this Allreduce step, each processor solves n single-variable convex problems and (redundantly) computes $\bar{u}^{k+1}$. This method simplifies our implementation and often helps to increase the speed.

For practical implementation, our distributed algorithms can be terminated even before convergence is achieved. This is a feature of ADMM, as it usually finds a reasonably good solution within just tens of iterations [7]. So, an early-braking mechanism can be safely incorporated into our algorithms, making them appealing for a wide range of applications.

3.4 Comparisons with Other Algorithms

Here, we compare our distributed algorithms with other possible algorithms. We begin with the dual-decomposition algorithm for problem (1).

Dual Decomposition Algorithm. Initialize $\{x_i^0\}$, $\{y_j^0\}$, $\{\lambda_j^0\}$. For $k = 0, 1, \ldots,$ repeat

1. x**-update:** Each user i solves the following sub-problem for x_i^{k+1}:

$$\begin{array}{ll} \text{min} & -f_i(x_i) + (\lambda^k)^T x_i \\ \text{s.t.} & x_i \in \mathcal{X}_i. \end{array}$$

2. y**-update:** Each facility j solves the following sub-problem for y_j^{k+1}:

$$\begin{array}{ll} \text{min} & g_j(y_j) - \lambda_j^k y_j \\ \text{s.t.} & y_j \in \mathcal{Y}_j. \end{array}$$

3. **Dual update:** Each facility j updates λ_j^{k+1}:

$$\lambda_j^{k+1} := \lambda_j^k + \rho^k \left(\sum_{i=1}^N x_{ij}^{k+1} - y_j^{k+1} \right),$$

where ρ^k is the step-size for the kth iteration.

At every iteration, each user in the dual decomposition solves an n-variable convex optimization, and each facility solves a single-variable optimization. Hence, the computational cost of dual decomposition is essentially the same as that of our distributed algorithms for each iteration. The x-update requires each user to know the value of λ^k, which can be achieved through a broadcast step. The dual-update requires each facility to know the sum $\sum_{i=1}^N x_{ij}^{k+1}$, which can be achieved through a reduce step. Hence, the message-passing overhead of dual decomposition is the same as that of our algorithms at each iteration. Since our algorithms achieve faster convergence, they enjoy lower overall computational complexity and lower message-passing overhead.

On the other hand, dual decomposition usually requires delicate adjustments of step sizes ρ^k, resulting in slow convergence; dual decomposition generally requires the cost functions $g_j(\cdot)$ to be strictly convex and the utility functions $f_i(\cdot)$ to be strictly concave. In contrast, our distributed algorithms do not suffer from these two difficulties. As we will show in Sec. 5.4, for solving the geographical load balancing problem (2), dual decomposition does not converge after hundreds of iterations, while our algorithms converge after 50 iterations.

There are some other ADMM-type distributed algorithms in the literature, such as linearized ADMM [21] and multi-block ADMM [20, 24]. However, they are not particularly suitable for the multi-facility resource allocation problem (1). For example, applying linearized ADMM to problem (1) gives the following iterations:

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i \in \mathcal{X}_i} \left(-f_i(x_i) + x_i^T g^k + (r/2)\|x_i - x_i^k\|_2^2 \right) \\ y_j^{k+1} &:= \operatorname*{argmin}_{y_j \in \mathcal{Y}_j} \left(g_j(y_j) + (\rho/2)(y_j - \sum_{i=1}^N x_{ij}^{k+1} - u_j^k)^2 \right) \\ u_j^{k+1} &:= u_j^k + \sum_{i=1}^N x_{ij}^{k+1} - y_j^{k+1}, \end{aligned}$$

where $g^k = \rho(\sum_i x_i^k - y^k + u^k)$ *linearizes* the penalty term $(\rho/2)\|\sum_i x_i - y\|_2^2$, and $(r/2)\|x_i - x_i^k\|_2^2$ is a *proximal term*. Although the above algorithm admits simple parallel implementation, its convergence requires $r > \rho N$. When N is sufficiently large, the x-update in each iteration just slightly changes x_i (due to a large r), making the convergence slow. Hence, linearized ADMM is not well suited for large-scale problems.

Multi-block ADMM is another candidate for solving problem (1). However, it generally requires users to solve their subproblems sequentially rather than in parallel. Moreover, it still lacks theoretical convergence guarantees for general convex objective functions. Indeed, a counter-example has just been reported showing the impossibility of convergence for multi-block ADMM [9].

During the final stage of this paper, we noticed a very similar work [40], which also applies the standard two-block ADMM algorithm to solve multi-block convex problems by using auxiliary

variables. Although the algorithms proposed in [40] can solve more general problems, they require the utility functions to be strictly concave *and* the cost functions to be strictly convex in order to achieve $\mathcal{O}(1/a^k)$ rate of convergence. Such requirements cannot be met in some scenarios. For example, the utility function $f_i(A_i^{-1}x_i)$ in backbone traffic engineering is non-strictly concave even if f_i itself is strictly concave, as we discussed before. We also noticed that our first algorithm is in spirit the same as the algorithm proposed in [7, Chapter 7] for solving the *sharing problem*. The main differences include the stopping rule and the analysis of convergence rates. As we will show in Sec. 4, our stopping rule is based on rigorous analysis, whereas their stopping rule is mainly based on some heuristic principles. Moreover, our analysis of convergence rates in Sec. 4 reveals some potential weakness of Algorithm 1, which motivates us to develop Algorithm 2. In contrast, the need for Algorithm 2 is not recognized in [7], mainly due to the lack of such analysis.

Compared to previous ADMM algorithms developed in the networking context in [41–43], the algorithms proposed here assume weaker technical assumptions to ensure convergence, and have lower computational complexity and lower message-passing overhead. For example, the algorithm in [41] requires strongly convex objective functions and bounded level set in order to achieve convergence (see Theorem 1 in [41]). In contrast, our algorithms converge with *arbitrary* convex objective functions. Moreover, the previous algorithm [41] needs to solve a large-scale quadratic problem at each iteration, whereas our algorithms only involve small-scale subproblems.

4. CONVERGENCE ANALYSIS

In this section, we study the convergence behavior of our distributed algorithms. Our analysis is based on several very recent results on ADMM, and leads to a simple yet rigorous stopping rule.

4.1 Assumptions

We first present the assumptions based on which our algorithms converge.

ASSUMPTION 1. *The optimal solution set of problem* (1) *is nonempty, and the optimal value p^* is finite.*

ASSUMPTION 2. *The utility functions $f_i : \mathbb{R}^n \to \mathbb{R}$ are concave, and the cost function $g_j : \mathbb{R} \to \mathbb{R}$ is convex.*

ASSUMPTION 3. *The constraints $\{\mathcal{X}_i\}$ and $\{\mathcal{Y}_j\}$ are bounded polyhedral sets.*

Assumptions 1 and 2 are rather mild. In particular, the utility functions $f_i(\cdot)$ need not to be strictly concave, and the cost functions $g_j(\cdot)$ need not to be strictly convex. Assumption 3 is satisfied in all of our previous problem settings.

The above three assumptions imply that strong duality holds for problem (1) (see, e.g., Proposition 5.2.1 of [3]). Since a feasible solution of problem (1) is also a feasible solution of problem (13), strong duality also holds for problem (13).

REMARK 1. *Assumption 3 is only needed for the proof of strong duality. Hence, Assumption 3 can be replaced by any conditions on $\{\mathcal{X}_i\}$ and $\{\mathcal{Y}_j\}$ so long as strong duality holds. Such conditions are usually weaker than Assumption 3 (see, e.g., [7]).*

4.2 $\mathcal{O}(1/k)$ Rate of Convergence

Although the global convergence of ADMM has been extensively studied, only very recently it has been proved that ADMM has an $\mathcal{O}(1/k)$ rate of convergence [21]. Based on this result, we are able to establish $\mathcal{O}(1/k)$ rate of convergence for our distributed algorithms and provide a simple stopping rule.

Let $(\{x_i^*\}, \{z_i^*\})$ be an primal optimal solution to problem (13) (in particular, we have $x_i^* = z_i^*$), and $\{\lambda_i^*\}$ be a dual optimal solution. Let $u_i^* = \lambda_i^*/\rho$. Their existence follows from the strong duality theorem. We have the following results.

THEOREM 1. *Let $\{\{x_i^k\}, \bar{z}^k, \bar{u}^k\}$ be any sequence generated by the distributed algorithm 1. Let*

$$V^k = \sum_{i=1}^{N} \left(\|z_i^k - z_i^*\|_2^2 + \|u_i^k - u_i^*\|_2^2 \right), \tag{16}$$

and

$$D^k = \sum_{i=1}^{N} \left(\|z_i^{k+1} - z_i^k\|_2^2 + \|u_i^{k+1} - u_i^k\|_2^2 \right). \tag{17}$$

Then starting with any initial point $\{\{x_i^0\}, \bar{z}^0, \bar{u}^0\}$, D^k is nonincreasing, and $D^k \le V^0/(k+1)$ for all k.

THEOREM 2. *Let $\{\{x_i^k\}, \bar{z}^k, \bar{u}^k\}$ be any sequence generated by the distributed algorithm 2. Let*

$$V^k = \sum_{i=1}^{N} \left(\|x_i^k - x_i^*\|_2^2 + \|u_i^k - u_i^*\|_2^2 \right), \tag{18}$$

and

$$D^k = \sum_{i=1}^{N} \left(\|x_i^{k+1} - x_i^k\|_2^2 + \|u_i^{k+1} - u_i^k\|_2^2 \right). \tag{19}$$

Then starting with any initial point $\{\{x_i^0\}, \bar{z}^0, \bar{u}^0\}$, D^k is nonincreasing, and $D^k \le V^0/(k+1)$ for all k.

We now outline the proofs for the above theorems. By symmetry, we need only to prove one of them, say, Theorem 2. For simplicity of notation, we rewrite the problem (13) as

$$\begin{array}{ll} \text{minimize} & f(x) + h(z) \\ \text{subject to} & x - z = 0 \\ & x \in \mathcal{X},\ z \in \mathcal{Z}, \end{array} \tag{20}$$

where $z = (z_1^T, \ldots, z_N^T)^T$, and $h(z) = g(\sum_{i=1}^N z_i)$. Note that $h(\cdot)$ is still a convex function (because affine mappings preserve convexity), and $\mathcal{Z}$ is still a polyhedral set. The scaled form of ADMM for problem (20) (with reversed x-update and z-update) is

$$z^{k+1} := \operatorname*{argmin}_{z \in \mathcal{Z}} \left(h(z) + (\rho/2)\|z - x^k - u^k\|_2^2 \right) \tag{21}$$

$$x^{k+1} := \operatorname*{argmin}_{x \in \mathcal{X}} \left(f(x) + (\rho/2)\|x - z^{k+1} + u^k\|_2^2 \right) \tag{22}$$

$$u^{k+1} := u^k + x^{k+1} - z^{k+1}, \tag{23}$$

which is equivalent to our distributed algorithm 2. Clearly, we have $V^k = \|x^k - x^*\|_2^2 + \|u^k - u^*\|_2^2$, and $D^k = \|x^{k+1} - x^k\|_2^2 + \|u^{k+1} - u^k\|_2^2$. Our proof consists of three steps.

Step 1: V^k is a Lyapunov function. This step can be established by showing the inequality

$$V^{k+1} \le V^k - D^k. \tag{24}$$

Since $D^k \ge 0$ for all k, this states that V^k is indeed a Lyapunov function. The proof of inequality (24) is provided in the Appendix.

Step 2: D^k is non-increasing. This step has been established in [21] through some matrix manipulations. (See the proof of Theorem 4.1 in [21] for details).

Step 3: D^k converges at rate $\mathcal{O}(1/k)$. Using (24) and the fact that D^k is non-increasing, we have

$$\begin{aligned} V^0 &\geq \sum_{t=0}^{k} D^t + V^{k+1} \\ &\geq (k+1)D^k. \end{aligned}$$

Hence, we have $D^k \leq V^0/(k+1)$. This completes the proof of Theorem 2.

REMARK 2. *The above theorems suggest that the sequence $\{D^k\}$ can be used as a natural stopping rule for our distributed algorithms, which decreases at rate $1/k$. This stopping rule is more rigorous compared to that in [7], since their stopping rule is mainly based on heuristic principles. In particular, their stopping-rule sequence does not enjoy the non-increasing property and may fluctuate over iterations.*

Note that our stopping rule can be easily implemented using the MPI Allreduce operation. For example, for our first algorithm, we have $D^k = \sum_{i=1}^{N} \|z_i^{k+1} - z_i^k\|_2^2 + N\|\bar{u}^{k+1} - \bar{u}^k\|_2^2$. Hence, at each iteration, all the processors can perform an Allreduce operation to obtain $\sum_{i=1}^{N} \|z_i^{k+1} - z_i^k\|_2^2$ and then compute $\|\bar{u}^{k+1} - \bar{u}^k\|_2^2$ locally. In actual implementation, this Allreduce operation can be combined with the existing Allreduce operation (which is used to obtain $\bar{x}^{k+1}$) so that there is only one such operation for each iteration.

4.3 $\mathcal{O}(1/a^k)$ Rate of Convergence

We next prove $\mathcal{O}(1/a^k)$ rate of convergence for our distributed algorithms under certain additional assumptions. We need to introduce two definitions.

DEFINITION 1 (STRONG CONVEXITY). *A function $f : \mathbb{R}^n \to \mathbb{R}$ is* strongly convex *with constant $\nu > 0$, if $f(x) - \frac{\nu}{2}\|x\|_2^2$ is convex. A function f is strongly concave if $-f$ is strongly convex.*

DEFINITION 2 (LIPSCHITZ CONTINUOUS GRADIENT). *A function $f : \mathbb{R}^n \to \mathbb{R}$ has a Lipschitz continuous gradient ∇f with constant $\kappa > 0$, if for all $x_1, x_2 \in \mathbb{R}^n$,*

$$\|\nabla f(x_1) - \nabla f(x_2)\|_2 \leq \kappa \|x_1 - x_2\|_2.$$

We have the following results.

THEOREM 3. *Let $\{\{x_i^k\}, \bar{z}^k, \bar{u}^k\}$ be any sequence generated by the distributed algorithm 1. Let V^k be the Lyapunov function defined in* (16)*. Assume that the cost functions $g_j(\cdot)$ are strictly convex with Lipschitz continuous gradients. Then starting with any initial point $\{\{x_i^0\}, \bar{z}^0, \bar{u}^0\}$, there exists some $\delta > 0$ such that $V^k \leq V^0/(1+\delta)^k$ for all k.*

THEOREM 4. *Let $\{\{x_i^k\}, \bar{z}^k, \bar{u}^k\}$ be any sequence generated by the distributed algorithm 2. Let V^k be the Lyapunov function defined in* (18)*. Assume that the utility functions $f_i(\cdot)$ are strictly concave with Lipschitz continuous gradients. Then starting with any initial point $\{\{x_i^0\}, \bar{z}^0, \bar{u}^0\}$, there exists some $\delta > 0$ such that $V^k \leq V^0/(1+\delta)^k$ for all k.*

Proof: By Jensen's inequality, f is strongly convex, if and only if for all $x_1, x_2 \in \mathbb{R}^n$, and all $\theta \in [0,1]$,

$$\begin{aligned} f(\theta x_1 + (1-\theta)x_2) \leq \theta f(x_1) + (1-\theta) f(x_2) \\ - \frac{1}{2}\nu\theta(1-\theta)\|x_1 - x_2\|_2^2. \end{aligned}$$

In particular, if f is strictly convex with a bounded domain, then f is strongly convex. Since the cost functions $g_j(\cdot)$ have bounded domains, they are strongly convex. Applying the results in [12] (in particular, see scenario 1 in Table 1.1[1]) gives $V^k \leq V^0/(1+\delta)^k$, which proves Theorem 3. A similar argument proves Theorem 4. □

REMARK 3. *The above theorems suggest that the distributed algorithm 1 is better suited for the case when only the cost functions are strictly convex, and the algorithm 2 is better suited for the case when only the utility functions are strictly concave. In this sense, our two algorithms have different strengths and complement each other in a natural way, as summarized in the Table 1 below.*

In addition, these two theorems help us to choose the step-size ρ. In particular, one can show that the parameter $\delta = \min\{c_9/\rho, c_{11}\rho\}$, where c_9 and c_{11} are given in [12]. This can be used as a guideline for choosing the step-size ρ such that the parameter δ is maximized.

Table 1: Comparison of two algorithms.

case	strictly convex	Lipschitz continuous	recommendation
1	none	none	Algorithm 1 or 2
2	$\{g_j\}$	$\{g_j\}$	Algorithm 1
3	$\{-f_i\}$	$\{f_i\}$	Algorithm 2
4	$\{-f_i\}, \{g_j\}$	$\{f_i\}, \{g_j\}$	Algorithm 1 or 2

5. EMPIRICAL STUDY

We present our empirical study of the performance of the distributed ADMM algorithms. For this purpose, it suffices to choose one of the two cloud traffic management problems since they are equivalent in nature. We use the geographical load balancing problem (2) with the utility and cost functions (5) and (6) as the concrete context of the performance evaluation. This problem corresponds to the most general case 1 in Table 1 since (5) is non-strictly concave and (6) is non-strictly convex. Thus it can be solved using either Distributed ADMM Algorithm 1 or 2. We use Algorithm 1 in all of our simulations. Note that if the objective function exhibits strict convexity, better simulation results can be obtained according to Theorem 3 and 4. In other words, we mainly focus on the "worse-case" performance of the algorithms in this section. We plan to make all our simulation codes publicly available after the review cycle.

5.1 Setup

We randomly generate each user's request demand t_i, with an average of 9×10^4. We then normalize the workloads to the number of servers, assuming each request requires 10% of a server's CPU. We assume the prediction of request demand is done accurately since prediction error is immaterial to performance of the optimization algorithms. The latency l_{ij} between an arbitrary pair of user and data center is randomly generated between 50 ms and 100 ms.

We set the number of data centers (facilities) $n = 10$. Each data center's capacity c_j is randomly generated so that the total capacity $\sum_j c_j$ is 1.4x the total demand. We use the 2011 annual average day-ahead on peak prices [15] at 10 different local markets as the

[1]Note that there is a typo in Table 1.1 of [12], in which $Q \succ 0$ should be $Q \succeq 0$.

power prices P_j for data centers. The servers have peak power $P_{\text{peak}} = 200$ W, and consume 50% power at idle. The PUE is 1.5. These numbers represent state-of-the-art data center hardware [14, 37].

We set the penalty parameter ρ of the ADMM algorithm to $\rho = 10^{-3}$ after an empirical sweep of $\rho \in \{10^{-4}, 10^{-3}, \ldots, 10^3, 10^4\}$. Although a more fine-grained search for ρ can further improve the performance of our algorithms, we confine ourselves to the above 9 choices to demonstrate the practicality.

5.2 Convergence and Scalability

We evaluate the convergence of Algorithm 1 under the previous setup. We vary the problem size by changing the number of users $N \in \{10^2, 10^3, 10^4, 10^5\}$ and scaling data center capacities linearly with N. We observe that our algorithm converges quickly after 50 iterations in all cases, *independent* of the problem size.

Convergence of objective functions. Figure 3 and 6 plot the convergence of objective values for $N = 10^2$ and $N = 10^4$, respectively. Notice that the objective values for $N = 10^4$ are roughly 100 times the corresponding values for $N = 10^2$ at each iteration. This means that our algorithm has excellent scalability, which is very helpful in practice. Since the number of iterations is independent of the problem size, it suggests that our algorithm can solve a large-scale problem with (almost) the same running time by simply scaling the amount of computing resources linearly with the number of users.

Convergence of D^k. Figure 4 and 7 show the trajectory of D^k as defined in (17) for $N = 10^2$ and $N = 10^4$, respectively. We observe that D^k is indeed non-increasing in both cases. Further, the two figures are in log scale, implying that D^k decreases sublinearly, which confirms Theorem 1 for the $\mathcal{O}(1/k)$ convergence rate. In addition, one can see that D^k scales linearly with N as expected from its definition. This implies that D^k is an ideal candidate for the stopping rule: the algorithm can be terminated when D^k/N is below a certain threshold.

Convergence of primal residuals. Figure 5 and 8 show the trajectory of the primal residual, which is defined as $\sum_i^N \|x_i - z_i\|_2^2$ here. It reflects how well the constraints $\{x_i = z_i\}$ are satisfied, and is sometimes called the primal feasibility gap. For example, if the primal residual is 10^4 for $N = 10^2$ (or, 10^6 for $N = 10^4$), then on average each $\|x_i - z_i\|$ is around 10, which is already small enough since x_i is in the order of 10^4. Hence, we conclude that the constraints are well satisfied after 50 iterations in both cases.

5.3 Fault-tolerance

We have observed that our algorithms converge fast to the optimal solution for large-scale problems. Yet, because failures are the norm rather than the exception, fault-tolerance is arguably the most important design objective for parallel computing frameworks that involve a large number of servers currently [11]. A parallel algorithm that is inherently robust against failures in the intermediate steps is highly desirable for practical deployment. To investigate the fault-tolerance of our algorithm, we carry out a new set of simulations where each user fails to update x_i^k with a probability p at each iteration (independent of each other). Whenever a failure happens, user i simply reuses its previous solution by setting $x_i^{k+1} := x_i^k$.

Figure 9–11 plot the convergence with different failure probabilities for $N = 10^2$, and Figure 12–14 for $N = 10^4$. Specifically, Figure 9 and 12 plot the relative error in objective value with failures (i.e. OBJ_FAIL/OBJ $-$ 1, where OBJ_FAIL is the objective value with failures, and OBJ is the objective value when every step is solved correctly). We observe that increasing the failure probability from 5% to 10% increases the relative error, causing the solution quality to degrade at the early stage. Yet surprisingly, the impact is very insignificant: The relative error is at most 1.5%, and ceases to 0 after 100 iterations. In fact, after 50 iterations the relative error is only around 0.2% for both problem sizes.

Moreover, failures do not affect the convergence of the algorithm at all. This is indicated by the relative error plots, and further illustrated by the overlapping curves in Figure 10, 11, 13, and 14 for D^k and primal residual.

Thus, we find that our distributed ADMM algorithms are inherently fault-tolerant, with less than 1% optimality loss and essentially the same convergence speed for up to 10% failure rate. They are robust enough to handle temporary failures that commonly occur in production systems.

5.4 Comparison with Dual Decomposition

We also simulate the conventional dual decomposition approach with subgradient methods as explained in Sec. 3.4 to solve problem (2). The step size ρ^k is chosen following the commonly accepted diminishing step size rule [6], with $\rho^k = 10^{-5}/\sqrt{k}$.

We plot the trajectory of objective values in Figure 15, and that of primal residuals in Figure 16. Compare to Algorithm 1, dual decomposition yields wildly fluctuating results. Though the objective value decreases to the same level as Algorithm 1 after about 200 iterations, the more meaningful primal variables $\{x_i\}$ never converge even after 400 iterations. One can see from Figure 16 that the primal residual does not decrease below 10^7. This implies that the equality constraints $\{x_i = z_i\}$ are not well-satisfied during the entire course, and the primal variables $\{x_i\}$ still violate the capacity constraints after 400 iterations.

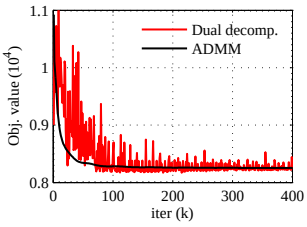

Figure 15: Objective value. $N = 10^2$.

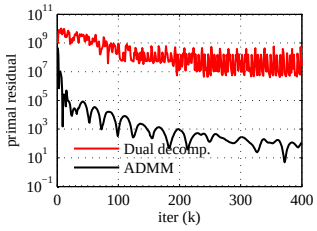

Figure 16: Primal residual. $N = 10^2$.

This phenomenon is due to the *oscillation problem* [28] when dual decomposition method is applied to non-strictly convex objective functions. To mitigate this problem, one can make the objective function strictly convex by adding a small penalty term, e.g., $\rho_1\|x\|_2^2 + \rho_2\|z\|_2^2$). Nevertheless, we found that the primal variables $\{x_i\}$ still converge very slowly after an extensive trial of different (ρ_1, ρ_2).

To summarize, our simulation results confirm our theoretical analysis, demonstrate fast convergence of our algorithms in various settings, and highlight several additional advantages, especially the scalability and fault-tolerance.

6. RELATED WORK

6.1 Network Utility Maximization

Network utility maximization (NUM) [4, 39] is closely related to our multi-facility resource allocation problem. A standard technique for solving NUM problems is dual decomposition. Dual decomposition was first applied to the NUM problem in [26], and has lead to a rich literature on distributed algorithms for network

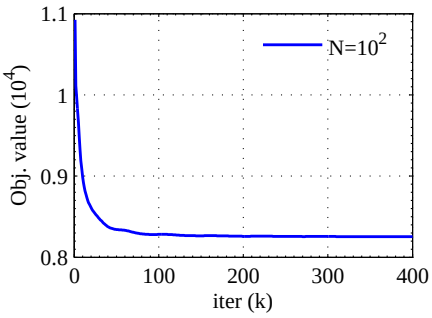

Figure 3: Objective value. $N = 10^2$.

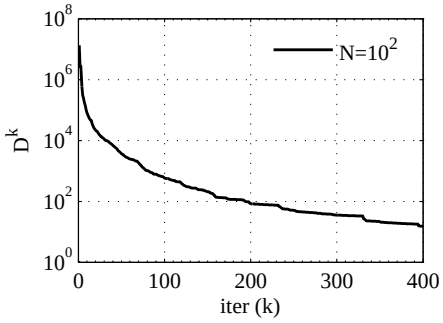

Figure 4: D^k. $N = 10^2$.

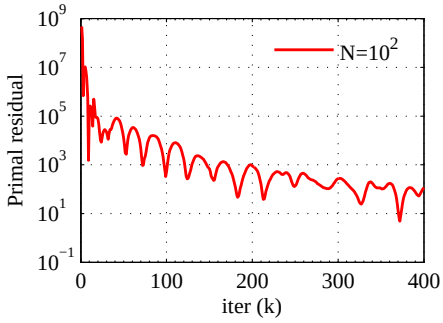

Figure 5: Primal residual. $N = 10^2$.

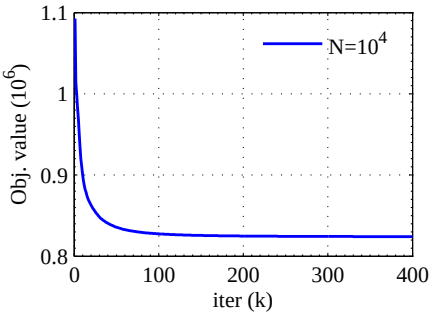

Figure 6: Objective value. $N = 10^4$.

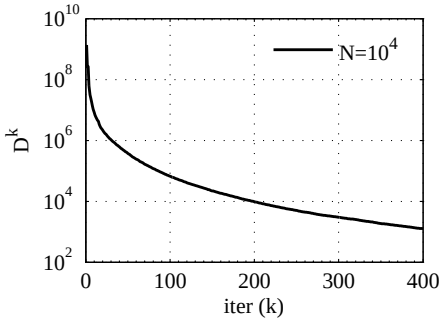

Figure 7: D^k. $N = 10^4$.

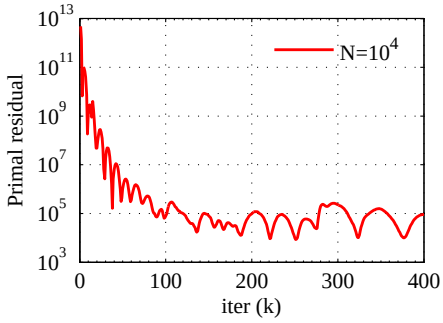

Figure 8: Primal residual. $N = 10^4$.

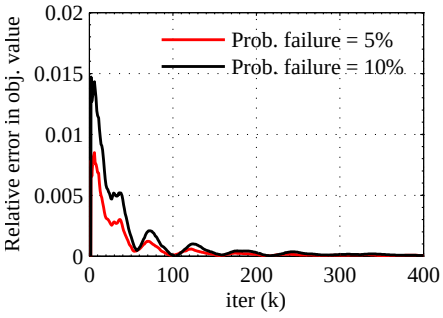

Figure 9: Relative errors in objective value. $N = 10^2$.

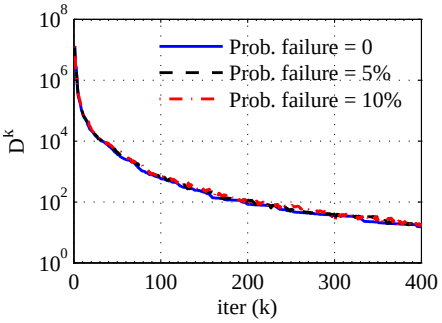

Figure 10: D^k. $N = 10^2$.

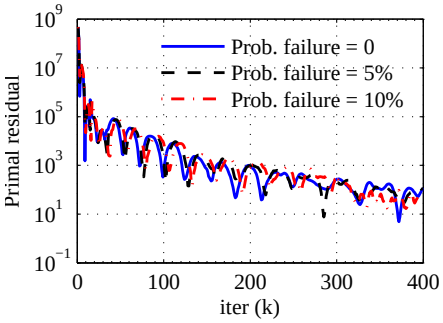

Figure 11: Primal residual. $N = 10^2$.

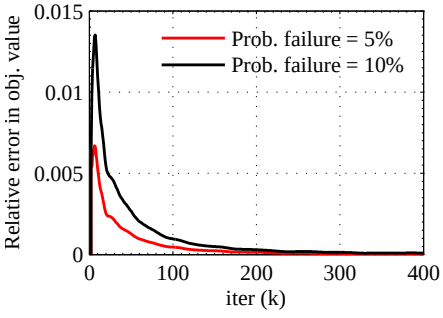

Figure 12: Relative errors in objective value. $N = 10^4$.

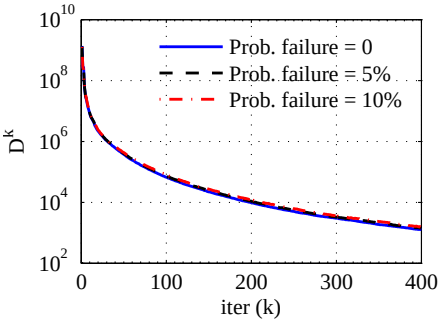

Figure 13: D^k. $N = 10^4$.

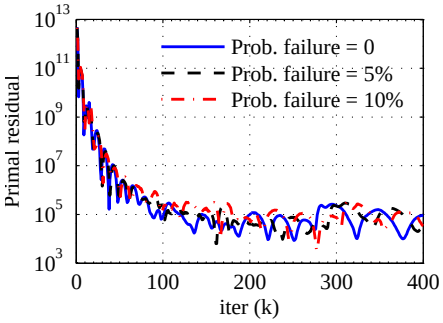

Figure 14: Primal residual. $N = 10^4$.

rate control [10,32,36] and new understandings of existing network protocols [31]. Despite its popularity, dual decomposition suffers from slow convergence, and generally requires the utility functions to be strictly concave and the cost functions to be strictly convex. Our ADMM-type algorithms overcome these difficulties, achieving faster convergence under weaker assumptions as discussed in Sec. 3.4 in detail. Another advantage is that our algorithms can easily handle multi-path routing, whereas dual decomposition requires non-trivial modifications to address multi-path routing [28].

6.2 ADMM and Its Variations

Originally proposed in the 1970s, ADMM has recently received much research attention and found practical use in many areas, due to its superior empirical performance in solving large-scale convex optimization problems [7]. While the convergence of ADMM is well known in the literature (see, e.g., [5,7]), its rate of convergence has only been established very recently. [21] proves rate-$\mathcal{O}(1/k)$ of convergence under the most general assumptions. [12] proves rate-$\mathcal{O}(1/a^k)$ of convergence under the assumptions that the objective function is strongly convex and its gradient is Lipschitz continuous in at least one block of variables. These results provide theoretical foundation for our algorithm design and analysis. ADMM has two important variations: linearized ADMM [21] and multi-block ADMM [20, 24]. However, they are not particularly suitable for problem (1), as discussed thoroughly in Section 3.4. In contrast, our ADMM-type algorithms exploit the structure of problem (1), thereby enjoying a number of unique advantages. Our algorithms are in spirit similar to a recent submission [40] and the algorithm proposed in [7, Chapter 7]. Still, our algorithms have clear advantages as discussed in Section 3.4.

6.3 Cloud Traffic Management

Cloud service providers operate two distinct types of WANs: user-facing WANs and backbone WANs [25]. The user-facing WAN connects cloud users and data centers by peering and exchanging traffic with ISPs. Through optimized load balancing, this type of networks can achieve a desired trade-off between performance and cost [17, 29, 30, 37, 41–43]. The backbone WAN provides connectivity among data centers for data replication and synchronization. Rate control and multi-path routing [18,23,25] can significantly increase link utilization and reduce operational costs of the network. Previous work developed different optimization methods for each application scenario separately, whereas our work provides a unified framework well suited to a wide range of network scenarios.

7. CONCLUSION

In this work, we have introduced a general framework for studying various cloud traffic management problems. We have abstracted these problems as a multi-facility resource allocation problem and developed two distributed algorithms that are amenable to parallel implementation. We have studied the convergence rates of our algorithms under various scenarios. When the utility function is non-strictly concave and the cost function is non-strictly convex, our algorithms achieve $\mathcal{O}(1/k)$ rate of convergence. When the utility function is strictly concave *or* the cost function is strictly convex, our algorithms achieve $\mathcal{O}(1/a^k)$ rate of convergence. Our analysis also provides a simple yet rigorous stopping rule as well as a guideline on how to choose the step-size ρ.

We have shown that, compared to dual decomposition and other ADMM-type distributed solutions, our algorithms have a number of unique advantages, such as achieving faster convergence under weaker assumptions, and enjoying lower computational complexity and lower message-passing overhead. These advantages are further confirmed by our extensive empirical studies. Moreover, our simulation results demonstrate some additional advantages of our algorithms, including the scalability and fault-tolerance, which we believe are highly desirable for large-scale production systems.

8. REFERENCES

[1] S. Agarwal, M. Kodialam, and T. V. Lakshman. Traffic engineering in software defined networks. In *Proc. IEEE INFOCOM*, 2013.

[2] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *Proc. ACM SIGCOMM*, 2011.

[3] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[4] D. P. Bertsekas. *Network Optimization: Continuous and Discrete Models*. Athena Scientific, 1998.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.

[6] S. Boyd and A. Mutapcic. Subgradient methods. Lecture notes of EE364b, Stanford University, Winter Quarter 2006-2007. `http://www.stanford.edu/class/ee364b/notes/subgrad_method_notes.pdf`.

[7] 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*, 3(1):1–122, 2010.

[8] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *Proc. USENIX NSDI*, 2005.

[9] C. Chen, B. He, Y. Ye, and X. Yuan. The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent. Technical report, September 2013.

[10] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle. Layering as optimization decomposition: A mathematical theory of network architectures. *Proc. IEEE*, 95(1):255–312, January 2007.

[11] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. OSDI*, 2004.

[12] W. Deng and W. Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. Technical report, Department of Computational and Applied Mathematics, Rice University, 2012.

[13] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja. Mate: Mpls adaptive traffic engineering. In *Proc. IEEE INFOCOM*, 2001.

[14] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proc. ISCA*, 2007.

[15] Federal Energy Regulatory Commission. U.S. electric power markets. `http://www.ferc.gov/market-oversight/mkt-electric/overview.asp`, 2011.

[16] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Netw.*, 17(6):6–16, November 2003.

[17] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav. It's not easy being green. In *Proc. ACM SIGCOMM*, 2012.

[18] A. Ghosh, S. Ha, E. Crabbe, and J. Rexford. Scalable multi-class traffic management in data center backbone networks. *IEEE J. Sel. Areas Commun.*, 31(12):1–12, December 2013.

[19] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *ACM SIGCOMM Comput. Commun. Rev.*, 35(5):41–54, October 2005.

[20] D. Han and X. Yuan. A note on the alternating direction method of multipliers. *J. Optim. Theory Appl.*, 155:227–238, 2012.

[21] B. He and X. Yuan. On non-ergodic convergence rate of Douglas-Rachford alternating direction method of multipliers. Technical report, 2012.

[22] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, 1969.

[23] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *Proc. ACM SIGCOMM*, 2013.

[24] M. Hong and Z.-Q. Luo. On the linear convergence of the alternating direction method of multipliers, August 2012.

[25] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined WAN. In *Proc. ACM SIGCOMM*, 2013.

[26] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *J. Operat. Res. Soc.*, 49(3):237–252, March 1998.

[27] R. Kohavi, R. M. Henne, and D. Sommerfield. Practical guide to controlled experiments on the web: Listen to your customers not to the hippo. In *Proc. ACM SIGKDD*, 2007.

[28] X. Lin and N. B. Shroff. Utility maximization for communication networks with multi-path routing. *IEEE Trans. Autom. Control*, 51(5):766–781, May 2006.

[29] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser. Renewable and cooling aware workload management for sustainable data centers. In *Proc. ACM Sigmetrics*, 2012.

[30] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew. Greening geographical load balancing. In *Proc. ACM Sigmetrics*, 2011.

[31] S. H. Low. A duality model of TCP and queue management algorithms. *IEEE/ACM Trans. Netw.*, 11(4):525–536, August 2003.

[32] S. H. Low and D. E. Lapsley. Optimization flow control—i: Basic algorithm and convergence. volume 7, pages 861–874, December 1999.

[33] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Trans. Netw.*, 8(5):556–567, October 2000.

[34] S. Narayana, J. W. Jiang, J. Rexford, and M. Chiang. Distributed wide-area traffic management for cloud services. In *Proc. ACM Sigmetrics*, Extended Abstract, 2012.

[35] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai network: A platform for high-performance Internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, August 2010.

[36] D. Palomar and M. Chiang. A tutorial on decomposition methods and distributed network resource allocation. *IEEE J. Sel. Areas Commun.*, 24(8):1439–1451, August 2006.

[37] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the electricity bill for Internet-scale systems. In *Proc. SIGCOMM*, 2009.

[38] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. The MIT Press, 1996.

[39] R. Srikant. *The Mathematics of Internet Congestion Control*. Birkhäuser, 2004.

[40] 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. Technical report, August 2013.

[41] H. Xu, C. Feng, and B. Li. Temperature aware workload management in geo-distributed datacenters. In *Proc. USENIX ICAC*, 2013.

[42] H. Xu, C. Feng, and B. Li. Temperature aware workload management in geo-distributed datacenters. In *Proc. ACM Sigmetrics*, Extended Abstract, 2013.

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

[44] Z. Zhang, M. Zhang, A. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian. Optimizing cost and performance in online service provider networks. In *Proc. USENIX NSDI*, 2010.

APPENDIX

Proof of inequality (24)

We need a technical lemma and two simple inequalities.

LEMMA 1. *Let $h_1, h_2 : \mathbb{R}^n \to \mathbb{R}$ be two convex functions. Assume that h_2 is differentiable in $\mathbb{R}^n$, and that $\mathcal{X} \subset \mathbb{R}^n$ is a closed convex set. Then*

$$x^* \in \operatorname*{argmin}_{x\in\mathcal{X}} \; h_1(x) + h_2(x)$$

if and only if

$$x^* \in \operatorname*{argmin}_{x\in\mathcal{X}} \; h_1(x) + x^T\nabla h_2(x^*).$$

The proof of Lemma 1 is standard and thus omitted here.

Let $p^k = f(x^k) + h(z^k)$, and p^* denote the optimal value for problem (20). The first inequality is

$$p^* - p^{k+1} \le (\lambda^*)^T(u^{k+1} - u^k). \tag{25}$$

Proof: Since (x^*, z^*) and λ^* is a primal-dual optimal solution pair, by the Saddle Point Theorem, we have

$$f(x^*) + h(z^*) \le f(x^{k+1}) + h(z^{k+1}) + (\lambda^*)^T(x^{k+1} - z^{k+1}).$$

Using $u^{k+1} = u^k + x^{k+1} - z^{k+1}$, the right-hand side is $p^{k+1} + (\lambda^*)^T(u^{k+1} - u^k)$. This gives (25). □

The second inequality is

$$\begin{aligned} p^{k+1} - p^* \le &-\rho(u^{k+1} - u^k)^T u^{k+1} \\ &+ \rho(x^* - x^{k+1} + u^{k+1} - u^k)^T(x^{k+1} - x^k). \end{aligned} \tag{26}$$

Proof: Applying Lemma 1 to the x-update (22), we have x^{k+1} minimizes $f(x) + (x - x^{k+1})^T\rho(x^{k+1} - z^{k+1} + u^k)$ for all $x \in \mathcal{X}$. In particular, we have

$$f(x^{k+1}) \le f(x^*) + \rho(x^* - x^{k+1})^T(x^{k+1} - z^{k+1} + u^k).$$

Since $u^{k+1} = x^{k+1} - z^{k+1} + u^k$, we obtain

$$f(x^{k+1}) \le f(x^*) + \rho(x^* - x^{k+1})^T u^{k+1}. \tag{27}$$

A similar argument gives

$$h(z^{k+1}) \le h(z^*) + \rho(z^* - z^{k+1})^T(x^{k+1} - x^k - u^{k+1}) \tag{28}$$

Adding the two inequalities above, we obtain

$$p^{k+1} - p^* \le \rho(z^{k+1} - x^{k+1})^T u^{k+1} + \rho(z^* - z^{k+1})^T(x^{k+1} - x^k).$$

Since $u^{k+1} = u^k + x^{k+1} - z^{k+1}$, we have

$$p^{k+1} - p^* \le -\rho(u^{k+1} - u^k)^T u^{k+1} + \rho(z^* - z^{k+1})^T(x^{k+1} - x^k).$$

Note that

$$z^* - z^{k+1} = x^* - x^{k+1} + x^{k+1} - z^{k+1} = x^* - x^{k+1} + u^{k+1} - u^k.$$

This gives (26). □

Adding (25) and (26), and regrouping terms gives

$$\begin{aligned} &(x^* - x^{k+1})^T(x^{k+1} - x^k) + (u^* - u^{k+1})^T(u^{k+1} - u^k) \\ &\qquad + (x^{k+1} - x^k)^T(u^{k+1} - u^k) \ge 0. \end{aligned} \tag{29}$$

Recall that x^{k+1} minimizes $f(x) + \rho x^T u^{k+1}$ and x^k minimizes $f(x) + \rho x^T u^k$, we have

$$f(x^{k+1}) - f(x^k) + \rho(x^{k+1} - x^k)^T u^{k+1} \le 0$$

and

$$f(x^k) - f(x^{k+1}) + \rho(x^k - x^{k+1})^T u^k \le 0.$$

Adding the two inequalities above gives

$$(x^{k+1} - x^k)^T(u^{k+1} - u^k) \le 0.$$

Substituting this into (29), we have

$$(x^* - x^{k+1})^T(x^{k+1} - x^k) + (u^* - u^{k+1})^T(u^{k+1} - u^k) \ge 0.$$

Note that

$$\begin{aligned} \|x^* - x^k\|_2^2 &= \|(x^* - x^{k+1}) + (x^{k+1} - x^k)\|_2^2 \\ &= \|x^* - x^{k+1}\|_2^2 + \|x^{k+1} - x^k\|_2^2 \\ &\quad + 2(x^* - x^{k+1})^T(x^{k+1} - x^k). \end{aligned}$$

Similarly,

$$\begin{aligned} \|u^* - u^k\|_2^2 = \|u^* - u^{k+1}\|_2^2 + \|u^{k+1} - u^k\|_2^2 \\ + 2(u^* - u^{k+1})^T(u^{k+1} - u^k). \end{aligned}$$

Combining the above three results, we obtain

$$\begin{aligned} V^k &= V^{k+1} + \|x^{k+1} - x^k\|_2^2 + \|u^{k+1} - u^k\|_2^2 \\ &\quad + 2(x^* - x^{k+1})^T(x^{k+1} - x^k) \\ &\quad + 2(u^* - u^{k+1})^T(u^{k+1} - u^k) \\ &\ge V^{k+1} + \|x^{k+1} - x^k\|_2^2 + \|u^{k+1} - u^k\|_2^2. \end{aligned}$$

This proves (24). □