Joint Request Mapping and Response Routing for Geo-distributed Cloud Services

Hong Xu, Baochun Li
henryxu, bli@eecg.toronto.edu
Department of Electrical and Computer Engineering
University of Toronto

***Abstract*—Many cloud services are running on geographically distributed datacenters for better reliability and performance. We consider the emerging problem of joint request mapping and response routing with distributed datacenters in this paper. We formulate the problem as a general workload management optimization. A utility function is used to capture various performance goals, and the location diversity of electricity and bandwidth costs are realistically modeled. To solve the large-scale optimization, we develop a distributed algorithm based on the alternating direction method of multipliers (ADMM). Following a decomposition-coordination approach, our algorithm allows for a parallel implementation in a datacenter where each server solves a small sub-problem. The solutions are coordinated to find an optimal solution to the global problem. Our algorithm converges to near optimum within tens of iterations, and is insensitive to step sizes. We empirically evaluate our algorithm based on real-world workload traces and latency measurements, and demonstrate its effectiveness compared to conventional methods.**

I. Introduction

Cloud services have already become an essential part of our life. Notable examples include online search (Google), video streaming (Netflix), social networking (Facebook), etc. Many cloud services are deployed on a geographically distributed infrastructure, i.e. datacenters located in different regions as shown in Fig. 1, for better performance and reliability.

Two problems are of particular importance to the efficient operation of cloud services running on geographically distributed datacenters. First, client requests across the wide area must be directed to an appropriate datacenter, which constitutes the *request mapping* problem. Second, a datacenter is usually connected through multiple ISP links to the Internet, a practice known as multi-homing [13]. When a request is processed, the response packets must be sent back to the client through one of the links available, which corresponds to the *response routing* problem.

Today, request mapping and response routing are managed independently, leading to poor performance and high costs in many cases [17], [21]. For example, too many requests may be directed to a datacenter whose upstream links then become congested, resulting in long queueing delays and poor performance. The objectives of the two decisions can also be misaligned and lead to sub-optimal equilibria.

In light of the problems, we study the *joint* request mapping and response routing problem that has started to gain attention recently [21] with distributed datacenters. Specifically, we formulate the problem as a general workload management optimization, where key performance and cost issues are realistically modeled. We use a utility function of the average latency [33] to capture various performance goals providers wish to achieve for their services. We consider both the electricity and bandwidth costs, which exhibit significant location and provider diversity [26], [28] and together account for the majority of the datacenter operational expense (OPEX) [14].

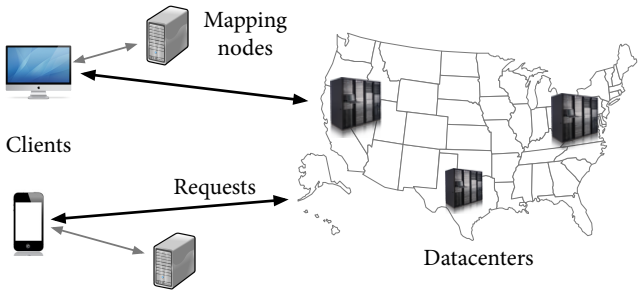

Fig. 1. A cloud service running on a geographically distributed infrastructure.

The workload management problem is a convex optimization, and can be solved in a centralized way. However, it is inherently a very large-scale problem that makes a centralized algorithm inefficient. In a production system, the problem typically has millions of variables and hundreds of thousands of constraints as we shall illustrate in Sec. II-D. A centralized algorithm cannot take advantage of the abundant server resources in a datacenter to parallelize the computation for such large-scale problems. Though solving the optimization at a central server periodically is possible, such a design also makes the system less responsive to handle sudden changes in request rates (i.e. flash crowds) or network conditions (i.e. link failures). In these situations, a solution with fast computation and modest accuracy is more desirable.

Thus, for reasons of performance, scalability, and robustness, we are motivated to develop a *distributed* solution for the workload management problem. Our algorithm is based on the *alternating direction method of multipliers* (ADMM), a simple yet powerful algorithm that recently has found practical use in many large-scale distributed convex optimization problems [6]. ADMM works by first separating the objective and variables into two parts, and then alternatively optimizing one set of variables that accounts for one part of the objective to iteratively reach the optimum. Merits of ADMM, compared to conven-

tional methods such as subgradient methods [5], are its fast convergence to modest accuracy, insensitivity to step sizes, and robustness without strong assumptions such as strict convexity of the objective function [4], [6].

Our contributions are three-fold. First, we develop a general formulation of the joint request mapping and response routing problem for cloud services in Sec. II. We use utility functions to capture various performance objectives, and consider the location diversity of the associated electricity and bandwidth costs. Our second contribution is a novel distributed algorithm based on ADMM to solve the large-scale optimization problem efficiently (Sec. III). We demonstrate that after a transformation, the problem can be decomposed into many small sub-problems, the solutions of which are coordinated to find the global optimal solution, and can be efficiently solved in the general case. We further provide solutions in analytical form for the case when the utility function is affine, and discuss issues pertaining to a parallel implementation of the algorithm in the cloud.

Our third contribution is an empirical evaluation of the algorithm using the Wikipedia workload traces [27], as well as real-world latency measurements [19] in Sec. IV. It is demonstrated that our algorithm offers near-optimal performance within 20 iterations. Finally, we stress that the techniques developed in the paper to transform the problem and apply ADMM are fairly general, and may be applicable to problems in datacenters and other domains, where an efficient parallel algorithm is required to solve large-scale convex optimization problems.

II. A Framework for Joint Mapping and Routing

Let us start by presenting our model and optimization framework.

A. Infrastructure

We consider a provider that runs her cloud service over a set of datacenters $\mathcal{N}$ in distinct geographical regions. Each datacenter n is multi-homed to a set of ISP links $\mathcal{M}_n$, each with a fixed capacity. Let $\mathcal{I}$ denote the set of clients, where in this work a client i is simply a unique IP prefix similar to [24].

The provider deploys a number of mapping nodes as shown in Fig. 1 to map client requests to an appropriate datacenter based on certain criteria. This is the *request mapping* decision. In practice, these mapping nodes can be authoritative DNS servers as used by Akamai and most CDNs, or HTTP ingress proxies as used by Google and Yahoo [24], [30]. We allow a mapping node to arbitrarily split a client's request traffic among the set of datacenters. DNS servers and HTTP proxies can achieve such flexibility in commercial products [17], [30].

When a datacenter finishes serving a request, it sends the response packets back to the client through one of the available ISP links. This corresponds to the *response routing* decision. Today's BGP routing picks a single egress ISP link for each IP prefix. We relax this constraint and allow the provider to arbitrarily split the response traffic among all ISP links, which is commonly accepted in the literature [21], [23]. Such fractional routing can be achieved by hash-based traffic splitting in practice [8].

Without loss of generality, we view every possible combination of datacenter and ISP link as a virtual *stub datacenter*, a concept we use to facilitate our analyses in the sequel. We let $j \in \mathcal{J}, \mathcal{J} := \mathcal{N} \times \{\mathcal{M}_n\}$ denote a stub datacenter, i.e. the tuple $\langle n, m \rangle, n \in \mathcal{N}, m \in \mathcal{M}_n$. Each stub datacenter then has a finite capacity C_j determined by its corresponding ISP link's capacity. Here we implicitly assume that the link capacity is the bottleneck of the service compared to the datacenter's computational capability, which is generally the case in reality. The request mapping and response routing decisions can then be treated jointly as a single *workload management* optimization between the clients and the stub datacenters.

The provider periodically, e.g. hourly or daily, computes the workload management decisions to better cope with dynamic request traffic under normal operations [12], [13], [26]. We use $\alpha_{ij} \in [0, 1]$ to denote the proportion of requests distributed to stub datacenter j from client i. α_{ij} is our decision variable. We assume the provider employs statistical machine learning techniques [23], [25] to predict the traffic demand of each client D_i before each optimization interval. Such an assumption is commonly made in the literature [21], [30], [31].

B. Performance

Latency is arguably the most important performance metric for most cloud services. A small increase in the user-perceived latency can cause substantial revenue loss for the provider [16]. In this paper we focus on the end-to-end propagation latency between users and datacenters, which largely accounts for the user-perceived latency compared to other factors such as request processing times at datacenters [11], [21].

The provider obtains the propagation latency L_{ij} between client i and stub datacenter j through active measurements [19] or other means. A client's performance depends on the *average* propagation latency its requests receive $\sum_j \alpha_{ij} L_{ij}$ through a generic utility function U. U can take various forms depending on the performance goals the provider pursues. We only require that U is a decreasing, differentiable, and concave function. This utility notion allows us a considerable amount of expressiveness. For example, it can incorporate *fairness* among clients by using the canonical alpha-fair utility functions [20].

C. Costs

Two kinds of operating costs — electricity and bandwidth — are involved in serving client requests, both of which scale with the total volume of the workload. The electricity price exhibits significant location diversity which has been exploited to save costs for datacenters [18], [26], [31]. We use P_j^E to denote the power price of the stub datacenter j, which is determined by the location of the corresponding datacenter. The bandwidth price varies across ISPs and also exhibits location diversity in practice [28], and is denoted by P_j^B depending on the corresponding ISP and location of the stub datacenter. In reality many ISPs adopt the 95-percentile charging scheme. However we assume the bandwidth cost is linear with the traffic volume. Optimizing a linear cost in each interval can reduce the monthly 95-percentile bill [32].

D. Problem Formulation

We are now in a position to formally formulate the workload management problem as an optimization that maximizes the total utility of serving the requests, minus the electricity and bandwidth costs incurred.

$$\min_{\alpha} \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} D_i\alpha_{ij}\left(P_j^E + P_j^B\right) - \sum_{i\in\mathcal{I}} D_i U\Big(\sum_{j\in\mathcal{J}}\alpha_{ij}L_{ij}\Big) \quad (1)$$

$$\text{s.t.} \sum_{j\in\mathcal{J}}\alpha_{ij} = 1, \forall i\in\mathcal{I}, \quad (2)$$

$$\sum_{i\in\mathcal{I}}\alpha_{ij}D_i \le C_j, \forall j\in\mathcal{J}, \quad (3)$$

$$\alpha_{ij}\ge 0, \forall i\in\mathcal{I}, j\in\mathcal{J}. \quad (4)$$

(1) is the objective function that poses the maximization problem in an equivalent minimization form. Note that by adding a scalar weight factor in front of the utility function, any desired trade-off point between performance and cost can be achieved. For simplicity we assume the weight is 1. (2) is the *workload conservation* constraint that dictates each client's demand has to be satisfied. (3) is the *capacity* constraint that prevents the ISP link of a stub datacenter from overflow. (4) is simply the non-negativity constraint for the variables.

Our formulation focuses on the performance-cost trade-off. In practice a provider may also need to consider various policies when designing the request mapping and response routing strategies. Though we do not consider policies in this paper, they can be modeled as additional constraints to the problem and do not fundamentally change the formulation.

The optimization (1) is a very large-scale problem. To have a rough understanding, the number of clients represented by the number of unique IP addresses is $O(10^5)$, and the number of datacenters and ISP links is around $O(10^2)$ in some production clouds [17], [21]. This implies that the problem can have $O(10^7)$ variables, and $O(10^5)$ constraints for a production system.

E. Existing Approaches

As argued in Sec. I, the lack of efficiency and robustness in centralized algorithms motivates our design of a distributed solution amenable to parallel implementations in the cloud. The common approach to develop distributed algorithms is to relax the constraints and employ dual decomposition to decompose the problem into many independent sub-problems [9]. Subgradient methods can then be used to update the dual variables towards the optimality of the dual problem [5].

Yet, these approaches are not applicable here. First of all, dual decomposition requires the utility function to be strictly convex, for an affine function will make the Lagrangian unbounded below in α. However, for workload management in cloud computing, an affine utility function is in fact one of the most popular and commonly studied utility functions [21], [31]. More importantly, subgradient methods suffer from the curse of step size. For the output to be close to the optimum, we need to strategically pick the step size at each iteration, leading to the well-known problems of slow convergence and performance oscillation when the problem scale is large. Even if U were indeed a strictly convex function, subgradient methods are not well suited in our problem.

Summarizing the discussions, we need a scalable and practical distributed algorithm that converges fast to modest accuracy, and is not sensitive to step sizes. In the following, we present such an algorithm based on the alternating direction method of multipliers (ADMM) [6].

III. Algorithm Design

We first provide a brief primer on ADMM which is the corner stone of our algorithm design.

A. A Primer on ADMM

ADMM, developed in the 1970s [4], has recently received renewed interest in solving large-scale distributed convex optimization in statistics, machine learning, and related areas [6]. The algorithm solves problems in the form

$$\begin{aligned} \min \quad & f(x)+g(z) \\ \text{s.t.} \quad & Ax+Bz=c, \\ & x\in C_1, z\in C_2, \end{aligned} \quad (5)$$

with variables $x\in\mathbf{R}^n$ and $z\in\mathbf{R}^m$, where $A\in\mathbf{R}^{p\times n}$, $B\in\mathbf{R}^{p\times m}$, and $c\in\mathbf{R}^p$. f and g are convex functions, and C_1, C_2 are non-empty polyhedral sets. Thus, the objective function is *separable* over two sets of variables, which are coupled through an equality constraint.

We can form the augmented Lagrangian [15] by introducing an extra $\mathcal{L}$-2 norm term $\|Ax+Bz-c\|_2^2$ to the objective:

$$L_\rho(x,z,\lambda) = f(x)+g(z)+\lambda^T(Ax+Bz-c) + (\rho/2)\|Ax+Bz-c\|_2^2. \quad (6)$$

$\rho>0$ is the penalty parameter (L_0 is the standard Lagrangian for the problem). The augmented Lagrangian can be viewed as the unaugmented Lagrangian associated with the problem

$$\begin{aligned} \min \quad & f(x)+g(z)+(\rho/2)\|Ax+Bz-c\|_2^2 \\ \text{s.t.} \quad & Ax+Bz=c, \\ & x\in C_1, z\in C_2, \end{aligned}$$

Clearly this problem is equivalent to the original problem (5), since for any feasible x and z the penalty term added to the objective is zero. The benefit of introducing the penalty term is that L_ρ is strictly convex even when f and g are affine, and we can work on the dual problem without strong assumptions on f and g. The penalty term is also called a regularization term and helps substantially improve the convergence of the algorithm.

ADMM solves the dual problem with the iterations:

$$x^{t+1} := \operatorname*{argmin}_{x\in C_1} L_\rho(x, z^t, \lambda^t) \quad (7)$$

$$z^{t+1} := \operatorname*{argmin}_{z\in C_2} L_\rho(x^{t+1}, z, \lambda^t) \quad (8)$$

$$\lambda^{t+1} := \lambda^t + \rho(Ax^{t+1}+Bz^{t+1}-c). \quad (9)$$

It consists of an x-minimization step (7), a z-minimization step (8), and a dual variable update (9). Note the step size

is simply the penalty parameter ρ. Thus, x and z are updated in an alternating or sequential fashion, which accounts for the term *alternating direction*. Separating the minimization over x and z is precisely what allows for decomposition when f or g are separable, which will be useful in our algorithm design.

The optimality and convergence of ADMM can be guaranteed under very mild technical assumptions.

Theorem 1: [4] Assume that the optimal solution set of problem (5) is non-empty, and either C_1 is bounded or else the matrix A^TA is invertible. Then a sequence $\{x^t, z^t, \lambda^t\}$ generated by (7)–(9) is bounded, and every limit point of $\{x^t, z^t\}$ is an optimal solution of the problem (5).

In practice, it is often the case that ADMM converges to modest accuracy within a few tens of iterations [6].

B. Our Algorithm

Our problem (1) cannot be readily solved using ADMM. The constraints (2) and (3) couple all variables together, whereas in ADMM problems the constraints are separable for each set of variables. The coupling is especially difficult, because it happens on two orthogonal dimensions simultaneously: The per-client workload conservation constraint (2) couples α across stub datacenters, and the per-stub datacenter capacity constraint (3) couples α across clients.

To address this challenge, we introduce a new set of auxiliary variables $\beta = \alpha$, and re-formulate the optimization:

$$\begin{aligned}\min_{\alpha,\beta} \sum_{i\in\mathcal{I}} D_i\left(\sum_{j\in\mathcal{J}}\alpha_{ij}P_j^E - U\Big(\sum_{j\in\mathcal{J}}\alpha_{ij}L_{ij}\Big)\right) + \sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{I}} D_i\beta_{ij}P_j^B \\ \text{s.t. } \sum_{j\in\mathcal{J}}\alpha_{ij} = 1, \forall i\in\mathcal{I}, \\ \sum_{i\in\mathcal{I}}\beta_{ij}D_i \le C_j, \forall j\in\mathcal{J}, \\ \alpha_{ij} = \beta_{ij} \ge 0, \forall i\in\mathcal{I}, j\in\mathcal{J}. \end{aligned} \tag{10}$$

This problem (10) is clearly equivalent to the original problem (1). We observe that the new formulation is in the ADMM form (5). The objective function is now separable over two sets of variables α and β. α controls the net utility gain of processing the requests, i.e. utility minus electricity cost, while β determines the bandwidth cost of transmitting the response packets. α and β are connected through an equality constraint. Overall, they control the provider's total utility gain of running the cloud service.

The use of auxiliary variables also enables the separation of per-client and per-stub datacenter constraint sets, which is the key step towards decomposing the problem as we demonstrate now. The augmented Lagrangian of (10) is

$$\begin{aligned}L_\rho(\alpha,\beta,\lambda) = &\sum_i D_i\left(\sum_j \alpha_{ij}P_j^E - U\Big(\sum_j \alpha_{ij}L_{ij}\Big)\right) \\ &+ \sum_j\sum_i D_i\beta_{ij}P_j^B \\ &+ \sum_i\sum_j \left(\lambda_{ij}(\alpha_{ij}-\beta_{ij}) + \rho/2(\alpha_{ij}-\beta_{ij})^2\right).\end{aligned} \tag{11}$$

The dual problem is solved by updating α and β sequentially. At the $(t+1)$-th iteration, the α-minimization step involves solving the following problem according to (7):

$$\begin{aligned}\min_\alpha \sum_i \left(\sum_j \alpha_{ij}\left(D_iP_j^E + \lambda_{ij}^t + \frac{\rho}{2}\left(\alpha_{ij} - 2\beta_{ij}^t\right)\right) - D_iU(\alpha_i)\right) \\ \text{s.t. } \sum_j \alpha_{ij} = 1, U(\alpha_i) = U\Big(\sum_j \alpha_{ij}L_{ij}\Big), \alpha_i \ge 0, \forall i,\end{aligned} \tag{12}$$

where α_i is the vector of α_{ij} for client i, and $U(\alpha_i)$ is a shorthand for i's utility function. This problem is *decomposable* over clients, since the objective function and constraints are separable over i. Effectively, each client needs to independently solve the following sub-problem:

$$\begin{aligned}\min_{\alpha_i} \sum_j \alpha_{ij}\left(D_iP_j^E + \lambda_{ij}^t + \frac{\rho}{2}\left(\alpha_{ij} - 2\beta_{ij}^t\right)\right) - D_iU(\alpha_i) \\ \text{s.t. } \sum_j \alpha_{ij} = 1, U(\alpha_i) = U\Big(\sum_j \alpha_{ij}L_{ij}\Big), \alpha_i \ge 0.\end{aligned} \tag{13}$$

The per-client sub-problem is of a much smaller scale, with $|\mathcal{J}|$ variables and $|\mathcal{J}|+1$ constraints, and can be efficiently solved by a standard optimization solver. As discussed in Sec. I, in reality the number of stub datacenters $|\mathcal{J}| = O(10^2)$ and is much smaller than the number of clients $|\mathcal{I}|$. Depending on the exact shape of the utility function, in some cases we can even provide analytical solution as we shall see in Sec. III-D.

We have solved the α-minimization step distributively across all clients by decomposing the problem (12) into $|\mathcal{I}|$ per-client sub-problems (13). After obtaining α^{t+1}, the β-minimization step can also be similarly attacked as we show now.

According to (8), the β-minimization step consists of solving the following:

$$\begin{aligned}\min_\beta \sum_j\sum_i \beta_{ij}\left(D_iP_j^B - \lambda_{ij}^t + \frac{\rho}{2}\left(\beta_{ij} - 2\alpha_{ij}^{t+1}\right)\right) \\ \text{s.t. } \sum_i \beta_{ij}D_i \le C_j, \forall j, \beta_{ij}\ge 0, \forall i,j.\end{aligned} \tag{14}$$

This problem is also *decomposable* over the set of stub datacenters $\mathcal{J}$ into $|\mathcal{J}|$ sub-problems. Specifically, each stub datacenter needs to solve

$$\begin{aligned}\min_{\beta_{1j},\beta_{2j},\ldots} \sum_i \beta_{ij}\left(D_iP_j^B - \lambda_{ij}^t + \frac{\rho}{2}\left(\beta_{ij} - 2\alpha_{ij}^{t+1}\right)\right) \\ \text{s.t. } \sum_i \beta_{ij}D_i \le C_j, \beta_{ij}\ge 0, \forall j.\end{aligned} \tag{15}$$

The per-stub datacenter problem is a quadratic program, whose solutions can be provided in analytical form as follows.

Lemma 1: At the $(t+1)$-th iteration, for all $i\in\mathcal{I}$ such that $\lambda_{ij}^t - D_iP_j^B + \rho\alpha_{ij}^{t+1} \le 0$, $\beta_{ij}^{t+1} = 0$. Denote the remaining set $\{i\in\mathcal{I} | \lambda_{ij}^t - D_iP_j^B + \rho\alpha_{ij}^{t+1} > 0\}$ as $\mathcal{I}_j^{t+1}$. Then β_{ij}^{t+1} for $i\in\mathcal{I}_j^{t+1}$ is: If $\sum_{i\in\mathcal{I}_j^{t+1}}(\lambda_{ij}^t - D_iP_j^B + \rho\alpha_{ij}^{t+1})D_i \le \rho C_j$,

$$\beta_{ij}^{t+1} = \frac{\lambda_{ij}^t - D_iP_j^B}{\rho} + \alpha_{ij}^{t+1},$$

otherwise,

$$\beta_{ij}^{t+1} = \max\left\{\frac{\lambda_{ij}^t - D_i(P_j^B + \nu_j^{t+1})}{\rho} + \alpha_{ij}^{t+1}, 0\right\},$$

where $\nu_j^{t+1} \geq 0$ is determined by the following

$$\sum_{i \in \mathcal{I}_j^{t+1}} \beta_{ij}^{t+1} D_i = C_j.$$

The proof can be found in Appendix A.

Having obtained the optimal α^{t+1} and β^{t+1}, the final step is to perform the dual variable update:

$$\lambda_{ij}^{t+1} = \lambda_{ij}^t + \rho(\alpha_{ij}^{t+1} - \beta_{ij}^{t+1}). \tag{16}$$

The entire procedure is summarized in Algorithm 1. Since the constraint set C_1 for α is clearly bounded in our problem (10), according to Theorem 1 the algorithm converges to the optimal solution.

Lemma 2: Our algorithm based on ADMM converges to the optimal solution α^* and β^* of (10) and equivalently (1).

Algorithm 1 *Optimal Distributed Solution for* (1)

1. Each stub datacenter j initializes $\beta_{ij}^0 = 0$, $\lambda_{ij}^0 = 0$, and broadcasts its electricity price P_j^E to each client.
2. Given $\beta_i^t = [\beta_{i1}^t, \beta_{i2}^t, \ldots]$ and $\lambda_i^t = [\lambda_{i1}^t, \lambda_{i2}^t, \ldots]$, each client i solves the per-client sub-problem (13), and sends the optimal solution α_{ij}^{t+1} to the corresponding stub datacenter j.
3. Given $\alpha_j^{t+1} = [\alpha_{1j}^{t+1}, \alpha_{2j}^{t+1}, \ldots]$, each stub datacenter solves the sub-problem (15) as in Lemma 1 with local information P_j^B and $\lambda_j^t = [\lambda_{1j}^t, \lambda_{2j}^t, \ldots]$.
4. Each stub datacenter j updates the dual variables $\lambda_j^t = [\lambda_{1j}^t, \lambda_{2j}^t, \ldots]$ as in (16). It then sends the optimal solution β_{ij}^{t+1} and updated dual variable λ_{ij}^{t+1} to the corresponding client i.
5. Return to step 2 until convergence.

Intuitively, the working of our algorithm follows a divide-and-conquer paradigm. Recall that α controls the net utility gain of processing the requests, while β determines the bandwidth cost of transmitting the response packets. Our algorithm first optimizes α for the mapping aspect of the problem given the response routing solution β^t. It then optimizes β for the response routing aspect of the problem given the previously computed mapping solution α^{t+1}. The dual update ensures the two sets of solutions converge to the same workload management solution, which is also optimal.

C. A Parallel Implementation in the Cloud

The distributed nature of Algorithm 1 allows for an efficient parallel implementation in the cloud that has abundant server resources. Here we discuss several issues pertaining to such an implementation in reality.

First, at each iteration, each client solves the per-client sub-problem in step 2. This can be readily implemented in a parallel fashion on each server of one of the datacenters the provider owns, which we call the *designated datacenter*. A production datacenter typically has $O(10^4)$–$O(10^5)$ servers [14]. Thus each server only needs to solve $O(10)$–$O(1)$ per-client sub-problems at each iteration. Since the per-client sub-problem (13) is a small-scale convex optimization, the computational complexity is low. A multi-threaded implementation can further speed up the algorithm on multi-core hardware. The penalty parameter ρ and utility function U can be configured across all servers before the algorithm starts off.

Similarly, step 3 of Algorithm 1, which solves the per-stub datacenter sub-problem, also has a parallel implementation in the designated datacenter. Only $|\mathcal{J}|$ servers are required, each responsible for solving one instance of (15) according to the solution in Lemma 1. It can even be implemented on the same servers that implement step 2 for the per-client sub-problems. The parallel implementation of our algorithm thus makes it well suited in the cloud environment.

Second, our algorithm can be terminated before convergence is reached. ADMM is not sensitive to step size ρ, and usually finds a solution with modest accuracy within tens of iterations [6]. As argued in Sec. I, a solution with modest accuracy is sufficient in situations of flash crowds of requests and failure recovery. A provider can apply an early-braking mechanism in these scenarios to terminate the algorithm after several tens of iterations without worrying about performance issues.

We finally comment that the message passing overhead of our algorithm is also low. As a prerequisite, the electricity and bandwidth prices of each datacenter and ISP needs to be gathered at the designated datacenter. The final output of the algorithm α_{ij} needs to be disseminated to the mapping nodes and datacenters (recall Fig. 1). All the other message passing, for exchanging α, β, and λ amongst servers, happens in the internal network of the designated datacenter, which in many cases is specifically designed to handle the broadcast and shuffle transmission patterns of HPC applications such as MapReduce [3]. Note that the amount of intermediate data our algorithm produces is much smaller than the bulky data of HPC applications [29]. Thus the message passing overhead incurred in the datacenter network is low.

D. Case Study: Affine Utility Functions

Before concluding this section, we provide a case study of the workload management problem with an affine utility function. Affine utility functions are the *de facto* utility function widely used in the literature [21], though some studies have argued for more complicated utility functions with fairness considerations [31].

An affine utility function has the following form:

$$U\Big(\sum_j \alpha_{ij} L_{ij}\Big) = -a \sum_j \alpha_{ij} L_{ij}, \tag{17}$$

where $a > 0$ is a conversion factor that translates user-perceived latency into utility (e.g., revenue). With an affine utility function, the per-client sub-problem (13) becomes a

quadratic program in the following form:

$$\min_{\alpha_i} \sum_j \alpha_{ij} \left(D_i \left(aL_{ij} + P_j^E \right) + \lambda_{ij}^t + \frac{\rho}{2} \left(\alpha_{ij} - 2\beta_{ij}^t \right) \right)$$
$$\text{s.t.} \sum_j \alpha_{ij} = 1, \alpha_i \geq 0. \quad (18)$$

Optimal solutions can then be derived in an analytical form through the KKT conditions.

Lemma 3: At the $(t+1)$-th iteration, the optimal solution of the per-client sub-problem (18) with an affine utility function for a given client i is as follows.

$$\alpha_{ij}^{t+1} = \max \left\{ \beta_{ij}^t - \frac{D_i(aL_{ij} + P_j^E) + \lambda_{ij}^t + \mu_i^{t+1}}{\rho}, 0 \right\},$$

where $\mu_i^{t+1} \neq 0$ is determined by the following

$$\sum_{j \in \mathcal{J}} \alpha_{ij}^{t+1} = 1.$$

The proof can be found in Appendix B. Essentially, this is a system of $|\mathcal{J}| + 1$ equations with $|\mathcal{J}| + 1$ variables, whose solution can be efficiently computed.

Thus, in the case of an affine utility function, the per-client sub-problem reduces to a quadratic program and is particularly easy to solve.

IV. Evaluation

To realistically evaluate the performance of our algorithm, we conduct trace-driven simulations in this section.

A. Setup

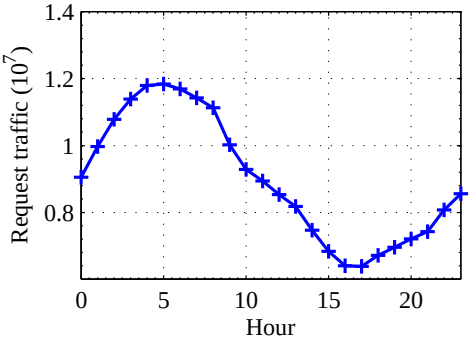

Fig. 2. Total request traffic of the Wikipedia traces [27].

We use the Wikipedia request traces [27] to represent the request traffic of a cloud service. The dataset we use contains, among other things, 10% of all user requests issued to Wikipedia from 3:56PM, January 1, 2008 GMT to 4:57PM, January 2, 2008 GMT. The prediction of workload can be done accurately as demonstrated by previous work [22], [23], and in the simulation we simply adopt the measured request traffic as the total demand. We assume the optimization is done hourly, and Fig. 2 plots the hourly request traffic of the traces for 24 hours of the measurement period.

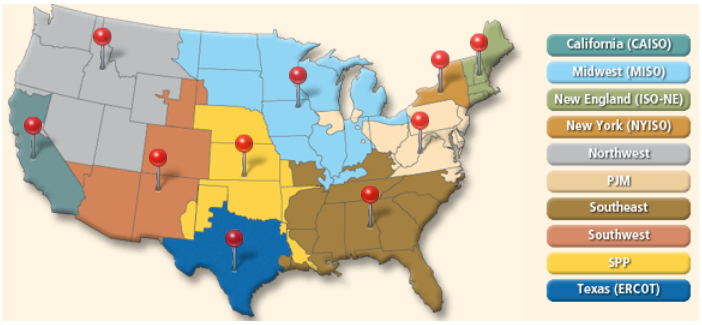

Fig. 3. The U.S. electricity market and our datacenter map. Source: [10].

We simulate a cloud that deploys ten datacenters across the continental U.S. According to the Federal Energy Regulatory Commission (FERC), the U.S. electricity market is consisted of multiple regional markets as shown in Fig. 3 [10]. Each regional market has several hubs with their own pricing. Thus for the ease of exploration, we assume that one datacenter is deployed in a randomly chosen hub in each of the ten regional markets as shown in Fig. 3. We use the 2011 annual average day-ahead on peak price as the electricity price for each datacenter, i.e. P_E, as summarized in Table I. In the simulations we calculate the cost by assuming that one request consumes 10W of energy on average, including the server, network, and cooling energy consumption.

TABLE I
2011 annual average day ahead on peak price ($/MWh) in different regional markets. Source: [10].

Region	Hub	Price
California	NP15	$35.83
Midwest	Michigan Hub	$42.73
New England	Mass Hub	$52.64
New York	NY Zone J	$62.71
Northwest	California-Oregon Border (COB)	$32.57
PJM	PJM West	$51.99
Southeast	VACAR	$44.44
Southwest	Four Corners	$36.36
SPP	SPP North	$36.41
Texas	ERCOT North	$61.55

TABLE II
Tiered bandwidth prices. Source: Amazon EC2

Link capacity (requests/hour)	Pricing ($/request)
$< 1.4 \times 10^5$	0.0012
1.4×10^5–5.6×10^5	0.0009
5.6×10^5–1.4×10^6	0.0007
$> 1.4 \times 10^6$	0.0005

Each datacenter has 3 ISP links. Thus the number of stub datacenters $|\mathcal{J}| = 30$. The prices of the ISP links are estimated in two steps. First, the capacity of each ISP link is randomly set such that the total capacity across the 30 links is 1.2×10^7 requests per hour. Then, the price of an ISP link is determined from a tiered structure based on the link capacity, where a link with larger capacity has a lower cost. We assume a request's response packets contain 1 MB of data on average, and use Amazon EC2 bandwidth prices in the U.S. east region to determine the exact price per request presented in Table II.

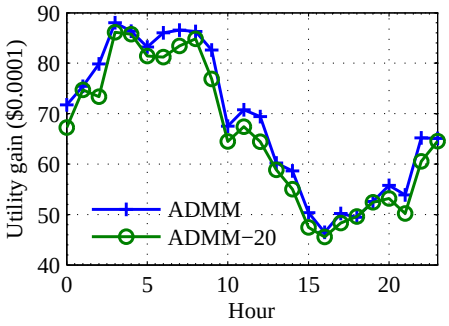

Fig. 4. Optimal average utility gain.

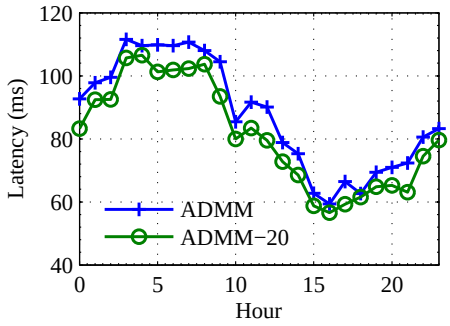

Fig. 5. Optimal average latency performance.

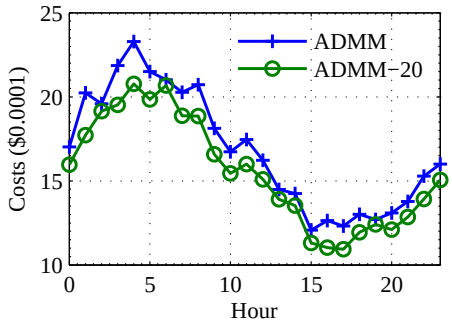

Fig. 6. Optimal average costs per request.

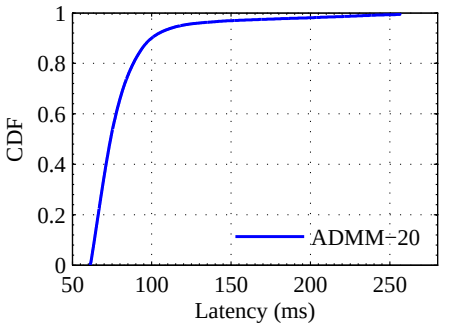

Fig. 7. CDF of per request latency.

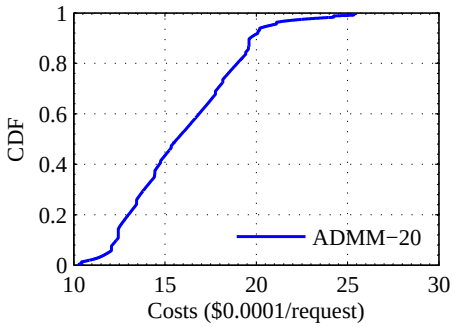

Fig. 8. CDF of per request costs.

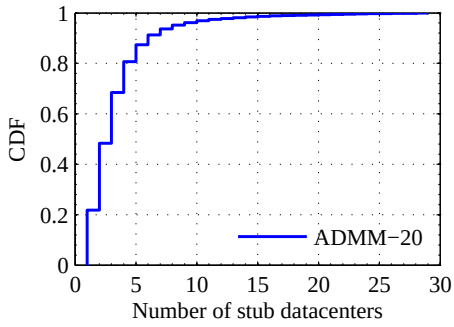

Fig. 9. CDF of number of stub datacenters per client.

This setup resembles the volume discount strategy commonly used in the industry.

We rely on iPlane [19], a system that collects wide-area network statistics from Planetlab vantage points, to obtain the latency information. We set the number of clients $|\mathcal{I}| = 10^5$, and choose 10^5 IP prefixes from a RouteViews [1] dump. We then extract the corresponding round trip latency information from the iPlane logs, which contain traceroutes made to a large number of IP addresses from Planetlab nodes. We only use latency measurements from Planetlab nodes that are close to our datacenter locations. Therefore the propagation latency depends on the datacenter location but not on the specific ISP link used. We believe this is a reasonable approximation when the geographical distance instead of link condition dominates the propagation delay.

Now since the Wikipedia traces do not contain any client information, to emulate the geographical distribution of requests, we split the total request traffic among the clients following a normal distribution. The utility function is the simple affine function as in (17) with $a = 10^{-4}$. That is, a request with 100 ms latency translates to \$0.01 revenue for the provider. Finally, the penalty parameter ρ is set to 1 in all our simulations.

B. Performance

We evaluate two variants of our algorithm in the simulations. The first variant, referred to as ADMM in the figures, runs Algorithm 1 until convergence is reached. The second variant, referred to as ADMM-20, applies an early-braking method and runs Algorithm 1 for only 20 iterations. Fig. 4 plots the average utility gain per request for the two variants. Throughout the day, we observe that ADMM-20 with 20 iterations can achieve utility gains close to optimum within \$0.0008 difference, while the regular ADMM converges within 56 iterations in all the cases (more on convergence in Sec. IV-C). The average value of $|\alpha_{ij} - \beta_{ij}|$ after 20 iterations is merely 2.7133×10^{-5}. Therefore, our algorithm converges quickly to near optimum.

Fig. 5–6 further plot the average latency and serving costs per request. Observe that the average client latency stands below 80 ms most of the time, and never exceeds 120 ms. The average serving costs is approximately \$0.0015 per request throughout the day. Both metrics fluctuate closely with the total traffic as shown in Fig. 2.

To understand the performance of our algorithm on a microscopic level, we plot the CDF of the request latency and serving costs across all clients and all hours for the ADMM-20 variant in Fig. 7 and 8. Most of the requests, more than 90%, are served with latency less than 100 ms. The CDF of costs is more skewed, implying that the per-request costs vary significantly across clients. This is because the (bandwidth) cost difference amongst the ISP links of the same datacenter is clearly larger than the latency difference, which is assumed to be zero.

One may wonder at this point that, our algorithm may direct requests only to the best stub datacenter for a client, which is not preferable for diversity and resilience purposes. However, Fig. 9 shows that this is not the case. We plot the CDF of the number of stub datacenters a client's requests are directed to (for hour 0 data and the ADMM-20 variant). The figure shows that for more than 80% of the clients, the requests are directed to 2-5 stub datacenters. On average, each client has 3.6 stub datacenters to serve its requests. This leads us to believe that our algorithm distributes the workload in a balanced way.

C. Convergence

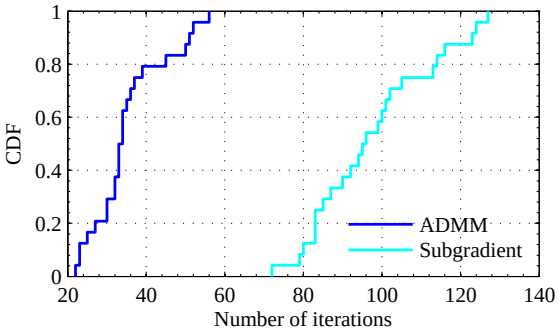

Fig. 10. CDF of the number of iterations to achieve convergence for our ADMM algorithm and the subgradient method.

We now investigate the convergence and running time of our algorithm. For comparison, we use the subgradient method [5] to solve the dual problem of the transformed optimization (10) with the augmented Lagrangian (11). Specifically, the primal variables α and β are jointly optimized instead of sequentially updated as in our ADMM algorithm to speed up the convergence, and the dual variables λ are updated by the subgradient method. The step size has to be carefully chosen, since a too large value will make the final output far away from the real optimum, and a too small value will slow down the convergence. We choose the step sizes according to the diminishing step size rule [5].

Fig. 10 plots the CDF of the number of iterations the two algorithms take to achieve convergence for the 24 runs on the traces. We can clearly see that our ADMM algorithm converges much faster than the subgradient method. Our algorithm takes at most 56 iterations to converge, while the subgradient method takes at least 72 iterations. For 80% of the time our algorithm converges within 40 iterations, while the subgradient method takes 110 iterations. This demonstrates the fast convergence of our algorithm compared to conventional methods.

We finally study the running time of our algorithm. Note that since we do not have enough hardware resources to experiment with a parallel implementation, our algorithm is implemented on a single server machine where each per-client and per-stub datacenter sub-problem is sequentially solved. We observe that one iteration takes on average 1500.6447 seconds on a Dual Dual-Core Intel Xeon 3.0 Ghz (64-bit) server. Since $|\mathcal{I}| = 10^5$ and $|\mathcal{J}| = 30$, solving each sub-problem takes around 0.015 second. Thus, a parallel implementation on 1000 servers will take less than a second to run one iteration, which demonstrates the efficiency of our algorithm for large-scale problems.

V. Related Work

The topics of request mapping and response routing for a geo-distributed infrastructure are usually treated separately in the literature. On the former, [26] introduced the idea of utilizing the location diversity of electricity price to intelligently direct requests to datacenters with lower prices. [30] developed a decentralized request mapping algorithm with configurable policies. [12], [18] considered the effect of request mapping on providing environmental gains by using green energy. [31] the performance fairness issue in request mapping. On the latter, [13] developed routing algorithms to optimize performance and cost for a multi-homing ISP. [32] proposed to optimize traffic engineering across all upstream ISPs, assuming requests are simply mapped to the closest ingress point.

The joint study of mapping and routing has started to gain attention recently. [2] studied the data placement problem in a geo-distributed cloud, considering the data locality, bandwidth costs, and storage capacity. We assume the content is replicated on all datacenters. [21] considered the joint problem with bandwidth costs, and the resulting linear program was solved by standard methods. We consider both bandwidth and electricity costs, and develop a new distributed algorithm to solve the convex optimization problem.

VI. Concluding Remarks

We studied the joint request mapping and response routing problem for geographically distributed datacenters. We formulated the problem as a general convex optimization, where the location diversity of performance and costs are modeled. We developed an efficient distributed algorithm based on ADMM to decompose the large-scale global problem into many sub-problems, each of which can be quickly solved. We discussed a parallel implementation of the algorithm that is well suited in a cloud environment with abundant server resources. Trace-driven simulations are conducted to evaluate the algorithm's performance. As future work, we plan to more thoroughly study its impact on existing wide-area traffic engineering schemes.

References

[1] http://www.routeviews.org.

[2] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *Proc. USENIX NSDI*, 2010.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. ACM SIGCOMM*, 2008.

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

[5] 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.

[6] 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.

[7] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[8] Z. Cao, Z. Wang, and E. Zegura. Performance of hashing-based schemes for Internet load balancing. In *Proc. IEEE INFOCOM*, 2000.

[9] 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.

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

[11] 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.

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

[13] D. K. Goldenberg, L. Qiu, H. Xie, Y. R. Yang, and Y. Zhang. Optimizing cost and performance for multihoming. In *Proc. ACM SIGCOMM*, 2004.
[14] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, 2009.
[15] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, 1969.
[16] 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.
[17] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao. Moving beyond end-to-end path information to optimize CDN performance. In *Proc. ACM IMC*, 2009.
[18] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew. Greening geographical load balancing. In *Proc. ACM Sigmetrics*, 2011.
[19] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An information plane for distributed services. In *Proc. USENIX OSDI*, 2006.
[20] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Trans. Netw.*, 8(5):556–567, October 2000.
[21] S. Narayana, J. W. Jiang, J. Rexford, and M. Chiang. To coordinate or not to coordinate? Wide-Area traffic management for data centers. Technical report, Princeton University, 2012.
[22] D. Niu, H. Xu, B. Li, and S. Zhao. Risk management for video-on-demand servers leveraging demand forecast. In *Proc. ACM Multimedia*, 2011.
[23] D. Niu, H. Xu, B. Li, and S. Zhao. Quality-assured cloud bandwidth auto-scaling for video-on-demand applications. In *Proc. IEEE INFOCOM*, 2012.
[24] 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.
[25] K. Papagiannaki, N. Taft, Z.-L. Zhang, and C. Diot. Long-term forecasting of Internet backbone traffic: Observations and initial models. In *Proc. IEEE INFOCOM*, 2003.
[26] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the electricity bill for Internet-scale systems. In *Proc. ACM SIGCOMM*, 2009.
[27] G. Urdaneta, G. Pierre, and M. van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009.
[28] V. Valancius, C. Lumezanu, N. Feamster, R. Johari, and V. V. Vazirani. How many tiers? Pricing in the Internet transit market. In *Proc. ACM SIGCOMM*, 2011.
[29] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *Proc. ACM SIGCOMM*, 2009.
[30] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. DONAR: Decentralized server selection for cloud services. In *Proc. ACM SIGCOMM*, 2010.
[31] H. Xu and B. Li. A general and practical datacenter selection framework for cloud services. In *Proc. IEEE CLOUD*, 2012.
[32] 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.
[33] Y. Zhu, B. Helsley, J. Rexford, A. Siganporia, and S. Srinivasan. LatLong: Diagnosing wide-area latency changes for CDNs. *IEEE Trans. Netw. Service Manag.*, to appear, 2012.

APPENDIX A
PROOF OF LEMMA 1

The KKT conditions [7] of the per-stub datacenter problem (15) constitute the following system of equations.

$$\rho(\beta_{ij}^{t+1}-\alpha_{ij}^{t+1})+D_i(P_j^B+\nu_j^{t+1})-\lambda_{ij}^t-\tau_{ij}^{t+1}=0,\forall i, \quad (19)$$

$$\nu_j^{t+1}\Big(C_j-\sum_i \beta_{ij}^{t+1}D_i\Big)=0, \beta_{ij}^{t+1}\tau_{ij}^{t+1}=0,\forall i \quad (20)$$

$$C_j-\sum_i \beta_{ij}^{t+1}D_i\geq 0, \nu_j\geq 0, \beta_{ij}^{t+1}\geq 0, \tau_{ij}^{t+1}\geq 0, \forall i. \quad (21)$$

β_{ij}^{t+1} is the optimal solution, and ν_j^{t+1} is the KKT multiplier. (19) is the first-order optimality conditions, (20) is the complementary slackness condition, and (21) are the primal and dual feasibility conditions.

For all $i\in\mathcal{I}$ that satisfy $\lambda_{ij}^t-D_iP_j^B+\rho\alpha_{ij}^{t+1}\leq 0$, assume $\beta_{ij}^{t+1}>0$. Then according to the complementary slackness condition (20) $\tau_{ij}^{t+1}=0$. The left hand side (LHS) of (19) is always positive, which contradicts the optimality condition. Thus $\beta_{ij}^{t+1}=0$.

As in Lemma 1, denote the rest of stub datacenters as the set $\mathcal{I}_j^{t+1}$. $\lambda_{ij}^t-D_iP_j^B+\rho\alpha_{ij}^{t+1}>0$ holds for all $i\in\mathcal{I}_j^{t+1}$. If $\sum_{i\in\mathcal{I}_j^{t+1}}(\lambda_{ij}^t-D_iP_j^B+\rho\alpha_{ij}^{t+1})D_i\leq\rho C_j$, then according to (20) $\nu_j^{t+1}=0$. This is so because for those $i\in\mathcal{I}_j^{t+1}$ such that $\beta_{ij}^{t+1}>0$, $\rho\beta_{ij}^{t+1}\leq\lambda_{ij}^t-D_iP_j^B+\rho\alpha_{ij}^{t+1}$ since $\nu_j^{t+1}\geq 0$ in (19). Thus $\sum_{i\in\mathcal{I}_j^{t+1}}\beta_{ij}^{t+1}D_i\leq C_j$, and $\nu_j^{t+1}=0$. Then, $\tau_{ij}^{t+1}=0$ must hold for all $i\in\mathcal{I}_j^{t+1}$, for otherwise $\beta_{ij}^{t+1}=0$ and the LHS of (20) is always negative. Substituting $\nu_j^{t+1}=0$ and $\tau_{ij}^{t+1}=0$ into (19) yields $\beta_{ij}^{t+1}=\frac{\lambda_{ij}^t-D_iP_j^B}{\rho}+\alpha_{ij}^{t+1}$.

If $\sum_{i\in\mathcal{I}_j^{t+1}}(\lambda_{ij}^t-D_iP_j^B+\rho\alpha_{ij}^{t+1})D_i>\rho C_j$, note that the objective of (15) is minimized at $\frac{\lambda_{ij}^t-D_i(P_j^B+\nu_j^{t+1})}{\rho}+\alpha_{ij}^{t+1}>0$ when the capacity constraint is absent, we must have $\beta_{ij}^{t+1}<\frac{\lambda_{ij}^t-D_i(P_j^B+\nu_j^{t+1})}{\rho}+\alpha_{ij}^{t+1}$ to conform to the capacity constraint. Since the objective function of (15) is convex in β_{ij}, for $\beta_{ij}\in\left[0,\frac{\lambda_{ij}^t-D_i(P_j^B+\nu_j^{t+1})}{\rho}+\alpha_{ij}^{t+1}\right]$ it is increasing. Thus the optimal β_{ij}^{t+1} must satisfy the capacity constraint at equality, and equal to $\max\left\{\frac{\lambda_{ij}^t-D_i(P_j^B+\nu_j^{t+1})}{\rho}+\alpha_{ij}^{t+1},0\right\}$.

APPENDIX B
PROOF OF LEMMA 3

The KKT conditions for the per-client sub-problem with an affine utility function (18) are

$$\rho(\alpha_{ij}^{t+1}-\beta_{ij}^t)+D_i(aL_{ij}+P_j^E)+\lambda_{ij}^t +\mu_i^{t+1}-\sigma_{ij}^{t+1}=0,\forall j, \quad (22)$$

$$\sum_j \alpha_{ij}^{t+1}-1=0, \quad (23)$$

$$\mu_i^{t+1}\neq 0, \sigma_{ij}^{t+1}\alpha_{ij}^{t+1}=0, \alpha_{ij}^{t+1}\geq 0, \sigma_{ij}^{t+1}\geq 0, \forall j, \quad (24)$$

where α_{ij}^{t+1} is the optimal solution as in (7), and μ_i^{t+1} and σ_{ij}^{t+1} are the KKT multiplier for the equality and inequality constraints of (18), respectively. (22) corresponds to the first-order optimality condition, (23) is one of the primal feasibility conditions, and (24) captures the other primal feasibility condition, the dual feasibility, and the complementary slackness conditions. Essentially, since α_{ij}^{t+1} and σ_{ij}^{t+1} never appear at the same time in (22), $\alpha_{ij}^{t+1}=\max\left\{\beta_{ij}^t-\left(D_i\left(aL_{ij}+P_j^E\right)+\lambda_{ij}^t+\mu_i^{t+1}\right)/\rho,0\right\}$, and must satisfy (23). Thus the proof.