An Asynchronous Fixed-Point Algorithm for Resource Sharing with Coupled Objectives

Di Niu, *Member, IEEE,* and Baochun Li, *Fellow, IEEE,*

***Abstract*—Distributed resource allocation and sharing can often be formulated as a utility maximization problem, with the objective being the sum of user utilities minus a coupled cost. A traditional distributed solution to such problems, called "consistency pricing", decouples the objective function via dual decomposition, which is then iteratively solved by the subgradient method. However, such gradient-based approaches may require many iterations of message passing to converge, which may not be sufficient in large-scale real-time applications. In this paper, we propose a new fixed-point-like distributed solution to resource sharing problems with coupled objective functions. While preserving the simple pricing interpretation, our approach speeds up convergence by exploiting the structural difference between user utilities and the coupled cost function. We theoretically analyze the asynchronous algorithm convergence conditions based on contraction mapping. Through a detailed case study of cloud bandwidth reservation based on real-world workload traces, we demonstrate the benefits of the proposed algorithm over state-of-the-art distributed optimization techniques including gradient descent, dual decomposition and ADMM. In addition, we also extend the proposed algorithm to approach a more general class of consensus optimization problems with not only a coupled objective function, but also a certain class of coupled constraints.**

***Index Terms*—Network utility maximization, coupled objective, resource allocation, distributed algorithm, decomposition, pricing, network control, sharing, consensus optimization, fixed-point iteration, asynchronous convergence.**

I. Introduction

Traditional network utility maximization (NUM) problems, including TCP congestion control models [1], [2] and wireless cross-layer designs [3], [4], generally aim to maximize uncoupled utilities [4], [5] in an additive form of $\sum_i U_i(x_i)$, where each utility function U_i depends only on its local decision variable x_i. Under this assumption, distributed solutions can be derived, using standard primal or dual decomposition to decouple resource sharing constraints [4], [5]. For example, dual decomposition [5] can be used to decouple an additive resource constraint in the form of $\sum_i h_i(x_i) \leq C$, such that each entity solves a local subproblem to maximize $U_i(x_i) - \lambda h_i(x_i)$, where λ is a dual variable (or price) that is associated with the resource constraint and is to be updated in a master problem iteratively.

However, many modern distributed resource allocation problems with sharing or competition need be modelled as utility maximization problems with a *coupled objective function* [4], [5], where the utility U_i of each entity may depend not only on its local variable x_i but also on the variables of other entities. In many applications, it is even more suitable to model sharing or competition with a coupled objective function rather than with hard additive resource constraints.

For example, in a multiparty conference operated by Skype, a cluster of centralized servers is used to relay video traffic among end users [6]. If we increase the video rate of each user, its content will be streamed to other users at a higher throughput, whereas in the meantime all users may suffer from a higher latency due to increased processing and queueing delays at the servers due to more traffic. Note that in this case, there is generally no hard constraint on the sum of user sending rates, since Skype video bit rates mostly range from 5 kbps up to only 1200 kbps [6] and the server bandwidth is over-provisioned. As another example, Linux containerization [7] is an emerging way of sharing parts of a single operating system among multiple isolated applications, as opposed to VM-based virtualization which supports multiple apps with their own OS on top of a single hypervisor. Google is launching on average about $3,300$ containers per second running discrete small jobs of their massive distributed applications in short order [8]. On a cluster of interconnected physical servers, there is a tradeoff between running as many containerized jobs as possible on each server to increase its utility and the job service delay caused by interference between jobs. Although one can specify on which core each container runs as well as its share of CPU time, memory and block I/O access, performance interference still exists due to resources shared on-chip, including inter-server connection networks, cache space, and memory bandwidth [9]. When interference is severe, all containerized jobs will suffer from longer delays and such delays are intercorrelated. A further example is wireless power control and DSL spectrum management of copper wires in a cable binder [5], where a user's utility is not only determined by its own transmit power but also by the signal-to-interference ratios that depend on the transmit powers of other users.

Nonetheless, in the NUM and pricing literature, few distributed solutions have been developed for utility maximization problems where competition is modeled through a coupled objective function. The only existing approach is the so-called "consistency pricing" [10], which seeks decomposition from the dual problem by introducing auxiliary variables and then solves the dual problem with the subgradient method. However, similar to gradient descent, the subgradient method usually requires to update variables for a large number of iterations before convergence. Given message-passing delays be-

D. Niu is with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta, T6G 2V4, Canada. E-mail: dniu@ualberta.ca.

B. Li is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, M5S 3G4, Canada. E-mail: bli@ece.toronto.edu.

This work is partially supported by the NSERC Discovery and NSERC Strategic Networks Grants held by the co-authors.

tween entities in practical large-scale systems, more iterations will slow down the entire optimization process. On the other hand, although faster centralized algorithms such as interior-point methods or Newton methods need fewer iterations to converge, they use more than first-order information and are thus harder to be decomposed and executed in a distributed manner.

In this paper, we propose a new distributed solution to resource sharing problems with coupling in the objective function. Our goal is to achieve convergence in only a few number of iterations to reduce the impact of iterative message passing delays and scale up to large systems. We begin with studying the classical *sharing* problem [11], that is to maximize $\sum_{i=1}^{n} U_i(x_i) - C(x_1, \ldots, x_n)$, i.e., the sum of utilities, each depending on a local variable x_i, minus a cost that is coupled among all variables $x_1, \ldots, x_n$. Our algorithm proceeds in the following iterative steps. First, given all the local decision variables x_i (e.g., resource requests) from users, the resource provider sets a price $p_i = \partial C(x_1, \ldots, x_n)/\partial x_i$ for each user i under all current resource requests. Then, each user updates its resource request x_i under the new price p_i by maximizing its own benefit $U_i(x_i) - p_i x_i$ and submits the updated x_i to the resource provider which in turn updates the prices. The above process is repeated until the algorithm converges. This conceptually simple method could be viewed as a fixed-point iteration in search for a resource allocation vector $(x_1^*, \ldots, x_n^*)$ such that $U_i'(x_i) = \partial C(x_1, \ldots, x_n)/\partial x_i$ at $(x_1^*, \ldots, x_n^*)$ for all i, i.e., to equalize *the marginal cost* and *marginal utility* for every user.

The new approach differs from consistency pricing in several aspects. *First*, consistency pricing updates the price vector $p = (p_1, \ldots, p_n)$ based not only on the x_i that maximizes each user's benefit, but also on another resource allocation vector $y = (y_1, \ldots, y_n)$ that maximizes the "provider benefit" $p^T y - C(y)$ [5], [10]. It is this subtle difference that enables the proposed algorithm to converge faster under certain conditions. In fact, our algorithm exploits the structural differences between each U_i and C in terms of second-order properties to achieve a "spiral" convergence effect instead of moving in the direction of gradient descent. *Second*, while in consistency pricing, we need to carefully choose the stepsize, our algorithm only has an inertia parameter between 0 and 1 to weigh between the old price from the last iteration and the new price. This inertia parameter is easier to set and achieves robustness. *Third*, our algorithm is a contraction mapping under certain conditions and thus achieves convergence even if local algorithms and pricing are carried out *asynchronously* for each entity. While enjoying the advantages mentioned above, the new approach is also a simple pricing-based solution, since the message passing only involves first-order information. In contrast, other distributed algorithms including Alternating Direction Method of Multipliers (ADMM) [12] rely on second-order terms that cannot be explained via pricing and thus have to make further assumptions on the cooperativeness of each selfish entity as well as its computational capacity.

Our technical contributions in this paper are manifold. *First,* we provide theoretical *asynchronous* convergence conditions of the proposed approach based on contraction mapping, and explain why the contraction conditions can be satisfied for many utility and cost functions in practical applications. *Second,* we extend the proposed method to solve a more general form of distributed optimization problems, referred to as consensus optimization [12], with additive coupled objective functions $\sum_i F_i(x_1, \ldots, x_n)$ as well as a certain class of coupled constraints. *Third,* through a case study of cloud bandwidth reservation based on real-world bandwidth demand traces, we evaluate the benefits of the new approach as compared to consistency pricing as well as ADMM and gradient descent directly applied to the primal problem (the latter two can not be interpreted as pricing and have to assume cooperativeness of users), in terms of convergence iterations, computational complexity, running time and the final achieved objective value. We show that by better exploiting the inherent problem structures, the proposed approach requires both fewer iterations to converge and lower computational complexity at each node in each iteration, while achieving better optimization accuracy, as compared to other state-of-the-art distributed algorithms.

The remainder of the paper is organized as follows. We present related work in Sec. II. Sec. III describes the classical sharing problem with an application example. In Sec. IV, we propose our fixed-point-like algorithms for the sharing problem in comparison to a few existing algorithms. In Sec. V, we rigorously analyze algorithm convergence conditions in an asynchronous sense. In Sec. VI, we extend the proposed approach to solve problems with arbitrary coupling in the objective function and a certain class of coupled constraints. In Sec. VII, we evaluate the benefits of the new approach over consistency pricing, gradient descent for the primal problem, and ADMM, through a case study of cloud bandwidth reservation based on a large amount of real-world traces. We conclude the paper in Sec. VIII.

II. Related Work

Distributed resource allocation is often modeled as network utility maximization (NUM) problems. A number of primal/dual decomposition methods have been proposed to solve such problems in a distributed way, with surveys provided in [4], [5]. However, most NUM problems considered in the literature, including the original TCP congestion control work by Kelly [1] and Low [2], and the cross-layer optimization in wireless networks considered by Lin *et al.* [3], are only concerned with *uncoupled* utilities, where the local variables of one entity do not affect the utility functions of other variables, which does not apply to systems with competition or cooperation, where utilities are often coupled.

There is one existing distributed solution [5], [10] to NUM with coupled objective functions, which seeks decomposition from Lagrangian dual problems by introducing auxiliary variables and solves the dual problems using (sub)gradient methods. This approach has a simple economic interpretation of *consistency pricing* [5], [10]. However, gradient methods may require many iterations of message-passing between entities before convergence, which incurs delay in a distributed setting, as the participating entities increase. Furthermore, existing

faster numerical convex problem solvers, e.g., *Newton methods* [11], can hardly be applied in a distributed way, as they involve a large amount of message-passing beyond first-oder information, which is hard to implement and justify physically in reality.

The *nonlinear Jacobi algorithm* [11] solves convex problems by optimizing the objective function for each variable in parallel while holding other variables unchanged, and converges under certain contraction conditions. However, it cannot be applied to our problem, as no network element has global information of all the utilities. However, Jacobi algorithm turns out to be a limiting case of Algorithm 1 executed asynchronously. And Algorithm 1 is essentially a distributed version of Jacobi algorithm. ADMM is a distributed optimization solver that has been applied to both statistical learning problems [13] and distributed networking systems [14], [15]. However, ADMM uses augmented Lagrangian, which is not a first-order term, and thus cannot be interpreted naturally through pricing for selfish users.

Pricing in distributed systems has been approached with distributed optimization and game theory [16], [17]. In many network applications, a distributed solution through pricing is required or desirable because it applies to selfish (benefit-maximizing) users without requiring them to be cooperative. Not only can the proposed algorithms be interpreted as simple pricing-based distributed solutions, under certain conditions, they can also achieve fast convergence within only a few iterations.

III. The Classical Sharing Problem

We start with considering a classical resource allocation problem called the *sharing* problem:

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n U_i(x_i) - C(x) \\ \text{subject to} & x_i \in [a_i, b_i], \quad i = 1, \ldots, n, \end{array} \tag{1}$$

for some real numbers a_i, b_i, where $x = (x_1, \ldots, x_n)$ represents resource allocation, each U_i is a concave and monotonically increasing utility function, and $C(x)$ is a convex cost function that is monotonically increasing in each x_i. Assume C and U_i are twice continuously differentiable.

The sharing problem (1) often arises in networked systems, where each user i is associated with an utility $U_i(x_i)$ by using x_i units of some resource, while $C(x)$ can be understood as the total cost at the resource provider or a common performance penalty, which is coupled among all x_i in an arbitrary way. Note that the general case where not all users are coupled in the same cost function will be discussed in Sec. VI. The objective is to make resource allocation decision x such that the social welfare $\sum_i U_i(x_i) - C(x)$ is maximized. Apparently, granting the maximum resources to users will maximize the sum of their utilities, but will also incur a high cost $C(x)$. As convex optimization, problem (1) has a unique solution $x^* = (x_1^*, \ldots, x_n^*)$.

It is undesirable to solve (1) in a centralized way using conventional algorithms such as interior point methods or Newton's methods [18], because each U_i may not be known to the resource provider or other users and the cost C may not be known to the users. However, due to the coupled objective function, the primal problem cannot be decomposed easily.

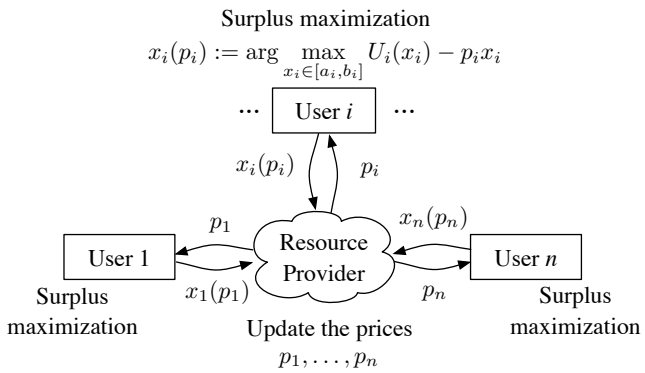


Fig. 1. A pricing interpretation of Algorithm 1 and consistency pricing.

A. Consistency Pricing and the Subgradient Method

We briefly review the existing distributed solution to the sharing problem (1), called *consistency pricing* [10], based on Lagrangian dual decomposition and subgradient methods. By introducing an auxiliary variable y, problem (1) is equivalent to

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n U_i(x_i) - C(y) \\ \text{subject to} & x = y, \quad x_i \in [a_i, b_i], \quad i = 1, \ldots, n. \end{array} \tag{2}$$

The Lagrangian for problem (2) is

$$L(x, y, p) = \sum_{i=1}^n U_i(x_i) - C(y) + p^{\mathsf{T}}(y - x)$$

and the dual function is

$$q(p) = \sum_{i=1}^n \sup_{x_i \in [a_i, b_i]} \{U_i(x_i) - p_i x_i\} + \sup_y \{p^{\mathsf{T}} y - C(y)\},$$

where p is the Lagrange multiplier or dual variable. The dual problem of problem (2) is

$$\text{minimize} \quad q(p), \tag{3}$$

with $p \in \mathbb{R}^n$. Since problem (2) is convex, the duality gap is zero: if p^* solves the dual problem, then $x_i^* = \arg\max_{x_i \in [a_i, b_i]} U_i(x_i) - p_i^* x_i$ solves the original problem.

Thus, we only need to solve the dual problem (3). It is known [4], [11], [18] that the subgradient of $q(p)$ is

$$\nabla q(p) = y(p) - x(p), \tag{4}$$

where $y(p) = (y_1(p), \ldots, y_n(p))$ is given by

$$y(p) = \arg\max_y p^{\mathsf{T}} y - C(y), \tag{5}$$

and $x(p) = (x_1(p_1), \ldots, x_n(p_n))$ is given by

$$x_i(p_i) = \arg\max_{x_i \in [a_i, b_i]} U_i(x_i) - p_i x_i, \quad \forall i. \tag{6}$$

Applying the subgradient algorithm to the dual problem (3) yields the following dual variable update rule:

$$p_i := p_i - \gamma\big(y_i(p) - x_i(p_i)\big), \tag{7}$$

where γ is a sufficiently small positive *stepsize*.

Obviously, iterations (7) can be interpreted as a pricing-based distributed solution: given a price vector p set by the

resource provider, each user returns the resource request $x_i(p_i)$ to the provider by maximizing its benefit $U_i(x_i) - p_i x_i$. The provider then computes $y(p)$ locally by maximizing its own profit $p^\mathsf{T} y - C(y)$ and updates price p according to (7). The new p_i will be passed back to each user i to compute a new resource request $x_i(p_i)$. Apparently, the entire algorithm is decentralized in that the provider updates the price vector only based on the current price p, all the returned resource requests $x_1(p_1), \ldots, x_n(p_n)$, and the local cost function C at the provider, while each user computes its resource request $x_i(p_i)$ only based on its local price p_i and its local utility function U_i. The decentralized workflow of the consistency pricing approach is illustrated in Fig. 1. The resource provider will update the prices p iteratively, such that $x(p) = (x_1(p_1), \ldots, x_n(p_n))$ produced by the benefit-maximizing users will converge to the optimal resource allocation x^*.

B. Cloud Bandwidth Reservation as an Application

A typical application of problem (1) is a large-scale real-time cloud network reservation problem described in [19]. Suppose that n video providers, which have random bandwidth demands $D_1, \ldots, D_n$ from their end users, rely on a cloud provider to run their services (e.g., Netflix relies on Amazon AWS). These video providers want to reserve the egress network bandwidth from a datacenter or CDN edge servers operated by a cloud provider to guarantee smooth delivery their videos. The reservation of egress network bandwidth from data centers and server clusters can be enabled by the detailed engineering technology recently developed in [20]–[22].

Since the bandwidth demand D_i is random, it is not convenient to decide how much absolute bandwidth to reserve for each video provider. Thus, for each video provider i, the cloud can guarantee x_i portion of the bandwidth demand D_i coming from end-user requests in video channel i, while the rest of the demand $(1 - x_i)D_i$ will be served with best efforts. We call x_i the guaranteed portion of video provider i. The problem is to decide all the guaranteed portions $x = (x_1, \ldots, x_n)$ to maximize the expected *social welfare*, which is the total utility gained in all the video channels minus the cost of jointly reserving bandwidth for them at the cloud. Such a social welfare can also be understood as the expected profit of the cloud provider if the utility in each video channel directly contributes to the revenue of the cloud provider.

If x_i portion of its demand D_i is guaranteed, video provider i will have a utility $u_i(x_i, D_i)$, e.g., in the form of

$$u_i(x_i, D_i) = w_{i1} x_i D_i - w_{i2} \mathrm{e}^{B_i(D_i - x_i D_i)},$$

which represents a linear utility gain from the guaranteed demand $x_i D_i$ and a utility (revenue) loss $\mathrm{e}^{B_i(1-x_i)D_i}$ for the part of end-user requests $(1-x_i)D_i$ that is not guaranteed for service, where B_i is a video-channel-specific parameter. We model each video channel's utility gain as a linear function of the guaranteed end-user requests $x_i D_i$, by assuming every guaranteed unit bandwidth will generate a revenue of w_{i1} in video channel i. On the other hand, we model the utility loss due to the demand $(1 - x_i)D_i$ not guaranteed as a convex function $w_{i2}\mathrm{e}^{B_i(1-x_i)D_i}$ of x_i to model the exponentially increasing reputation loss (thus revenue loss) in video channel i as more of its end-user requests are not guaranteed for service.

Then, the cloud provider exploits multiplexing and reserves a total amount of absolute bandwidth K to satisfy all the guaranteed demand $x_1 D_1, \ldots, x_n D_n$ in all the channels with high probability. Given a small service violation probability ϵ, K should take the value such that the total guaranteed demand $\sum_i x_i D_i$ satisfies

$$\Pr\left(\textstyle\sum_i x_i D_i > K\right) = \epsilon. \tag{8}$$

Apparently, if the statistics of D_i are known, K will be a function of all chosen guaranteed portions $x = (x_1, \ldots, x_n)$. The problem is to determine the guaranteed portions $x = (x_1, \ldots, x_n)$ to maximize the expected social welfare (or cloud profit if utility contributes to the revenue of the cloud), i.e.,

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n \mathbf{E}[u_i(x_i, D_i)] - \beta K(x), \\ \text{subject to} & x_i \in [0,1], \quad i = 1, \ldots, n, \end{array} \tag{9}$$

where β is the cost of reserving a unit amount of bandwidth in the cloud.

Suppose $D_1, \ldots, D_n$ are Gaussian with predictable means $\mu = (\mu_1, \ldots, \mu_n)$, standard deviations $\sigma = (\sigma_1, \ldots, \sigma_n)$, and covariance matrix $\Sigma = [\sigma_{ij}]_{n \times n}$. Then, by straightforward deduction, we have

$$\mathbf{E}_{D_i}[u_i(x_i, D_i)] = w_{i1} x_i \mu_i - w_{i2} \mathrm{e}^{B_i(1-x_i)\mu_i + \frac{1}{2} B_i^2 (1-x_i)^2 \sigma_i^2}, \tag{10}$$

which is a concave function of x_i in general. Note that setting $w_{i2} = 0$ will yield linear utilities. Under Gaussian demands, the bandwidth reservation K that satisfies (8) is given by

$$\begin{aligned} K(x) &= \mathbf{E}\left[\textstyle\sum_i x_i D_i\right] + \theta(\epsilon)\sqrt{\mathbf{Var}\left[\sum_i x_i D_i\right]} \\ &= \mu^\mathsf{T} x + \theta(\epsilon)\sqrt{x^\mathsf{T} \Sigma x}, \end{aligned} \tag{11}$$

where $\theta(\epsilon) = F^{-1}(1 - \epsilon)$ is a constant depending on ϵ, F being the CDF of normal distribution $\mathcal{N}(0, 1)$ [19]. Clearly, $K(x)$ is a convex function of x in general, although setting Σ to zero will yield a linear cost function. Therefore, problem (9) is convex optimization that can be written in the form of the sharing problem (1) and be solved using the consistency pricing approach.

The guaranteed portions $x = (x_1, \ldots, x_n)$ should be chosen according to both the utility expected from each video channel and the difficulty of providing such a guarantee. Each channel's contribution to the aggregate multiplexed cost for bandwidth guarantee is different. Intuitively speaking, it is harder to provide guaranteed service in a highly volatile channel since more cushion bandwidth needs to be reserved to avoid random demand surges.

IV. A New Class of Distributed Solutions

In this section, we propose a new class of distributed algorithms to solve the sharing problem (1), and discuss its close relationships to fixed-point iterations and the traditional

nonlinear Jacobi algorithm as well as its fundamental differences from the gradient-descent-based consistency pricing method. Let

$$\nabla_i C(x_1, \ldots, x_n) := \partial C(x_1, \ldots, x_n)/\partial x_i$$

denote the partial derivative of the cost function $C(x)$ with respect to the component x_i.

Let x_i be the resource request of user i and p_i be the price charged for user i. We propose two simple pricing-based algorithms to solve problem (1).

Algorithm 1 (Pricing): In iteration t, perform the following updates:

$$p_i^{t+1} := \gamma \nabla_i C\big(x_1^t, \ldots, x_n^t\big) + (1-\gamma) p_i^t, \quad (12)$$

$$x_i^t := \arg\max_{x_i \in [a_i, b_i]} U_i(x_i) - p_i^t x_i, \quad (13)$$

where $\gamma \in (0, 1]$ is an *inertia parameter*. Every price p_i is updated by the resource provider, and each x_i is updated by each user. The update of the (p_i, x_i) pair can be carried out for each user i asynchronously.

Algorithm 2 (Bidding): In iteration t, perform the following updates:

$$p_i^{t+1} := \gamma U_i'\big(x_i^t\big) + (1-\gamma) p_i^t, \quad (14)$$

$$x^t := \arg\max_{x \in \prod_i [a_i, b_i]} (p^t)^{\mathsf{T}} x - C(x), \quad (15)$$

where $\gamma \in (0, 1]$ is an *inertia parameter*, the price p_i is updated by each user in a distributed fashion and the resource allocation x is updated by the resource provider. The update of the $(p_i, x_i(p))$ pair can be carried out for each user i asynchronously.

The following proposition shows that as long as Algorithm 1 (Algorithm 2) converges, it will solve problem (1).

Proposition 1. *For any $\gamma \in (0, 1]$, if Algorithm 1 converges to a fixed point (p^*, x^*), then x^* is the optimal solution to problem (1). The same statement holds for Algorithm 2.*

Proof: This turns out to be a special case of Proposition 6 for the vector versions of the algorithms. ∎

The decentralized workflow of Algorithm 1 is the same as that of consistency pricing, in that each user updates its resource request x_i based on the price p_i set by the provider. The key difference is that in our algorithm, the price p_i is updated by (12), while the price p_i in consistency pricing is updated by (7). On the other hand, Algorithms 2 follows a different workflow from Algorithm 1. Each user updates its price (bid) p_i. Under a price vector p, the resource provider will determine the resource allocation $x(p) = (x_1(p), \ldots, x_n(p))$ by maximizing its profit $p^{\mathsf{T}} x - C(x)$. Then the provider sends $x_i(p)$ back to each user i, which then updates p_i based on $x_i(p)$, and submits the new bid p_i back to the resource provider.

Both algorithms mentioned above have *synchronous* and *asynchronous* versions, depending on whether updates of p_i and $x_i(p)$ are performed for all users i synchronously or asynchronously. For example, in an asynchronous Algorithm 1, p_i and $x_i(p_i)$ can be updated for an user i for any finite number of iterations, before other $x_j(p_j)$ for $j \neq i$ are updated.

The proposed algorithms preserve the following benefits of consistency pricing:

First, the proposed algorithms incur the minimum message-passing overhead and preserve the simple pricing interpretation. Only the prices p and resource allocations x are passed between users and the resource provider. Apparently, any algorithm that relies on more than first-order information, e.g., Newton methods, ADMM, [12], will incur much more message overhead when implemented in a distributed way and can not be interpreted through pricing.

Second, in the proposed algorithms, each network element only utilizes its local information and the available feedback to update variables. In particular, the provider has full knowledge of the cost function C but no knowledge of any U_i, while each user i has full knowledge of its own utility function U_i, but no knowledge of C or any U_j for $j \neq i$.

Third, the proposed algorithms make the minimum assumption on the cooperativeness of network elements. In Algorithm 1, once the resource provider sets a price vector p, each user just needs to maximize its utility minus the cost of using resources, which is a natural action of selfish users. In Algorithm 2, if we can control each user to update p_i using (14), it is natural that the provider will maximize its profit. This property is in contrast to more complicated distributed solutions like ADMM [12], which requires each network element to perform more complicated variable updates than merely maximizing profit or benefit.

However, the proposed algorithms are fundamentally different from consistency pricing which is based on traditional gradient descent. In the following, we explain the difference of proposed algorithms from gradient descent, and interpret them via fixed-point iterations as well as a novel decentralization of the *non-linear Jacobi* algorithm.

A. Interpretation via Fixed-Point Iterations

Let us first show why Algorithms 1 and 2 have an interpretation of fixed-point iterations. For ease of explanation, let us ignore the constraint $x_i \in [a_i, b_i]$ and set the inertia parameter $\gamma = 1$. We show that Algorithms 1 and 2 actually observe the optimality conditions of problem (1).

By first-order conditions, if x^* solves problem (1), we have

$$U_i'(x_i^*) = \nabla_i C(x^*), \quad \forall i. \quad (16)$$

Let us consider Algorithm 1. Suppose the iterations (12), (13) have a fixed point (p^*, x^*). Then at this point, we have

$$U_i'(x_i^*) = p_i^* = \nabla_i C(x_1^*, \ldots, x_n^*), \quad \forall i, \quad (17)$$

where the first equality is due to the first-order condition of (13) and the second equality is due to (12). Thus, x^* satisfies the conditions (16) and is an optimal solution, since problem (1) is convex. Therefore, when $\gamma = 1$, Algorithm 1 is a fixed-point iteration performed on (17) from right to left that aims to converge to x^*. Similarly, Algorithm 2 is a fixed-point iteration performed on (17) in the reverse direction of Algorithm 1.

B. Differences from Traditional Gradient Descent

Although our approach can also be interpreted as pricing, the fundamental difference between the subgradient method in consistency pricing and our approach lies in the price updating rule. In subgradient method (7), the price vector p is updated in the direction of steepest descent of the dual function $q(p)$ for a small step according to the gradient $y(p)x(p)$, i.e., $p_i := p_i - \gamma(y_i(p) - x_i(p_i))$, where $x_i(p_i)$ and $y_i(p)$ maximize the user benefit in (6) and provider profit in (5), respectively. In contrast, in Algorithm 1 (when $\gamma = 1$), $p := \nabla C(x_1, \ldots, x_n)$, i.e., in each iteration, the price vector is directly set to the gradient of the cost function C. Similarly, in Algorithm 2, the price p_i is updated directly by $U_i'(x_i)$, the gradient of each utility function U_i.

This difference will impact the performance in three aspects. *First*, we will show that the new algorithms can better exploit the structural gap between each U_i and C to achieve faster convergence than always following the steepest descent direction. *Second*, in Algorithm 1 or 2, benefit (profit) maximization is performed at *either* users *or* the provider, but not both, whereas in subgradient method (7), such maximization is performed at both users and the provider to obtain $x(p)$ and $y(p)$. Thus, our algorithms can greatly reduce computational complexity when n is large. For example, Algorithm 1 does not require the provider to maximize its profit $p^{\mathsf{T}}y - C(y)$, the complexity of which escalates as n increases, and thus can scale up to a large n. *Third*, we will also show that it is easier to choose γ in the new algorithms, since unlike the subgradient method, γ here is an inertia parameter between 0 and 1 instead of a small stepsize.

We will compare our approach with consistency pricing and gradient descent in terms convergence condition and speed.

C. Nonlinear Jacobi Algorithm as a Limiting Case

In the following, we show that the nonlinear Jacobi algorithm [11] is a limiting case of the asynchronous version of Algorithm 1, while Algorithm 1 can be understood as a novel decentralization of the nonlinear Jacobi algorithm. The nonlinear Jacobi algorithm iteratively optimizes the objective function with respect to each variable x_i (in a parallel fashion) while keeping the rest of the variables x_j ($j \neq i$) fixed. Formally, for problem (1), the algorithm performs the following update for all i in parallel:

$$x_i := \underset{x_i \in [a_i, b_i]}{\arg\max}\, U_i(x_i) - C(x_1, \ldots, x_n) + \textstyle\sum_{j \neq i} U_j(x_j). \quad (18)$$

Apparently, this approach is not a gradient descent algorithm. Nevertheless, it is worth noting that the above nonlinear Jacobi algorithm *cannot* be applied to problem (1) as a distributed solution, because the resource provider does not know any user utility U_i, and the users do not know the cost function C. Neither can the nonlinear Jacobi algorithm be interpreted through pricing.

However, the nonlinear Jacobi algorithm is a limiting case of Algorithm 1 executed *asynchronously*. If for each user i, we perform (12) and (13) of Algorithm 1 for an infinite number of iterations before updating the other variables x_j (for all $j \neq i$) in equation (12), then the x_i obtained this way is the same as the one given by (18), and thus Algorithm 1 becomes the nonlinear Jacobi algorithm. In other words, the proposed Algorithm 1, with a convenient interpretation through pricing, is essentially a novel distributed version of the nonlinear Jacobi algorithm in the limiting case. However, Algorithm 1 is more general than and not the same as the nonlinear Jacobi algorithm if we update all x_i together and asynchronously.

V. Asynchronous Convergence Analysis

We analyze the convergence of the proposed algorithms, and in particular, aim to find sufficient conditions under which an algorithm is a *contraction mapping* [11]. Although more general convergence conditions may be found for an algorithm when executed synchronously, contraction mapping can ensure its convergence even if executed completely asynchronously [11]. Note that an asynchronous algorithm is much better than its synchronous counterpart, since in reality, the latencies of message-passing between network elements are usually heterogeneous. On the other hand, however, the proposed conditions are stronger than necessary for the corresponding algorithms to converge in reality, since they serve only as sufficient conditions for asynchronous convergence.

As some preliminaries, we first briefly describe *contraction mapping* and its properties. Many iterative algorithms can be written as

$$x^{t+1} := T(x^t), \quad t = 0, 1, \ldots, \quad (19)$$

where $x^t \in X \subset \mathbb{R}^n$, and $T : X \mapsto X$ is a mapping from X into itself. The mapping T is called a *contraction* if

$$\|T(x) - T(y)\| \leq \alpha \|x - y\|, \quad \forall x, y \in X, \quad (20)$$

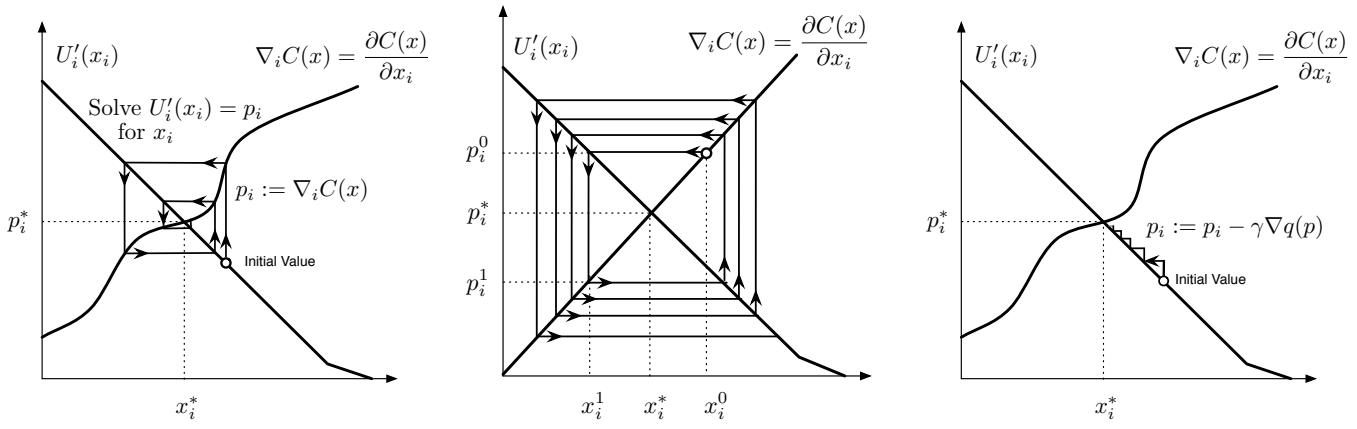
where $\|\cdot\|$ is some norm, and the constant $\alpha \in [0, 1)$ is called the *modulus* of T. Furthermore, the mapping T is called a *pseudo-contraction* if T has a fixed point $x^* \in X$ ($x^* = T(x^*)$) and $\|T(x) - x^*\| \leq \alpha \|x - x^*\|, \quad \forall x \in X$. The following theorem provided and proved in [11] establishes the geometric convergence of both contractions and pseudo-contractions:

Theorem 2. *(Geometric Convergence [11]) Suppose that $X \subset \mathbb{R}^n$ and the mapping $T : X \mapsto X$ is a contraction or a pseudo-contraction with a fixed point $x^* \in X$. Suppose the modulus of T is $\alpha \in [0, 1)$. Then, T has a* unique *fixed point x^* and the sequence $\{x^t\}$ generated by $x^{t+1} := T(x^t)$ satisfies*

$$\|x^t - x^*\| \leq \alpha^t \|x^0 - x^*\|, \quad \forall t \geq 0, \quad (21)$$

for every choice of the initial vector $x^0 \in X$. In particular, $\{x^t\}$ converges to x^* geometrically.

When an update rule T is a contraction or pseudo-contraction with respect to the maximum norm $\|\cdot\|_\infty$, *asynchronous* convergence of the corresponding algorithm can be established using Proposition 2.1 in [11] (pp. 431). In other words, under the sufficient conditions to be provided in the remainder of this section, the corresponding algorithms can converge *asynchronously*.



(a) Algorithm 1: Convergence (b) Algorithm 1: Divergence (c) Gradient Descent

Fig. 2. Illustration of the convergence behavior of Algorithm 1 and the subgradient algorithm (7) in a one dimensional case. "o" represents the starting point.

A. Contraction of Algorithm 1

To simplify notations, in the following context, we will use $c_i(x)$ to stand for $\nabla_i C(x)$ and $u_i(x_i)$ for $U_i'(x_i)$. We denote $\partial_{x_j x_i} C(x) = \nabla_j c_i(x)$ the second order partial derivatives of C. To avoid cluttered notations, we may drop the superscript t for iteration number.

We analyze the convergence of Algorithm 1 when $\gamma = 1$. We rewrite Algorithm 1 in another form amenable to analysis. Denote $[x_i]_i^+$ the *projection* of $x_i \in \Re$ onto the interval $[a_i, b_i]$, i.e.,

$$[x_i]_i^+ = \arg\min_{z \in [a_i, b_i]} |z - x_i|, \quad i = 1, \ldots, n. \tag{22}$$

It is easy to check that (13) is equivalent to

$$x_i(p_i) := \big[\arg\max_{x_i} U_i(x_i) - p_i x_i\big]_i^+ = [u_i^{-1}(p_i)]_i^+ \tag{23}$$

Therefore, letting $\gamma = 1$ and substituting (12) into (23) yields

$$x_i := T_i(x) = [u_i^{-1}(c_i(x))]_i^+, \quad i = 1, \ldots, n, \tag{24}$$

which is an equivalent iteration to Algorithm 1 when $\gamma = 1$.

The following result gives a sufficient (yet not necessary) condition for the convergence of Algorithm 1 with $\gamma = 1$.

Proposition 3. *Suppose $\gamma = 1$. If, for all i, we have*

$$\sum_{j=1}^n |\partial_{x_j x_i} C(x)| < \min_{z_i} |U_i''(z_i)|, \ \forall x \in \prod_i [a_i, b_i], \tag{25}$$

then $T = (T_1, \ldots, T_n)$ given by (24) is a contraction and the sequence $\{x^t\}$ generated by Algorithm 1 converges geometrically to the optimal solution x^* to problem (1), given any initial price vector p^0.

Proof: For each i, define a function $g_i : [0,1] \mapsto \Re$ by

$$g_i(t) = u_i^{-1}\big(c_i(z(t))\big) = u_i^{-1}\big(c_i(tx + (1-t)y)\big).$$

Note that g_i is continuously differentiable. We have

$$\begin{aligned} |T_i(x) - T_i(y)| &= |[u_i^{-1}(c_i(x))]_i^+ - [u_i^{-1}(c_i(y))]_i^+| \\ &\le |g_i(1) - g_i(0)| = \left| \int_0^1 \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} \mathrm{d}t \right| \\ &\le \int_0^1 \left| \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} \right| \mathrm{d}t \le \max_{t \in [0,1]} \left| \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} \right|, \end{aligned}$$

where the first inequality is because $\big|[x_i]_i^+ - [y_i]_i^+\big| \le |x_i - y_i|$ for all $x_i, y_i \in \Re$. Furthermore, the chain rule yields

$$\begin{aligned} \left| \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} \right| &= \big|\textstyle\sum_{j=1}^n \nabla_j u_i^{-1} \circ c_i(tx + (1-t)y) \cdot (x_j - y_j)\big| \\ &\le \big|(u_i^{-1})'(c_i(z(t)))\big| \cdot \textstyle\sum_{j=1}^n \big|\nabla_j c_i(z(t))\big| \cdot |x_j - y_j|, \end{aligned}$$

where the notation $u_i^{-1} \circ c_i(x)$ is used to denote the function $u_i^{-1}(c_i(x))$. If condition (25) holds, we have

$$\begin{aligned} \textstyle\sum_{j=1}^n |\nabla_j c_i(x)| &< \min_{z_i} |u_i'(z_i)| \le |u_i'\big(u_i^{-1}(c_i(x))\big)| \\ &= 1/|(u_i^{-1})'(c_i(x))|, \quad \forall x \in \textstyle\prod_i [a_i, b_i]. \end{aligned}$$

Therefore, there exists a positive $\alpha < 1$ such that

$$\left| \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} \right| \le \alpha \max_j |x_j - y_j| = \alpha \|x - y\|_\infty, \ \forall t \in [0,1], \ \forall i.$$

Therefore, we have shown that

$$\|T(x) - T(y)\|_\infty \le \alpha \|x - y\|_\infty, \quad \forall x, y \in \textstyle\prod_i [a_i, b_i],$$

and T is a contraction with modulus α with respect to the maximum norm in $\prod_i [a_i, b_i]$. By Theorem 2, Algorithm 1 converges geometrically to x^*. ∎

Proposition 3 implies that Algorithm 1 can converge when $|U_i''(x_i)|$ are large and $|\partial_{x_j x_i} C(x)|$ are small. Each $|U_i''(x_i)|$ is large when U_i is nonlinear with a relatively big curvature. On the other hand, $|\partial_{x_j x_i} C(x)|$ will be small, for example, if the covariances Σ in the example (11) is small to yield a near-linear cost. In fact, the cost function in (11) is a *cone*, and thus its curvatures $|\partial_{x_j x_i} C(x)|$ are nearly zero if x is away from 0.

Note that Proposition 3 provides a rather strong sufficient condition for Algorithm 1 to be a contraction and thus to achieve convergence even when each variable is updated asynchronously in an arbitrary order. In reality, such conditions are not *necessary* for Algorithm 1 to converge. The convergence condition of Algorithm 1 when the inertia parameter $\gamma < 1$ is much harder to analyze. In Sec. VII, we show that as long as $|\partial_{x_j x_i} C(x)|$ are small relative to $|U_i''(x_i)|$, Algorithm 1 can also converge for $\gamma < 1$.

B. Contraction of Algorithm 2 and Consistency Pricing

The convergence of Algorithm 2 is much harder to analyze in general when there are constraints $a_i \le x_i \le b_i$, since $x(p)$ in (15) cannot be decoupled as projections onto intervals $[a_i, b_i]$. As many engineering problems have solutions in the interior of the feasible region, in the following, we give sufficient convergence conditions of Algorithm 2 in the case of unconstrained optimization for both $\gamma = 1$ and $\gamma < 1$. In this case, Algorithm 2 is rewritten as

$$\begin{cases} p_i := T_i(p) = (1-\gamma)p_i + \gamma u_i\big(x_i(p)\big), \quad \forall i, \\ x(p) := \arg\max_x p^{\mathsf{T}} x - C(x) \end{cases} \tag{26}$$

Proposition 4. *Suppose that problem (1) has no constraints and that the Hessian matrix of $C(x)$ is $H(x) = [\partial_{x_j x_i} C(x)]_{n\times n}$. Denote the inverse of $H(x)$ as*

$$P(x) = [H(x)]^{-1} = [\partial_{x_j x_i} C(x)]^{-1}_{n\times n} = [P_{ij}(x)]_{n\times n}.$$

If the following condition holds:

$$\begin{cases} \gamma = 1, \\ \sum_{j=1}^n |P_{ij}(x)| < |U_i''(x_i)|^{-1}, \quad \forall x,\ \forall i, \end{cases} \tag{27}$$

or if the following condition holds:

$$\begin{cases} 0 < \gamma \le (1 + P_{ii}(x)\cdot |U_i''(x_i)|)^{-1}, \\ \sum_{j\neq i} |P_{ij}(x)| - P_{ii}(x) < |U_i''(x_i)|^{-1}, \end{cases} \quad \forall x,\ \forall i, \tag{28}$$

then $T = (T_1, \ldots, T_n)$ given by (26) is a contraction and the sequence $\{x^t\}$ generated by Algorithm 2 converges geometrically to the optimal solution x^* to problem (1).

Please refer to the appendix for the proof of the above proposition. Unlike the convergence condition of Algorithm 1, Proposition 4 essentially implies the contrary. That is, Algorithm 2 can converge when $|U_i''(x_i)|$ is small and $|\partial_{x_j x_i} C(x)|$ is big. To see this, we notice that both condition (27) (the case of $\gamma = 1$) and condition (28) (the case of $\gamma < 1$) require a large $|U_i''(x_i)|^{-1}$, which implies a small $|U_i''(x_i)|$. Furthermore, both (27) and (28) favour a small $|P_{ij}(x)|$ which indicates a bigger $|\partial_{x_j x_i} C(x)|$, since $P(x)$ is the inverse of $H(x) = [\partial_{x_j x_i} C(x)]_{n\times n}$. The above conditions, for example, can easily be satisfied when each utility $U_i(x_i)$ is a *linear function* of x_i with $|U_i''(x_i)| = 0$.

There are many results on the convergence of (sub)gradient algorithms. For example, for a sufficiently small constant stepsize γ, the algorithm is guaranteed to converge to the optimal value [18]. For comparison, here we derive a sufficient condition similar to (28), under which the subgradient algorithm applied to the dual problem, i.e., the consistency pricing algorithm (7), is a contraction for the unconstrained version of problem (1).

Proposition 5. *Suppose that problem (1) has no constraints. Recall that the Hessian matrix of $C(x)$ is $H(x) = [\partial_{x_j x_i} C(x)]_{n\times n}$ and the inverse of $H(x)$ is $P(x) = [H(x)]^{-1} = [P_{ij}(x)]_{n\times n}$.*

If we have the following condition for all x and i:

$$\begin{cases} 0 < \gamma \le \big(P_{ii}(x) + 1/|U_i''(x_i)|\big)^{-1}, \\ \sum_{j:j\neq i} |P_{ij}(x)| - P_{ii}(x) < |U_i''(x_i)|^{-1}, \end{cases} \tag{29}$$

then $T = (T_1, \ldots, T_n)$ defined by consistency pricing (7) is a contraction and the sequence $\{x(p^t)\}$ converges geometrically to x^*.

Please refer to the appendix for the proof of the above proposition. For consistency pricing to be a contraction, we find that condition (29) is exactly the same as the convergence condition (28) for Algorithm 2 when $\gamma < 1$, except for the choice of γ. However, it is much easier to choose γ in Algorithm 2. In condition (28) for Algorithm 2, we only need to know the relative values of $\partial_{x_j x_i} C(x)$ and $|U_i''(x_i)|$ to estimate $P_{ii}(x)\cdot|U_i''(x_i)|$, since $P(x)$ is the inverse of $H(x) = [\partial_{x_j x_i} C(x)]_{n\times n}$. In contrast, in condition (29) for consistency pricing, we need knowledge of the absolute values of $P_{ii}(x) + 1/|U_i''(x_i)|$ to estimate a valid range for γ.

C. Discussions

We now discuss the insights from the above analysis for algorithm and parameter selection. Basically, whether we should use Algorithm 1 or Algorithm 2 depends on the relative *curvatures* of C and U_i: when $|U_i''|$ is large, Algorithm 1 may converge; when $|U_i''|$ is small and $|U_i''|^{-1}$ is big, Algorithm 2 may converge.

We further illustrate the convergence condition and speed of Algorithm 1 when $\gamma = 1$ in Fig. 2. For user i, starting from an initial x_i^0, the initial price p_i^0 is first set to $p_i^0 = \nabla_i C(x^0)$. Then user i maximizes its benefit $U_i(x_i) - p_i^0 x_i$ to get x_i^1, which is actually the solution to $U_i'(x_i) = p_i^0$ by the first-order condition. Therefore, Algorithm 1 updates (x_i, p_i) in a "spiral" fashion jumping between the two curves $U_i'(x_i)$ and $\nabla_i C(x)$ until their intersection is reached, where $U_i'(x_i^*) = \nabla_i C(x^*)$.

Clearly, the convergence will depend on the slopes of U_i' and $\nabla_i C$. Algorithm 1 will converge when the gradient of $\nabla_i C(x)$ is small as compared to the derivative of U_i', or equivalently, when $\partial_{x_j x_i} C(x)$ are small compared to U_i''. The larger the gaps, the faster the convergence. On the other hand, Fig. 2(b) shows the opposite case when $\partial_{x_j x_i} C(x)$ (the gradient of $\nabla_i C(x)$) is large compared to U_i'' (the derivative of U_i'). In this case, Algorithm 1 might diverge. However, since Algorithm 2 updates *in the reverse direction of* Algorithm 1, Algorithm 2 may converge instead.

The proposed algorithms essentially utilizes the curvature differences between C and U to achieve the spiral convergence, e.g., when $|\partial_{x_j x_i} C(x)|$ are small relative to $|U_i''(x_i)|$, and vice versa. This fact will be exemplified in our case study with real-world traces in Sec. VII. In contrast, the convergence behavior of consistency pricing is illustrated in Fig. 2(c), which actually uses the subgradient algorithm (essentially gradient descent). Similar to Algorithm 1, each user i will choose a benefit-maximizing x_i in each iteration. However, the difference is that the subgradient method will update price p_i by a small step in the direction of the steepest descent of the dual function, i.e., $\nabla q(p)$. As a result, when $y_i(p)$ and $x_i(p_i)$ do not differ much, it may take a large number of iterations before the algorithm converges.

Finally, γ is an *inertia parameter* that trades convergence speed off for less stringent convergence conditions. From Proposition 4, we can see that when $\gamma < 1$ in condition (28),

the requirement on the structures of U and C is *less strong* than that in condition (27). This is because we have

$$\sum_{j:j\neq i}|P_{ij}(x)| - P_{ii}(x) \leq \sum_{j=1}^{n}|P_{ij}(x)|,$$

and thus $\sum_{j=1}^{n}|P_{ij}(x)| < |U_i''(x_i)|^{-1}$ implies $\sum_{j:j\neq i}|P_{ij}(x)| - P_{ii}(x) < |U_i''(x_i)|^{-1}$, but not vice versa. In other words, by allowing $\gamma < 1$, we can achieve convergence on a wider range of U and C functions. This fact will be substantiated in simulation for both Algorithm 1 and Algorithm 2.

VI. Extension to Handle Coupled Constraints

In this section, we extend the proposed algorithms to approach more general utility maximization problems with arbitrary coupling in the objective function and a certain class of coupled constraints. A general form to pose distributed optimization problems is through *consensus optimization* [12]:

$$\begin{array}{ll}\text{minimize} & \sum_{i=0}^{m}F_i(x_1,\ldots,x_n)\\ \text{subject to} & (x_1,\ldots,x_n)\in\mathcal{C}_i,\quad i=0,1,...,m,\end{array} \tag{30}$$

where $F_i : \mathbb{R}^n \mapsto \mathbb{R}$ are strictly convex functions, and $\mathcal{C}_i$ are convex subsets of $\mathbb{R}^n$. *We assume that each F_i may only depend on a subset of $x_1,\ldots,x_n$.*

Such a problem naturally arises in networking systems, where the utility of each user F_i depends on a subset of all resource allocation decisions $(x_1,\ldots,x_n)$ and each $(x_1,\ldots,x_n) \in \mathcal{C}_i$ represents a convex constraint coupled among $x_1,\ldots,x_n$, e.g., $x_1 + \ldots + x_n \leq B$. Maximizing the sum of concave utility functions in utility maximization is equivalent to minimizing the sum of loss functions F_i in consensus optimization (30). Note that problem (1) is a special case of problem (30), where each U_i only depends on x_i, while $-C$ plays the role of F_0 and there is no coupled constraint.

Problem (30) not only models utility maximization problems with arbitrary coupling in the objective and constraints, but also has wide applications in machine learning problems [12]. For example, in model fitting, the vector x represents the set of parameters in a model and F_i represents the loss function associated with the ith training sample. The learning goal is to find the model parameters x that minimizes the sum of loss functions for all training samples.

A. The Vector Version of Fixed-Point Pricing

We first extend Algorithm 1 to the case that each variable is a vector instead of a scalar. Suppose $z_i \in \mathbb{R}^{l_i}$ for $i = 1,\ldots l$, and $z = (z_1,\ldots,z_l) \in \mathbb{R}^{\sum_{i=1}^{l} l_i}$. Now the vector version of the sharing problem (1) is

$$\begin{array}{ll}\text{maximize} & \sum_{i=1}^{l}U_i(z_i) - C(z)\\ \text{subject to} & z_i\in\mathcal{Q}_i,\quad i=1,\ldots,l,\end{array} \tag{31}$$

where $\mathcal{Q}_i \subseteq \mathbb{R}^{l_i}$ is a convex set. Accordingly, we obtain revised Algorithm 1 as follows:

Algorithm 1 (Vector-Version Pricing): In iteration t, perform the following updates:

$$p_i^{t+1} := (1-\gamma)p_i^t + \gamma\nabla_i C\big(z_1^t,\ldots,z_l^t\big), \tag{32}$$

$$z_i^t := \arg\max_{z_i\in\mathcal{Q}_i} U_i(z_i) - (p_i^t)^{\mathsf{T}} z_i, \tag{33}$$

where the prices $p_i \in \mathbb{R}^{l_i}$ are updated by the resource provider, and $z_i \in \mathbb{R}^{l_i}$ is updated by each user i in a distributed fashion.

The vector version of Algorithm 2 can be derived in a similar way, which we omit due to space constraint. Then we have the following statement:

Proposition 6. *For any $\gamma \in (0, 1]$, if the vector version of Algorithm 1 (or Algorithm 2) has a fixed point (p^*, z^*), then z^* is the optimal solution to problem (31).*

Please refer to the appendix for the proof of the above proposition. Note that convergence conditions similar to those in Sec. V can be derived for the vector versions of the algorithms. For example, in Proposition 3, $\partial_{z_j z_i}C(z)$ and $\partial_{z_i z_i}U_i(z_i)$ will all become matrices, as $z_i \in \mathbb{R}^{l_i}$ are vectors. Thus, the absolute values should be replaced by matrix norms (e.g., Frobenius norm [11]) in the corresponding analysis.

B. Solving Consensus Optimization (30)

We show that the dual problem of consensus optimization (30) can be converted into an equivalent problem of the vector sharing problem (31) and can be solved by the vector version of Algorithm 1 for a specific class of coupled constraints. Introducing auxiliary variables $y_i \in \mathbb{R}^n$, (30) is equivalent to

$$\begin{array}{ll}\text{minimize} & F_0(x) + \sum_{i=1}^{m}F_i(y_i)\\ \text{subject to} & x=(x_1,\ldots,x_n)\in\mathcal{C}_0,\\ & y_i = x,\ y_i\in\mathcal{C}_i,\quad i=1,...,m,\end{array} \tag{34}$$

where $\mathcal{C}_i \subseteq \mathbb{R}^n$, $i = 0, 1, \ldots, m$. It is not hard to check that the dual problem of (34) is

$$\begin{array}{ll}\text{maximize} & q(\lambda) = \sum_{i=1}^{m}q_i(\lambda_i) + q_0\big(\sum_{i=1}^{m}\lambda_i\big)\\ \text{subject to} & \lambda_i\in\mathbb{R}^n,\quad i=1,\ldots,m,\end{array} \tag{35}$$

where $\lambda_i \in \mathbb{R}^n$ is the Lagrangian multiplier corresponding to the constraint $y_i = x$. Define $\lambda := (\lambda_1,\ldots,\lambda_m) \in \mathbb{R}^{mn}$, and concave functions q_0 and q_i are given by

$$q_0(\lambda) = q_0\big(\textstyle\sum_{i=1}^{m}\lambda_i\big) = \min_{x\in\mathcal{C}_0}\big\{F_0(x) - \big(\sum_{i=1}^{m}\lambda_i\big)^{\mathsf{T}}x\big\} \tag{36}$$

$$q_i(\lambda_i) = \min_{y_i\in\mathcal{C}_i}\{F_i(y_i) + \lambda_i^{\mathsf{T}}y_i\}. \tag{37}$$

The dual function $q(\lambda)$ is continuously differentiable with

$$\frac{\partial q(\lambda)}{\partial\lambda_i} = \frac{\partial q_i(\lambda_i)}{\partial\lambda_i} + \frac{\partial q_0(\lambda)}{\partial\lambda_i} = y_i(\lambda_i) - x(\lambda), \tag{38}$$

where $x(\lambda) \in \mathcal{C}_0$ and $y_i(\lambda_i) \in \mathcal{C}_i$ are the *unique minimizing vectors* in (36) and (37), respectively.

Now the dual problem (35) is exactly the vector version of the sharing problem (31), with

$$U_i(\lambda_i) = q_i(\lambda_i),\quad i=1,\ldots,m$$
$$C(\lambda) = C(\lambda_1,\ldots,\lambda_m) = -q_0(\textstyle\sum_{i=1}^{m}\lambda_i),$$

subject to $\lambda_i \in \mathcal{Q}_i = \mathbb{R}^n$, for $i = 1,\ldots,m$.

Therefore, we can apply the vector version of Algorithm 1 to the dual problem (35) with $\lambda_1,\ldots,\lambda_m$ being the variables, leading to the following **alternating update rules** for all $i = 1,\ldots,m$:

$$p_i := (1-\gamma)p_i - \gamma\cdot\frac{\partial q_0(\sum_{i=1}^{m}\lambda_i)}{\partial\lambda_i} = (1-\gamma)p_i + \gamma x(\lambda) \tag{39}$$

$$\lambda_i := \arg\max_{\lambda_i} q_i(\lambda_i) - p_i^{\mathsf{T}}\lambda_i, \tag{40}$$

where $\lambda_i \in \mathbb{R}^n$ are decision variables, $p_i \in \mathbb{R}^n$ are price vectors. F_0 plays the role of a "resource provider" and each F_i represents a "user" node. On one hand, each "user" updates its variable λ_i in a distributed fashion by solving (40), where $q_i(\lambda_i)$ is given by (37) with $\partial q_i(\lambda_i)/\partial\lambda_i = y_i(\lambda_i)$ such that the optimal λ_i^* for (40) is found when $y_i(\lambda_i^*) = p_i$. On the other hand, the "provider" will update all the prices $p_1, \ldots, p_m$ via an intermediate variable $x(\lambda) \in \mathcal{C}_0$ computed by solving the minimization in (36). Then the price vector $p_i \in \mathbb{R}^n$ is sent to user i again to compute a new λ_i.

It is worth noting that the condition to use the above updates is that (40) must be well defined. Note that (40) is solving the optimization

$$\max_{\lambda_i \in \mathbb{R}^n} \min_{y_i \in \mathcal{C}_i} \{F_i(y_i) + \lambda_i^{\mathsf{T}}(y_i - p_i)\},$$

which is exactly solving the dual problem of $\min F_i(y_i)$ subject to $y_i \in \mathcal{C}_i$ and $y_i = p_i$ (where subgradient algorithms can be used). For this problem to be feasible, we need $p_i \in \mathcal{C}_i$, since otherwise $\min F_i(y_i)$ would be unbounded and the update (40) would not be well defined. Since p_i is a convex combination of the previous p_i and $x(\lambda)$, and $x(\lambda) \in \mathcal{C}_0$, we have $p_i \in C_0$. Therefore, a *sufficient (yet not necessary)* condition for updates (39) and (40) to work is that $\mathcal{C}_0 \subseteq \mathcal{C}_i$ for $i = 1, \ldots, m$.

This condition can easily be satisfied in a class of resource sharing problems, where each user i has a decision variable x_i known to fall in some range $\mathcal{C}_i$ *a priori*, while the provider further enforces some coupled constraint across different x_i. For example, in the following sharing problem with an additional coupled constraint:

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n U_i(x_i) - C(x) \\ \text{subject to} & x_i \in [0,1], \quad i = 1, \ldots, n, \\ & \sum_{i=1}^n x_i \le B, \end{array} \tag{41}$$

where $B \in \mathbb{R}$, the variable range $\mathcal{C}_i = [0,1]$ is often known *a priori* by the provider C. Then when the provider imposes an additional constraint of bounded sum in $\mathcal{C}_0$, we have $\mathcal{C}_0 \subset \mathcal{C}_i$ for all i.

If $\mathcal{C}_0$ is the *interior* of each $\mathcal{C}_i$, we can further simplify (40). Recall that λ_i in (40) is the solution to $y_i(\lambda_i) = p_i$. In this case, $p_i \in \mathcal{C}_0$ is in the interior of $\mathcal{C}_i$. By first-order conditions of (37), the optimization in (40) at each user i can be replaced by the following straightforward computation:

$$\lambda_i = -\nabla F_i\big(y_i(\lambda_i)\big) = -\nabla F_i(p_i).$$

VII. Cloud Bandwidth Sharing: a Case Study

We evaluate the algorithm performance on a typical large-scale real-time cloud network reservation problem described in Sec. III-B. In this case study, we show that contraction assumptions for gradient algorithms often do not hold in reality, which causes slow convergence and challenges for stepsize selection. We also compare the proposed approach with ADMM in terms of convergence and running time, the latter being a powerful distributed optimization solver which *cannot* be interpreted as pricing.

Recall from Sec. III-B that problem (9) is to decide the guaranteed portions x for all channels to maximize the expected social welfare $\sum_i U_i(x_i) - C(x)$, where $U_i(x_i) = \mathbf{E}[u_i(x_i, D_i)]$ and $C(x) = \beta K(x)$ are given by (10) and (11), respectively. Problem (9) can be solved through pricing, i.e., each channel i can be charged a price $p_i x_i$ to control the channel's choice of x_i.

The demand statistics μ and Σ are calculated from real traces collected from a commercial VoD system, called UUSee [19]. The data used here contains the bandwidth demands (in Mbps) of $n = 468$ video channels (each we suppose to be a *tenant* of the cloud), measured every 10 minutes, over a 810-minute period during 2008 Olympics. Assume the means μ and covariance matrix Σ of the channel demands remain the same within each 10-minute period. We can estimate μ, Σ for the upcoming 10 minutes based on historical demands using the time-series forecasting techniques such as ARIMA and GARCH presented in our previous work [19], [23]. Once μ and Σ are obtained, we solve problem (9) for the optimal guaranteed portions x in this period. After the 10-minute period has past, the system proceeds to estimate μ, Σ for the next 10 minutes and computes a new x by solving problem (9) again. The problem (9) must be solved *distributively* with the minimum message-passing, assuming that each channel does not know other channels' utilities or the multiplexed (coupled) cost function $K(x)$. Furthermore, problem (9) must be solved in at most a few seconds in order not to delay the entire resource allocation process performed every 10 minutes.

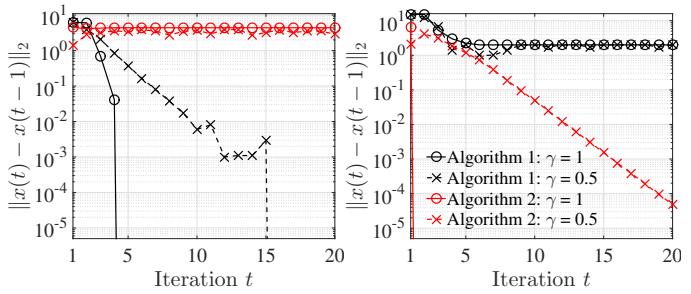
We set $\epsilon = 0.01$, $B_i = 0.5$, $\beta = 0.5$ in the experiments, and apply different algorithms for each of the 81 consecutive 10-minute periods.

A. Convergence Analysis and Algorithm Selection

From the discussions in Sec. V, we know that a basic rule to follow when choosing between Algorithm 1 and 2 is to check the relative largeness of $|U_i''|$ and $|\partial_{x_j x_i} C|$. The functions in this case study is a good example to discuss the convergence conditions and parameter selection for Algorithm 1 and 2. Apparently, if $w_{i2} = 0$ in (10), we have a linear U_i with $|U_i''| = 0$. In this case, conditions in Proposition 4 are met, while the condition in Proposition 3 is violated. Therefore, we should use Algorithm 2. On the other hand, if the demand covariances Σ is small, we will have a nearly linear cost function from (11), with $|\partial_{x_j x_i} C|$ all close to 0. In this case, the convergence condition in Proposition 3 will be met and Algorithm 1 should be used.

However, we note that the cost (11) is a cone centered at 0, and thus has small $|\partial_{x_j x_i} C(x)|$ values as long as x is away from 0. Therefore, we fix μ and Σ to be those computed from the real traces and thus fix the form of C, while considering two versions of U_i: 1) concave utility with $w_{i1} = w_{i2} = 1$ in (10), and 2) linear utility with $w_{i1} = 1$ and $w_{i2} = 0$. We run Algorithm 1 and 2 under these two cases (in the first 10-minute period) and plot their convergence behavior in Fig. 3. The algorithms will stop when $\|x(t) - x(t-1)\|_2 \le 10^{-30}$.

We observe that Algorithm 1 converges for concave utilities while Algorithm 2 does not. On the contrary, Algorithm 2



(a) Concave U_i ($w_{i1} = w_{i2} = 1$) (b) Linear U_i ($w_{i1} = 1$, $w_{i2} = 0$)

Fig. 3. The evolution of $\|x(t) - x(t-1)\|_2$ under both Algorithm 1 and Algorithm 2 for different forms of U_i given by (10), while the form of the cost function C is fixed to (11).

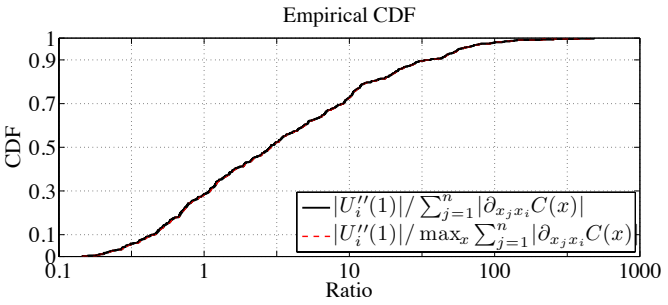


Fig. 4. Sampling of $\min_{x_i} |U_i''(x_i)|/\max_x \sum_{j=1}^n |\partial_{x_j x_i} C(x)|$ and $\min_{x_i} |U_i''(x_i)|/\sum_{j=1}^n |\partial_{x_j x_i} C(x)|$ for all $i = 1, \ldots, 468$. The max is taken over all the randomly sampled x between 0.8 and 1. In this case, $\min_{x_i} |U_i''(x_i)| = |U_i''(1)|$.

converges for linear utilities while Algorithm 1 does not. Such observation corroborates the correctness and usefulness of our analysis in Sec. V.

Furthermore, for both algorithms, convergence is faster when we set $\gamma = 1$, provided that convergence *can* be achieved when $\gamma = 1$. When we solve 81 optimization problems, one for each 10-minute period, it will be clear that with an inertia parameter $\gamma < 1$, although convergence may be slower, the algorithm is more robust and permits a larger range of U and C.

In a general setting, let us demonstrate how to check convergence conditions and select algorithms. We illustrate how to check condition (25) for Algorithm 1 in the first 10-minute period. (Proposition 4 for Algorithm 2 can be checked in a similar way.) From (11), we get

$$\partial_{x_j x_i} C(x) = \beta\theta(\epsilon)\left(\frac{\sigma_{ij}}{\sqrt{x^\mathsf{T}\Sigma x}} - \frac{x^\mathsf{T}\Sigma_i^\mathsf{T}\Sigma_j x}{(x^\mathsf{T}\Sigma x)^{\frac{3}{2}}}\right),$$

where Σ_i is the ith row of the covariance matrix Σ, and σ_{ij} is the covariance between channels i and j. Similarly, it is easy to derive $U_i''(x_i)$ from (10) and verify that $|U_i''(x_i)|$ *always reaches the minimum* at $x_i = 1$, i.e., $\min_{x_i} |U_i''(x_i)| = |U_i''(1)|$. Although we do not know which x maximizes $\sum_{j=1}^n |\partial_{x_j x_i} C(x)|$, we can use a sampling method. For each i, we sample x uniformly at random between 0.8 and 1 for 100 times and plot the CDFs of $|U_i''(1)|/\sum_{j=1}^n |\partial_{x_j x_i} C(x)|$ and $|U_i''(1)|/\max_x \sum_{j=1}^n |\partial_{x_j x_i} C(x)|$ in Fig. 4, from which we see that the inequality $\min_{x_i} |U_i''(x_i)| > \max_x \sum_{j=1}^n |\partial_{x_j x_i} C(x)|$ is satisfied for about 70% of all the 468 channels (users). Note that since the presented conditions are sufficient (not necessary) conditions, the algorithm may actually converge under much milder conditions. Now we know that the inequality (25) is satisfied for most i and $|U_i''(x_i)|$ is much greater than $\sum_{j=1}^n |\partial_{x_j x_i} C(x)|$ in most cases, we may tentatively believe Algorithm 1 will converge for the dataset. The trial run on this single time period asserts that Algorithm 1 indeed converges while Algorithm 2 does not.

In general, such a sampling method for algorithm selection is described as follows. Sample the second-order derivatives $\partial_{x_j x_i} C(x)$ and $U_i''(x_i)$ over the feasible domain of x and check whether $\min_{x_i} |U_i''(x_i)| > \max_x \sum_{j=1}^n |\partial_{x_j x_i} C(x)|$ is met (for Algorithm 1) or $\sum_{j\neq i} |P_{ij}(x)| - P_{ii}(x) < |U_i''(x_i)|^{-1}$ is met (for Algorithm 2) for most i and x. If the inequality is met for the majority of i and x, we can use the corresponding algorithm. Moreover, since Algorithm 1 and 2 update prices in opposite directions, when one algorithm does not converge, the other one is likely to converge.

In the following, we always set $w_{i1} = w_{i2} = 1$, since a concave utility is closer to reality. And in this case, we already know that Algorithm 1 converges while Algorithm 2 does not. Therefore, we will focus on the comparison of Algorithm 1 with existing algorithms.

B. Comparison to Consistency Pricing and Gradient Descent

We compare the performance of Algorithm 1 to consistency pricing (gradient descent applied to the dual problem) as well as the gradient descent directly applied to the primal problem (9). However, note that unlike consistency pricing and Algorithm 1 the latter approach cannot be interpreted as pricing. Instead, the provider needs to collect all $U_i'(x_i)$ in addition to x_i in each round to compute the gradient of $\sum_i U_i(x_i) - C(x)$ and thus has to assume that the video channels will cooperate to pass this information.

The initial price $p(1)$ is set to the optimal price vector $\beta(\mu + \theta(\epsilon)\sigma)$ without multiplexing, which is easy to compute due to no coupling [19]. Since the value of $\mathrm{SW}(x)$ is unknown to the cloud, we let an algorithm stop at iteration t if either $\|x(t) - x(t-1)\|_\infty < 10^{-2}$ or $t = 100$. Since Algorithm 1 is a contraction, which even converges asynchronously, we adopt an asynchronous stop criterion: if $|x_i(t) - x_i(t-1)| < 10^{-2}$, then tenant i stops updating x_i. In contrast, as consistency pricing is not a contraction, we evaluate both synchronous and asynchronous stop criteria. We do not assume heterogeneous message-passing delays, which may further enlarge the benefit of our asynchronous algorithm.

Fig. 5 plots the CDF of iterations needed to converge in all 81 experiments, one for each 10-minute period. Algorithm 1 always converges in 10 iterations, when $\gamma = 0.5$. If $\gamma = 1$, the convergence rate can be further speeded up to 5 iterations on average, although it does not converge in one single period. This substantiates our theoretical analysis in Sec. V that an inertia parameter $\gamma < 1$ can be used to increase the robustness of the algorithm to ensure convergence for a wider range of

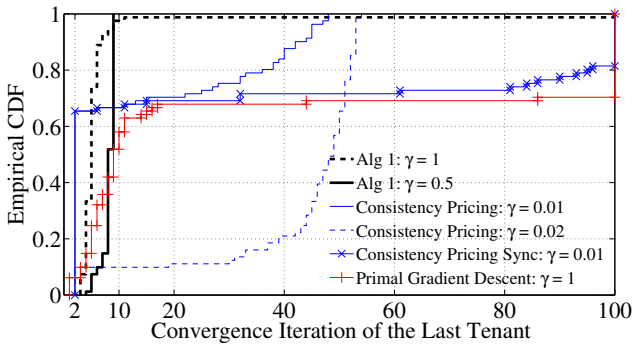


Fig. 5. CDF of convergence iterations in all 81 10-minute periods.

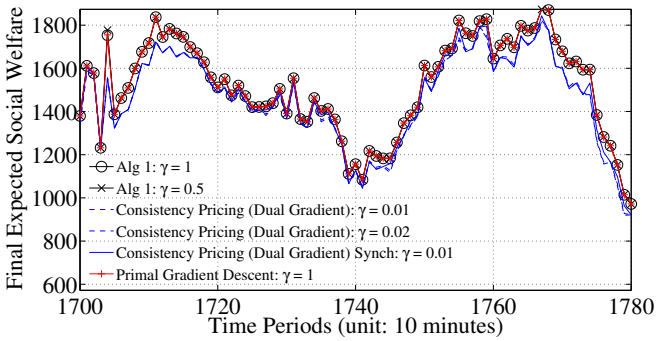


Fig. 6. Final objective values $\mathrm{SW}(x^*)$ produced by different algorithms.

U_i and C (which vary across all 81 experiments), although it may slightly slow down the convergence speed.

In general, gradient methods need much more iterations to converge, Consistency pricing (dual gradient descent) faces a dilemma in selecting the stepsize γ. Since algorithms will stop when x is changing by less than 10^{-2}, we cannot set too small a γ, in which case, the algorithm will stop when $t = 2$ due to too small a change in price p. Fig. 5 shows that consistency pricing with all considered γ produce erroneous outputs in many cases, as they stop at $t = 2$. Increasing γ reduces errors, but will lead to even longer convergence time. Fig. 5 suggests that unlike Algorithm 1, no constant stepsize γ for consistency pricing can simultaneously achieve correct outputs and fast convergence in all 81 experiments.

For the primal gradient descent, the stepsize is optimized to $\gamma = 1$ to achieve the fastest convergence. We can see that the primal gradient descent can not beat our algorithm in terms of convergence iterations, even though it makes additional assumption on user cooperativeness.

We further plot the final objectives $\mathrm{SW}(x^*)$ achieved by different algorithms in Fig. 6. We can see that Algorithm 1 produces the same best final objective value as the primal gradient descent. Consistency pricing with asynchronous stop rules, although converging, may output wrong answers because they are not contractions and in theory, should not be applied with asynchronous stop rules. Consistency pricing with synchronous stopping achieves slightly higher objective values if not stopping at $t = 2$. But since they do not converge within 100 iterations, they are still inferior to Algorithm 1.

Let us zoom into the experiment for the first 10-minute period. From Fig. 7, we see that Algorithm 1 has an increasing

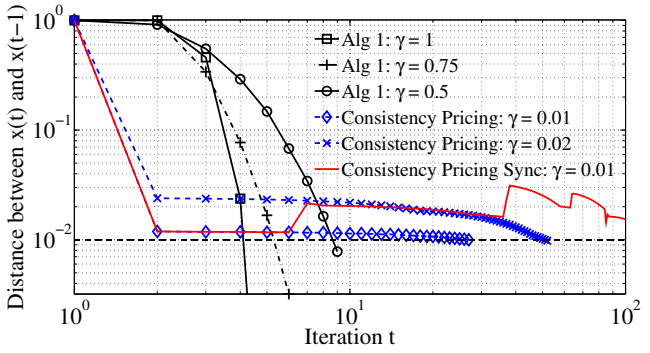


Fig. 7. The evolution of $\|x(t) - x(t-1)\|_\infty$ in the first experiment.

convergence speed as t increases. In contrast, consistency pricing performs well in the first two iterations, but its convergence slows down as the gradient becomes smaller. It is confirmed that consistency pricing is extremely sensitive to γ, the choice of which is a challenge.

C. Comparison to ADMM

Recently, ADMM is gaining its popularity in tackling large-scale distributed optimization problems. We now provide a performance comparison to ADMM. Let

$$\begin{cases} f_i(x_i) = -\mathbf{E}_{D_i}[u_i(x_i, D_i)], & i = 1, \ldots, n, \\ f_0(x) = \beta K(x). \end{cases} \tag{42}$$

Then, problem (9) is rewritten as minimize $\sum_{i=1}^n f_i(x_i) + f_0(x)$, where $x_i \in [0, 1]$. Further rewriting the above into a *general form consensus optimization* [12], problem (9) is equivalent to

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{i=0}^n f_i(y_i) \\ \text{subject to} \quad & y_i - \tilde{z}_i = 0, \quad i = 0, 1, \ldots, n, \end{aligned} \tag{43}$$

where $y_0 \in \mathbb{R}^n$, $y_i \in \Re$ for $i = 1, \ldots, n$, and each $\tilde{z}_i$ is a linear function of a global auxiliary variable $z = (z_1, \ldots, z_n) \in \mathbb{R}^n$ defined as $\tilde{z}_0 = (z_1, \ldots, z_n)$, $\tilde{z}_i = z_i$, $i = 1, \ldots, n$. According to the ADMM approach for general consensus optimization outlined in pp. 53-55 [12], the iterative updates based on augmented Lagrangian are given by

$$\begin{aligned} y_i^{t+1} &:= \arg\min_{y_i} \left(f_i(y_i) + \lambda_i^{t\mathsf{T}} y_i + (\rho/2)\|y_i - \tilde{z}_i^t\|_2^2\right), \\ z^{t+1} &:= \arg\min_{z} \left(\textstyle\sum_{i=0}^n (-\lambda_i^{t\mathsf{T}} \tilde{z}_i + (\rho/2)\|y_i^{t+1} - \tilde{z}_i\|_2^2)\right), \\ \lambda_i^{t+1} &:= \lambda_i^t + \rho(y_i^{t+1} - \tilde{z}_i^{t+1}), \end{aligned} \tag{44}$$

where $(y_1, \lambda_1), \ldots, (y_n, \lambda_n)$ are updated at the n users, respectively, and y_0, λ_0, z are updated at the provider. $\rho > 0$ is called the penalty parameter.

Table I shows the convergence iterations, total execution time and the achieved final expected social welfare under Algorithm 1, ADMM, and consistency pricing for resource allocation experiment in the first time period (period 1700). The experiment is conducted on a single machine of 2.6 GHz Intel Core i7 processor. We can see that Algorithm 1 converges in 5-9 iterations, ADMM converges in 3-17 iterations and consistency pricing converges in 27-52 iterations.

Although ADMM may take fewer iterations to converge under a good choice of ρ, it incurs much longer execution

TABLE I
PERFORMANCE COMPARISON OF OUR ALGORITHM, ADMM AND CONSISTENCY PRICING FOR RESOURCE ALLOCATION IN THE TIME PERIOD 1700.

Method	Algorithm 1			ADMM					Consistency Pricing	
Parameter	$\gamma=0.5$	$\gamma=0.75$	$\gamma=1$	$\rho=0.5$	$\rho=1$	$\rho=10$	$\rho=50$	$\rho=100$	$\gamma=0.01$	$\gamma=0.02$
Convergence Iterations	9	6	5	17	12	4	3	3	27	52
Total Execution Time (sec)	4.60	4.41	4.31	876.95	424.80	313.62	297.17	507.84	229.76	503.66
Final Expected Social Welfare	1378.9	1379.5	1379.5	1377.5	1378.3	1378.8	1378.7	1378.8	1344.1	1348.2

times, *which are about* $100\times$ *worse than Algorithm 1*. The reason is that in Algorithm 1, the only optimization is the single-variable benefit maximization performed at each user. In contrast, in each iteration of ADMM (44), not only each user needs to perform a single-variable optimization over $y_i \in \Re$, the provider also has to update $y_0 \in \mathbb{R}^n$ and $z \in \mathbb{R}^n$ by solving a convex optimization with vector variables. When n is as large as 468, it is mainly the optimization for $y_0 \in \mathbb{R}^n$ and $z \in \mathbb{R}^n$ that slows down the program. As mentioned in Sec. VII-B, consistency pricing incurs relatively long execution time as well because the provider also needs to solve a 468-dimensional optimization problem. However, it still takes shorter execution times than ADMM, because its optimization steps do not involve squared norms as in (44). Furthermore, Algorithm 1 achieves a higher final objective value, i.e., the final expected social welfare, than both ADMM and consistency pricing, with consistency pricing being the worst. Thus, the proposed algorithm is also better in terms of numerical accuracy.

VIII. CONCLUDING REMARKS

In this paper, we propose a fast fixed-point-like approach to solve resource sharing problems which are often formulated as utility maximization with a coupled objective function, and analyze the asynchronous algorithm convergence conditions through contraction mapping. In a case study of real-time cloud network reservation, we show through trace-driven simulations that the proposed method can speed up convergence by 5 times over the "consistency pricing" approach, which is a traditional gradient method applied with Lagrangian dual decomposition, while achieving better robustness to parameter configurations. The new approach converges in a similar number of iterations comparable to ADMM while incurring much lower computational complexity in the subproblem solved at each individual node. We also extend the proposed approach to solve a more general class of consensus optimization problems with arbitrary coupling in the objective function as well as a certain class of coupled constraints.

APPENDIX A
PROOFS

Proof of Proposition 4: For each i, define a function $g_i : [0,1] \mapsto \Re$ by

$$g_i(t) = (1-\gamma)(tp_i+(1-t)q_i)+\gamma u_i\big(x_i(tp+(1-t)q)\big). \quad (45)$$

Similarly, we have the following bound:

$$|T_i(p) - T_i(q)| = |g_i(1) - g_i(0)| \le \max_{t\in[0,1]} \left|\frac{\mathrm{d}g_i(t)}{\mathrm{d}t}\right|,$$

Similar to the proof of Proposition 3, it suffices to bound $|\mathrm{d}g_i(t)/\mathrm{d}t|$. Let $r = tp + (1-t)q$. We have

$$\begin{aligned}\left|\frac{\mathrm{d}g_i(t)}{\mathrm{d}t}\right| &= \Big|(1-\gamma)(p_i-q_i) + \gamma\textstyle\sum_j \nabla_j u_i \circ x_i\big(r(t)\big)(p_j-q_j)\Big| \\ &= \Big|\big(1-\gamma+\gamma u_i'(x_i(r))\nabla_i x_i(r)\big)\cdot(p_i-q_i) \\ &\quad +\gamma u_i'(x_i(r))\cdot\textstyle\sum_{j\neq i}\nabla_j x_i(r)(p_j-q_j)\Big| \\ &\le \max_j |p_j-q_j|\cdot\Big(|1-\gamma+\gamma u_i'(x_i(r))\nabla_i x_i(r)| + \\ &\quad +\gamma|u_i'(x_i(r))|\cdot\textstyle\sum_{j\neq i}|\nabla_j x_i(r)|\Big) \end{aligned} \quad (46)$$

If the following condition holds:

$$\begin{cases} \gamma = 1, \\ \sum_{j=1}^n |\nabla_j x_i(r)| < 1/|u_i'(x_i(r))|, \quad \forall r,\ \forall i, \end{cases} \quad (47)$$

or the following condition holds:

$$\begin{cases} 0 < \gamma \le (1+\nabla_i x_i(r)\cdot|u_i'(x_i(r))|)^{-1}, \\ \sum_{j\neq i}|\nabla_j x_i(r)| < \nabla_i x_i(r) + |u_i'(x_i(r))|^{-1}, \end{cases} \quad \forall r,\ \forall i, \quad (48)$$

then there exists a constant $\alpha \in (0,1)$ such that

$$\left|\frac{\mathrm{d}g_i(t)}{\mathrm{d}t}\right| \le \alpha \max_j |p_j-q_j| = \alpha\|p-q\|_\infty,\ \forall t\in[0,1],\ \forall i,$$

or in other words, T is a contraction with modulus α.

We then derive $\nabla_j x_i(r)$ based on $C(x)$. Applying the first-order conditions to the definition of $x(r)$ in (26), we have $r_i - c_i(x(r)) = 0$, for all i. Thus, the chain rule yields

$$1 = \frac{\partial c_i(x(r))}{\partial r_i} = \sum_{l=1}^n \nabla_l c_i(x(r))\cdot\frac{\partial x_l(r)}{\partial r_i}, \quad \forall i,$$

$$0 = \frac{\partial c_i(x(r))}{\partial r_j} = \sum_{l=1}^n \nabla_l c_i(x(r))\cdot\frac{\partial x_l(r)}{\partial r_j}, \quad \forall i,j:\ i\neq j.$$

Therefore, we have

$$[\nabla_j x_i(r)]_{n\times n} = [\nabla_j c_i(x(r))]_{n\times n}^{-1} = [H(x(r))]^{-1},$$

or in other words, $\nabla_j x_i(r) = P_{ij}(x(r))$. Substituting $\nabla_j x_i(r)$ into (47) and (48) leads to (27) and (28), respectively. ■

Proof of Proposition 5: We notice that $T_i(p)$ in (7) can be written in the form $T_i(p) = p_i - \gamma f_i(p)$, where $f_i(p)$ is defined by $f_i(p) := y_i(p) - x_i(p)$, with f_i being continuously differentiable. By Proposition 1.11 of [11] (pp. 194), $T_i(p)$ is a contraction if

$$\begin{cases} 0 < \gamma \le 1/\nabla_i f_i(p), \\ \sum_{j\neq i}|\nabla_j f_i(p)| < \nabla_i f_i(p), \end{cases} \quad \forall p,\ \forall i, \quad (49)$$

Similar to Proposition 4, We can derive $\nabla_j f_i(p)$ from the Hessian matrix of $C(x)$ and $U(x)$. In particular, we have

$$\nabla_i f_i(p) = \nabla_i y_i(p) - \nabla_i x_i(p) = P_{ii}(x(p)) - \left(U_i''(x_i(p))\right)^{-1}$$
$$\nabla_j f_i(p) = \nabla_j y_i(p) - \nabla_j x_i(p) = P_{ij}(x(p)), \quad \forall j \neq i.$$

Substituting the above into (49) and utilizing the fact $U_i''(x_i) < 0$ will prove the proposition. ■

Proof of Proposition 6: We use the following lemma, whose proof can be found in [11] (pp. 210, Proposition 3.1):

Lemma 7. *Suppose F is convex on a convex set X. Then, $z \in X$ minimizes F over X, if and only if $(y-z)^{\mathsf{T}} \nabla F(z) \geq 0$ for every $y \in X$.*

Suppose Algorithm 1 has a fixed point $p^* = (p_1^*, \ldots, p_n^*)$ and an associated z^*. Then we have

$$\begin{cases} p_i^* = \nabla_i C(z^*), & \forall i, \\ z_i^* = \arg\max_{z_i \in \mathcal{C}_i} U_i(z_i) - (p_i^*)^{\mathsf{T}} z_i, & \forall i. \end{cases} \tag{50}$$

Since z_i^* maximizes $U_i(z_i) - (p_i^*)^{\mathsf{T}} z_i$ on $\mathcal{C}_i$, by Lemma 7,

$$(y_i - z_i^*)^{\mathsf{T}} \left(\nabla U_i(z_i^*) - p_i^*\right) \leq 0, \quad \forall y_i \in \mathcal{C}_i, \ \forall i.$$

Since $p_i^* = \nabla_i C(z^*)$, we have

$$\sum_i (y_i - z_i^*)^{\mathsf{T}} \left(\nabla U_i(z_i^*) - \nabla_i C(z^*)\right) \leq 0, \ \forall y \in \prod_i \mathcal{C}_i,$$

which means z^* solves (31) according to Lemma 7. In a similar way, we can prove Proposition 6 for Algorithm 2. ■

REFERENCES

[1] F. P. Kelly, A. K. Maulloo, and D. K. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research society*, pp. 237–252, 1998.

[2] S. Low, "A duality model of tcp and queue management algorithms," *Networking, IEEE/ACM Transactions on*, vol. 11, no. 4, pp. 525–536, Aug 2003.

[3] X. Lin, N. B. Shroff, and R. Srikant, "A Tutorial on Cross-Layer Optimization in Wireless Networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1452–1463, August 2006.

[4] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as Optimization Decomposition: A Mathematical Theory of Network Architectures," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 255–312, January 2007.

[5] D. P. Palomar and M. Chiang, "A Tutorial on Decomposition Methods for Network Utility Maximization," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1439–1451, August 2006.

[6] Y. Xu, C. Yu, J. Li, and Y. Liu, "Video telephony for end-consumers:measurement study of google+, ichat, and skype," in *Proc. of ACM conference on Internet measurement conference*, 2012.

[7] Containers-not virtual machines-are the future cloud. [Online]. Available: http://www.linuxjournal.com/

[8] Google runs all software in containers. [Online]. Available: http://www.enterprisetech.com/2014/05/28/google-runs-software-containers/

[9] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 22.

[10] C. W. Tan, D. P. Palomar, and M. Chiang, "Distributed Optimization of Coupled Systems With Applications to Network Utility Maximization," in *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing*, Toulouse, France, 2006.

[11] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[12] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[13] J. F. Mota, J. M. Xavier, P. M. Aguiar, and M. Puschel, "D-admm: A communication-efficient distributed algorithm for separable optimization," *Signal Processing, IEEE Transactions on*, vol. 61, no. 10, pp. 2718–2723.

[14] H. Xu, C. Feng, and B. Li, "Temperature aware workload management in geo-distributed datacenters," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX, 2013, pp. 303–314.

[15] D. Zennaro, E. Dall'Anese, T. Erseghe, and L. Vangelista, "Fast clock synchronization in wireless sensor networks via admm-based consensus," in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2011 International Symposium on*, May 2011, pp. 148–153.

[16] F. P. Kelly, "Charging and Rate Control for Elastic Traffic," *European Transactions on Telecommunications*, vol. 8, pp. 33–37, 1997.

[17] R. Johari and J. N. Tsitsiklis, "Efficiency Loss in a Network Resource Allocation Game," *Mathematics of Operations Research*, vol. 29, no. 3, pp. 407–435, August 2004.

[18] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar, *Convex Analysis and Optimization*. Athena Scientific, 2003.

[19] D. Niu, C. Feng, and B. Li, "Pricing Cloud Bandwidth Reservations under Demand Uncertainty," in *Proc. of ACM SIGMETRICS*, 2012.

[20] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: a Data Center Network Virtualization Architecture with Bandwidth Guarantees," in *Proc. of ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2010.

[21] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards Predictable Datacenter Networks," in *Proc. of SIGCOMM'11*, Toronto, ON, Canada, 2011.

[22] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers," in *Proc. of ACM SIGCOMM*, 2012.

[23] D. Niu, Z. Liu, B. Li, and S. Zhao, "Demand Forecast and Performance Prediction in Peer-Assisted On-Demand Streaming Systems," in *Proc. of IEEE INFOCOM Mini-Conference*, 2011.

Di Niu received the B.Engr. degree from the Department of Electronics and Communications Engineering, Sun Yat-sen University, China, in 2005 and the M.A.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, in 2009 and 2013. Since 2012, he has been with the Department of Electrical and Computer Engineering at the University of Alberta, where he is currently an Assistant Professor. His research interests span the areas of cloud computing and storage, multimedia delivery systems, data mining and statistical machine learning for social and economic computing, distributed and parallel computing, and network coding. He is a member of IEEE and ACM.

Baochun Li received his B.Engr. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and his M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. He holds the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include cloud computing, multimedia systems, applications of network coding, and wireless networks. He was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000, the Multimedia Communications Best Paper Award from the IEEE Communications Society in 2009, and the University of Toronto McLean Award in 2009. He is a member of ACM and a Fellow of IEEE.