Cloud-Based Optimization: A Quasi-Decentralized Approach to Multi-Agent Coordination

M.T. Hale and M. Egerstedt†

***Abstract*— New architectures and algorithms are needed to reflect the mixture of local and global information that is available as multi-agent systems connect over the cloud. We present a novel architecture for multi-agent coordination where the cloud is assumed to be able to gather information from all agents, perform centralized computations, and disseminate the results in an intermittent manner. The cloud model accounts for delays in communications both when sending data to and receiving data from the agents. This architecture is used to solve a multi-agent optimization problem in which each agent has a local objective function unknown to the other agents and in which the agents are collectively subject to global constraints. Leveraging the cloud, a dual problem is formulated and solved by finding a saddle point of the associated Lagrangian.**

I. Introduction

Distributed optimization and algorithms have received significant attention during the last decade, e.g., [1], [20], [11], [8], [15], [22], [6], due to the emergence of a number of application domains in which individual decision makers have to collectively arrive at a decision in a distributed manner. Examples of these applications include communication networks [12], [5], sensor networks [13], [23], [2], multi-robot systems [26], [21], and smart power grids [3].

Distributed algorithms are needed mainly because the scale of large distributed systems is such that no central, global decision maker can collect all relevant information, perform all required computations, and then disseminate the results back to individual nodes in the network in a timely fashion. However, one can envision a scenario in which such globally obtained information can be used in conjunction with local computations performed across the network. This could, for example, be the case when a cloud computer is available to collect information, as was envisioned in [9]. The question then becomes that of designing the appropriate architecture and algorithms that can leverage this mix of prompt decentralized computations with delayed centralized computations.

One approach to multi-agent optimization that will prove useful towards achieving this hybrid architecture is based on primal-dual methods to find saddle points of a problem's Lagrangian [19], [7]. In fact, the study of saddle point dynamics in optimization can be traced back to earlier results from Uzawa in [24], which will provide the starting point for the work in this paper. The primary difference between this paper and the established literature is the cloud-based architecture used to solve the problem; indeed the architecture is this paper's main contribution. The architecture we introduce uses a cloud computer in order to receive information from each agent, perform global computations, and transmit this information to other agents so that they can update the information that they have stored locally and use it in their own local computations. We will see that this division of labor results in globally asymptotic convergence to Lagrangian saddle points.

The goal of this paper is to serve as a first attempt at understanding how centralized, cloud-based information might be injected in a delayed but useful manner into a network of agents where such information would otherwise be absent. In order to highlight how cloud-based, centralized information might prove useful to such a system, we choose to consider an extreme case where no inter-agent communication occurs at all, in contrast to existing distributed multi-agent optimization techniques, e.g., [14], [16], [17], [18]. Under this architecture the cloud handles all communications, and computations are divided between the cloud and the agents in the network.

The rest of the paper is organized as follows. Section II gives a detailed problem statement and describes the cloud architecture, and then Section III provides the convergence analysis for the given problem. Next, Section IV provides numerical results to attest to the viability of this approach, and finally Section V concludes the paper.

II. Problem Statement and Architecture

We now explain the interplay between the cloud architecture and the problem under consideration here. A detailed explanation is given below, with a summary and example following at the end of this section . Consider a collection of N agents indexed by $i \in A$, $A = \{1, \dots, N\}$, where each agent is associated with a scalar state $x_i \in \mathbb{R}$ and where there is no communication at all between the agents. Let the task agent i is trying to solve be encoded in a strictly convex objective function, $f_i(x_i) \in C^2$. We assume that $\frac{\partial f_i}{\partial x_i} \in o(x_i)$ for all $i \in A$, i.e.,

$$\lim_{x_i \to 0} \frac{x_i}{\frac{\partial f_i}{\partial x_i}(x_i)} = \infty. \tag{1}$$

This condition will be used later to determine the step-size used by all agents and the cloud when performing gradient descent-based optimization. The condition on the order of $\frac{\partial f_i}{\partial x_i}$ is not excessively restrictive since f_i is a design parameter selected by the user. In particular, the assumption on the order of $\frac{\partial f_i}{\partial x_i}$ is simply a constraint, like

†The authors are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA. Email: {matthale, magnus}@gatech.edu. Research supported in part by the NSF under Grant CNS-1239225.

strict convexity, imposed upon the choices that the user can make for objective functions, and this restriction allows a broad class of functions to be used for each f_i, e.g., strictly convex polynomials of degree greater than two.

The only goal of each agent is to minimize its own objective function. Agent i is assumed to have immediate access to its own state, which seemingly makes this problem very simple. However, what prevents agent i from simply computing $\frac{df_i}{dx_i}$ and setting this equal to zero – a completely decentralized operation as f_i only depends on x_i – is that the agents need to coordinate their actions through a globally defined constraint, that can, for example, represent finite resources that must be shared across the team. But in this paper we assume that agent i cannot measure the state of any other agents directly and, as mentioned above, that there is no communication between agents. Instead, this information must be obtained in some other manner, which is where the cloud enters into the picture.

The team-wide coordination is encoded through the global constraint

$$g(x) = \begin{pmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_m(x) \end{pmatrix} \leq 0, \tag{2}$$

where $x = (x_1, \ldots, x_N)^T$ is a state vector containing all the individual states. It is further assumed that each $g_j(x) \in C^2$ is convex and that each $\frac{\partial g_j}{\partial x_i} \in o(x_i)$ if g_j is a function of x_i, with $o(x_i)$ being used in the same sense as above for f_i. With g_j, as above with f_i, the assumption that $\frac{\partial g_j}{\partial x_i} \in o(x_i)$ is merely a limit placed upon what constraint functions a user can select when designing a problem. As above, we note that this is not a very harsh restriction as it merely guides the design process of the user when selecting constraint functions and it admits a broad collection of functions to be used. This assumption on the order of $\frac{\partial g_j}{\partial x_i}$ will also be used in determining stepsizes. The cloud architecture discussed here applies to any problem in which the user has selected functions f_i and g_j that meet the above criteria and the forthcoming analysis fully characterizes all such problem formulations.

Let

$$F(x) = \sum_{i=1}^{N} f_i(x_i). \tag{3}$$

Then F is strictly convex and the problem under consideration becomes that of minimizing F subject to g. The Kuhn-Tucker Theorem on concave programming (e.g., [25]) states that the optimum of this constrained problem is a saddle point of the Lagrangian $L(x, \mu) = F(x) + \mu^T g(x)$, where $\mu_j \geq 0$ for all $j \in \{1, \ldots, m\}$. We assume that the minimizer of L with respect to x, denoted $\hat{x}$, is a regular point of g so that there is a unique saddle point, $(\hat{x}, \hat{\mu})$, of L [4].

Using Uzawa's algorithm [24], the problem of finding $(\hat{x}, \hat{\mu})$ can be solved from the initial point $(x(0), \mu(0))$ using the difference equations

$$x(k) = x(k-1) - \rho \frac{\partial L}{\partial x}(x(k-1), \mu(k-1)) \tag{4}$$

$$\mu(k) = \max\left\{0,\ \mu(k-1) + \rho \frac{\partial L}{\partial \mu}(x(k-1))\right\} \tag{5}$$

where $\rho > 0$ is a constant, and where the maximum defining μ is taken component-wise so that each component of μ is projected onto the non-negative orthant of $\mathbb{R}^m$, denoted by $\mathbb{R}^m_+$. In the context of Uzawa's algorithm, the i^{th} element of the state vector x is updated according to

$$x_i(k) = x_i(k-1) - \rho \frac{\partial L}{\partial x_i}(x(k-1), \mu(k-1)). \tag{6}$$

Under the envisioned cloud-based organization of the agents and the lack of inter-agent communication, Uzawa's algorithm cannot be directly applied. To see this, observe that if agent i is to compute its own state update in (6), a fundamental problem is encountered: computing $\frac{\partial L}{\partial x_i}$ will require knowledge of states of (possibly all) other agents and agent i cannot directly access these. Furthermore, determining μ at each timestep using (5) will also require the full state vector x, which no single agent has direct access to.

To account for the need of each agent for global information in applying equation (6) and to compute μ using aggregated global information, the cloud computer is used. The cloud computer is taken to be capable of large batch computations, is subject to communications delays when sending and receiving information, and receives occasional (though not necessarily periodic) wireless transmissions from each agent containing each agent's own state.

The cloud computer uses the agents' states to compute the next value of μ using Equation (5) and then transmits the states it received and the newly computed μ vector to each agent. Each agent then uses the information from the cloud to update its own state in the vein of (6). While the cloud is computing or handling transmissions, we assume that each agent continues to update its own state in the absence of updated information about μ or the other agents' states. Since the cloud only occasionally receives and sends information, the agents will necessarily receive (and use) "old" information, in the sense that not all information agent i uses to compute its state at time k will have been measured or computed at time $k-1$.

To formalize these ideas, say at some time k each agent transmits its state to the cloud and denote the full vector of states sent to the cloud at time k by $x^c(k)$, where the superscript c denotes "cloud". Some communication delay is incurred and the states reach the cloud at some time $k + p_k$, $p_k \in \mathbb{N}$ (recall that each agent continues to update its own state based on the "old" information it has onboard while these transmissions are taking time to reach the cloud; note also that each agent will likewise continue to update its own state while the cloud is performing computations and transmitting data back to the agents). The delay p_k is associated with each timestep k at which the agents transmit

their states to the cloud and is not assumed to be constant but instead changes with each transmission time k.

Within timestep $k + p_k$, the cloud computes the next μ value, $\mu(k + p_k)$, which is a function of $x^c(k)$ and the previously computed μ value (from some timestep before k). At timestep $k+p_k+1$ the cloud begins to transmit $\mu(k+p_k)$ to each agent and for notational convenience we denote by $\boldsymbol{\mu}^i$ the most recent μ vector sent from the cloud to agent i.

At timestep $k + p_k + 1$ the cloud also begins to transmit to agent i the vector $\boldsymbol{y}^i \in \mathbb{R}^{N-1}$ defined as

$$\boldsymbol{y}^i = \begin{pmatrix} x_1^c(k) \\ \vdots \\ x_{i-1}^c(k) \\ x_{i+1}^c(k) \\ \vdots \\ x_N^c(k) \end{pmatrix}. \tag{7}$$

This vector contains states stored by the cloud in the vector x^c and contains information originally from time k. The subscripts in (7) denote that agent i does not receive its own old state value from the cloud, which is logical since agent i can sense its own most recent state. In $\boldsymbol{y}^i$, then, the cloud sends to agent i the most recent state information it has about each *other* agent. In this notation, agent j's state in $\boldsymbol{y}^i$ is denoted $\boldsymbol{y}^i_j$. In the forthcoming analysis, $\boldsymbol{y}^i$ always refers to the most recent state information that agent i has received from the cloud. It will not be written as an explicit function of any time step. Similarly, we use the notation $\boldsymbol{z}^i$ to denote the most recent transmission to agent i containing both $\boldsymbol{y}^i$ and $\boldsymbol{\mu}^i$.

Under this model, the transmissions back to the agents are received by the agents at time $k + n_k - 1$ for some $n_k \in \mathbb{N}$ with $n_k > p_k$; as with p_k, we associate a delay n_k with each timestep k at which the agents send their states to the cloud and we let n_k vary with each such k. We assume that $n_k \leq n_{max}$ for some $n_{max} \in \mathbb{N}$. Note that the condition $n_k > p_k$ does not mean that the delay when the cloud sends data to the agents is greater than the delay when the agents send data to the cloud. Instead, the condition $n_k > p_k$ enforces that the transmission from the cloud back to the agents occur after the transmission from the agents to the cloud.

Then in timestep $k + n_k$, agent i computes a state update using its most recent state and the "old" states of the other agents it has received from the cloud. Also within timestep $k + n_k$ each agent begins transmitting its state to the cloud and the cycle of transmission and computation is repeated. Throughout this process it is only assumed that agent i has the information necessary to compute f_i, g and their derivatives with respect to its own state variable. In particular, a given agent does not need to know any other agent's objective function. It is important to note that communications cycles do not overlap and that the agents *do not* send their states to the cloud at every timestep.

Due to the nature of the communications delays present in the system, updating the value of μ in the cloud does not occur at the same time that the agents receive information from the cloud. This means that by the time the cloud is computing an update for the μ vector, the states it is using to do so are already "old;" this is reflected in the modified μ update equation below.

Along these lines, each agent can only sense its own state and relies on the cloud for other agents' states. Then each agent always has current knowledge of its own state and "outdated" knowledge of the other agents' states. Accordingly, there is no reason to expect that two agents, i and j, will agree on agent i's current state. To model such differences, equation (4) is modified to reflect that each agent stores and manipulates a local copy of the problem. Each local copy receives occasional updates from the cloud, and the problem of agent i has a state vector, denoted $x^i(k)$ at time k, that is assumed to be different from $x^j(k)$ when $i \neq j$.

Using the fact that agent i will update its own state even when it does not receive an update from the cloud, equation (4) is modified so that onboard agent i it is

$$x^i(k) = \begin{cases} \bar{\boldsymbol{y}}^i - \rho \nabla^i L(\bar{\boldsymbol{y}}^i, \boldsymbol{\mu}^i) & \boldsymbol{z}^i \text{ received at time } k-1 \quad (8a) \\ x^i(k-1) - \rho \nabla^i L\big(x^i(k-1), \mu^i(k-1)\big) & \text{else,} \quad (8b) \end{cases}$$

where we define

$$\nabla^i L(\bar{\boldsymbol{y}}^i, \boldsymbol{\mu}^i) = \begin{pmatrix} 0 \\ \vdots \\ \frac{df_i}{dx_i}(x_i^i(k-1)) + (\boldsymbol{\mu}^i)^T \frac{\partial g}{\partial x_i}(\bar{\boldsymbol{y}}^i) \\ \vdots \\ 0 \end{pmatrix}, \tag{9}$$

and where $\nabla^i L\big(x^i(k-1), \mu^i(k-1)\big)$ is defined similarly. Here, $\bar{\boldsymbol{y}}^i$ is defined as a vector onboard agent i which contains $\boldsymbol{y}^i$ and the most recent state of agent i inserted in the appropriate place. In essence, $\bar{\boldsymbol{y}}^i$ is the most up-to-date information about all of the agents that agent i has access to. Note that $\nabla^i L$ is simply $\frac{\partial L}{\partial x}$ with all entries except the i^{th} set to 0. This is because agent i does not itself compute any updates for the other agents' states which it stores onboard, but instead waits for the cloud to provide such updates.

Under the architecture of this problem, only the cloud computes values of μ, and there is only a single update equation needed for μ. Using the symbols p_k and n_k as above (and remembering that they are not constant but instead vary with each communication cycle), equation (5) is modified to account for delays and takes the form

$$\mu(k+p_k) = \max\left\{0,\ \mu(k+p_{k'}-n_{k'}) + \rho \frac{\partial L}{\partial \mu}(x^c(k))\right\} \tag{10}$$

where k' is the most recent time at which the agents sent their states to the cloud before k and $k + p_{k'} - n_{k'}$ is the last timestep in which μ was computed in the cloud.

We note that equation (10) is not indexed on a per-agent basis since only the cloud computes values of μ. However, we will use the notation $\mu^i(k)$ to denote the μ vector stored on agent i at time k (which may be different from the μ vector stored in the cloud at time k). It is important to note

that the argument of $\mu^i(k)$ is intended to reflect at what time agent i has μ^i onboard and *does not* imply that μ^i was computed at time k or that agent i computed it. In this notation $\boldsymbol{\mu}^i$ represents the μ vector most recently sent from the cloud to agent i, while μ^i represents the μ vector stored on agent i.

The timestep on the left-hand side is $k + p_k$ in (10) to emphasize the fact that μ is not updated at the same times as the vectors x^i are. With this model (and associated caveats) in mind, instead of considering the system defined in equations (4) and (5), we consider N copies of the system defined by (8) and (10). Returning to the notation that $\mu^i(k)$ represents the vector μ as stored on agent i at time k, we can write the full update equations onboard agent i as

$$x^i(k)=\begin{cases}\bar{\boldsymbol{y}}^i - \rho\nabla^i L(\bar{\boldsymbol{y}}^i, \boldsymbol{\mu}^i) & \boldsymbol{z}^i \text{ received at time } k-1 \quad (11a)\\ x^i(k-1) - \rho\nabla^i L\big(x^i(k-1), \mu^i(k-1)\big) & \text{else,} \quad (11b)\end{cases}$$

$$\mu^i(k) = \begin{cases}\boldsymbol{\mu}^i & \boldsymbol{z}^i \text{ was received at time } k-1 \quad (12a)\\ \mu^i(k-1) & \text{else.} \quad (12b)\end{cases}$$

To illustrate the communications cycle described above, Table 1 contains a sample schedule for a single cycle with $p_k = 2$ and $n_k = 5$. Each timestep is listed on the left and the corresponding actions taken at that timestep are listed on the right. Note that the same actions are taken at timesteps k and $k+5$, though $k+5$ is explained in greater detail because it has the context of preceding timesteps available to aid in its explanation.

Timestep	Actions
k	Each agent takes 1 step in its own copy of the problem to update (only) its own state. This is computed using the cloud update that was just received. Then each agent sends its own resulting state to the cloud.
$k+1$	All agents take 1 step in their own copy of the problem to update their own states using the information stored onboard (originally from timestep k).
$k+2$	The cloud receives the transmission from time k and stores it in $x^c(k)$. It then computes the updated μ vector. All agents take 1 step in their own copy of the problem to update their own states using the information stored onboard (originally from timestep k).
$k+3$	The cloud now begins to send to each agent the new μ and each other agent's state originally from time k. For agent i, these become $\boldsymbol{\mu}^i$ and $\boldsymbol{y}^i$. All agents take 1 step in their own copy of the problem to update their own states using the information stored onboard (originally from timestep k).
$k+4$	All agents take 1 step in their own copy of the problem to update their own states using the information stored onboard (originally from timestep k). Then the agents receive the latest cloud transmission.
$k+5$	Agent i replaces its onboard μ vector with $\boldsymbol{\mu}^i$ and replaces all onboard states except its own with $\boldsymbol{y}^i$, thereby forming $\bar{\boldsymbol{y}}^i$. The agents each take 1 step in their own onboard copy of the problem to update (only) their own states, and then send their own resulting state to the cloud.

Table 1: A sample schedule for one communications cycle used by the agents and cloud to exchange information.

To show that the agents' states converge to the (unique) saddle point of L, we must show that the local problem stored by each agent converges to the point $(\hat{x}, \hat{\mu})$.

III. Convergence

The results in this section are presented without proof due to space constraints. Full proofs of the results here can be found in [10]. Before stating the main convergence result, we provide the following lemma which establishes a positive upper bound on the stepsizes that can be used.

Lemma 1: Let the constant ρ_{max} be defined as

$$\rho_{max} = \min\left\{ \frac{(x-\hat{x})\cdot\nabla^i L(x,\mu)}{\|\nabla^i L(x,\mu)\|^2} \;\middle|\; V(x,\mu) \le K \right\} \quad (13)$$

$$= \min\left\{ \frac{x_i - \hat{x}_i}{\frac{\partial L}{\partial x_i}(x,\mu)} \;\middle|\; V(x,\mu) \le K \right\}. \quad (14)$$

Then $\rho_{max} > 0$.

Proof: Please see [10]. ■

Using Lemma 1, we now state the main result of the paper.

Theorem 1: Let every agent use a strictly convex objective function $f_i : \mathbb{R} \to \mathbb{R}$, $f_i \in C^2$ and let the global constraints, $g : \mathbb{R}^N \to \mathbb{R}^m$, be convex with $g_j \in C^2$ for each j. Let $\frac{\partial f_i}{\partial x_i} \in o(x_i)$ and let $\frac{\partial g_j}{\partial x_i} \in o(x_i)$ whenever g_j is a function of x_i. Then for any stepsize ρ such that $0 < \rho \le \rho_{max}$ used by all agents and the cloud, the saddle point $\hat{z}$ is globally asymptotically stable in all N copies of the system defined in equations (11) and (12).

Proof: Please see [10]. ■

IV. Simulation Results

A numerical implementation of the above cloud architecture was run for a particular choice of simulation example. Then problem simulated was chosen to use $N = 6$ agents, each associated with a scalar state as above. The objective function of each agent was chosen to be $f_i(x_i) = (x_i - \tilde{x}_i)^4$, where

$$\tilde{x} = \begin{pmatrix} -3.0 \\ 6.0 \\ -5.0 \\ 4.0 \\ 2.0 \\ -6.0 \end{pmatrix}. \quad (15)$$

Since each objective function is quartic, it satisfies all assumptions required by the architecture of the problem. The constraints in this problem were chosen to be

$$g(x) = \begin{pmatrix} x_1^4 + x_4^4 - 50 \\ x_2^4 + x_5^6 - 100 \\ x_3^6 + x_6^4 - 100 \end{pmatrix} \le 0. \quad (16)$$

These will provide active inequality constraints, thus causing the constrained optimum of each agent to differ from the unconstrained optimum. The Lagrangian of this problem is

$$L(x,\mu) = \sum_{i=1}^{6} f_i(x_i) + \mu^T g(x) \quad (17)$$

where $\mu \in \mathbb{R}^3_+$.

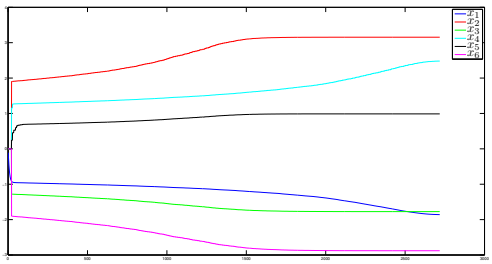

Fig. 1. The states onboard agent 1 over time. Note that each state monotonically approaches its final value. Since this is a gradient-based method, we see larger changes in earlier iterations, followed by smaller steps taken at later iterations.

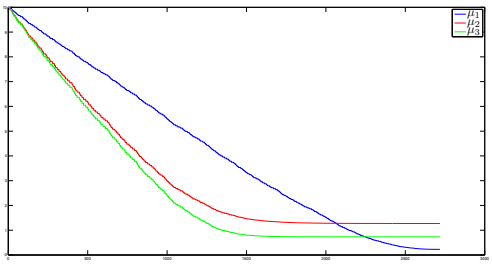

Fig. 2. The Kuhn-Tucker multipliers onboard agent 1 over time. As with the states, each Kuhn-Tucker multiplier monotonically approaches its final value, with larger steps generally coming earlier in the time-evolution of the problem because they are computed using a gradient-based method.

For this simulation each n_k is a randomly-generated integer between 5 and 15 for each transmission time k. The stepsize used was $\rho = 0.002$. The gradient descent algorithm described above was initialized with all agents and the cloud having all states set to 0. All agents and the cloud had all Kuhn-Tucker multipliers initialized to 10 because we anticipate these constraints being active. The algorithm was run until at some time k the condition

$$\left\| \frac{\partial L}{\partial x}\left(x^c(k), \mu(k+p_k)\right) \right\| + \left| \mu(k+p_k)^T g(x^c(k)) \right| \leq \epsilon \tag{18}$$

was satisfied in the cloud. Here we set $\epsilon = 0.3$.

The points $\hat{x}$ and $\hat{\mu}$ were computed ahead of time to be

$$\hat{x} = \begin{pmatrix} -1.9596 \\ 3.1533 \\ -1.7758 \\ 2.4367 \\ 0.9937 \\ -2.8739 \end{pmatrix} \tag{19}$$

and

$$\hat{\mu} = \begin{pmatrix} 0.2397 \\ 1.2749 \\ 0.7357 \end{pmatrix}. \tag{20}$$

Here, the cloud algorithm took $1,479$ iterations to run and resulted in a final $x^c(k)$ of

$$x^c = \begin{pmatrix} -1.8594 \\ 3.1549 \\ -1.7744 \\ 2.4809 \\ 0.9881 \\ -2.8798 \end{pmatrix} \tag{21}$$

and a final $\mu^c(k)$ of

$$\mu^c = \begin{pmatrix} 0.2266 \\ 1.2718 \\ 0.7334 \end{pmatrix}. \tag{22}$$

The final value of V in the cloud was $V(x^c, \mu^c) = 0.0126$. Based on the definition of V, this means that the square of the Euclidean distance from $(x^c(1,479), \mu^c(1,479))$ to $(\hat{x}, \hat{\mu})$ is just 0.0126, indicating very close convergence of this algorithm to the unique saddle point.

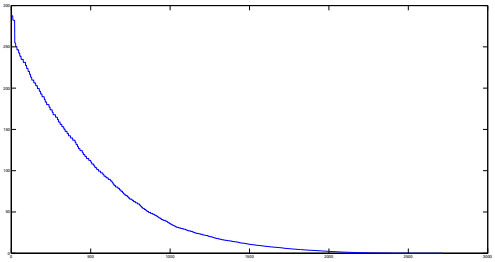

Fig. 3. The value of $V(x^1(k), \mu^1(k))$ over time. As expected from the Lyapunov analysis in Section III, the Lyapunov function is generally decreasing with potential periods of constancy seen when a particular delay is long. These periods of constancy terminate when a cloud update is received and V continues to decrease until reaching its final value.

Though approximately $1,500$ iterations is perhaps more than a centralized algorithm would require, a longer convergence time such as this one is expected given the structure of the problem. Specifically, there are non-negligible communications delays present in this system and delayed communications certainly cause a longer time until convergence. Moreover, when an agent does receive information, it necessarily receives old information about the rest of the system. Then, since each agent receives old information and receives it late, the convergence of this problem will take more time than an algorithm that does not have delays of any kind.

To further illustrate the convergence of this problem the histories of the states, Kuhn-Tucker multipliers, and value of V over time onboard agent 1 are shown in Figures 1, 2, and 3, respectively. In each plot, the sequences of states, Kuhn-Tucker multipliers, and $V^1(k)$ values are all monotone as they approach their final values. This is expected given the globally asymptotically stable nature of the saddle point. In addition, larger decreases in the distance to the saddle point are seen in earlier iterations, a common characteristic of gradient descent algorithms.

V. Conclusion

We presented a cloud architecture for coordinating a team of mobile agents in a distributed optimization task. Each agent has direct knowledge only of its own local objective function and its own influence upon the global constraint functions but receives occasional updates from the cloud computer containing old values of each other agent's state and updated Kuhn-Tucker multipliers. Global asymptotic convergence of this system was proven using a radially unbounded Lyapunov function that was shown to be non-increasing along trajectories. Simulation results were provided to attest to the viability of this approach.

References

[1] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[2] L. Carlone, V. Srivastava, F. Bullo, and G. C. Calafiore. Distributed random convex programming via constraints consensus. 52(1):629–662, 2014.

[3] S. Caron and G. Kesidis. Incentive-based energy consumption scheduling algorithms for the smart grid. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pages 391–396, Oct 2010.

[4] Benoit Chachuat. Nonlinear and dynamic optimization: From theory to practice. Technical report, Automatic Control Laboratory, EPFL, Switzerland, 2007.

[5] Mung 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*, 95(1):255–312, Jan 2007.

[6] Greg Droge and Magnus Egerstedt. Proportional integral distributed optimization for dynamic network topologies. In *IEEE American Control Conference (ACC), 2014*, June 2014.

[7] Diego Feijer and Fernando Paganini. Stability of primal-dual gradient dynamics and applications to network optimization. *Automatica*, 46(12):1974–1981, December 2010.

[8] B. Gharesifard and J. Cortes. Distributed continuous-time convex optimization on weight-balanced digraphs. *Automatic Control, IEEE Transactions on*, 59(3):781–786, March 2014.

[9] Ken Goldberg and Ben Kehoe. Cloud robotics and automation: A survey of related work. Technical Report UCB/EECS-2013-5, EECS Department, University of California, Berkeley, Jan 2013.

[10] M.T. Hale and M. Egerstedt. Cloud-based optimization: A quasi-decentralized approach to multi-agent coordination. Technical Memorandum, Georgia Institute of Technology, 2014. Available at http://arxiv.org/abs/1404.0098.

[11] Bruce Hendrickson and Tamara G. Kolda. Graph partitioning models for parallel computing. *Parallel Comput.*, 26(12):1519–1534, November 2000.

[12] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. In *Journal of the Operational Research Society*, volume 49, 1998.

[13] M. Khan, G. Pandurangan, and V.S.A. Kumar. Distributed algorithms for constructing approximate minimum spanning trees in wireless sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 20(1):124–139, Jan 2009.

[14] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on*, 54(1):48–61, Jan 2009.

[15] Angelia Nedic and Alex Olshevsky. Distributed optimization over time-varying directed graphs. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 6855–6860, Dec 2013.

[16] Angelia Nedic and Asuman Ozdaglar. On the rate of convergence of distributed subgradient methods for multi-agent optimization. In *Proceedings of IEEE CDC*, pages 4711–4716, 2007.

[17] Angelia Nedić and Asuman Ozdaglar. Convergence rate for consensus with delays. *Journal of Global Optimization*, 47(3):437–456, 2010.

[18] Angelia Nedic, Asuman Ozdaglar, and Pablo A Parrilo. Constrained consensus and optimization in multi-agent networks. *Automatic Control, IEEE Transactions on*, 55(4):922–938, 2010.

[19] Sikandar Samar, Stephen Boyd, and Dimitry Gorinevsky. Distributed estimation via dual decomposition. In *Proc. European Control Conference*, pages 1511–1519, 2007.

[20] H.D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(23):135 – 148, 1991. Parallel Methods on Large-scale Structural Analysis and Physics Applications.

[21] Daniel E Soltero, Mac Schwager, and Daniela Rus. Decentralized path planning for coverage tasks using gradient descent adaptive control. *The International Journal of Robotics Research*, 2013.

[22] Andre Teixeira, Euhanna Ghadimi, Iman Shames, Henrik Sandberg, and Mikael Johansson. Optimal scaling of the admm algorithm for distributed quadratic programming. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 6868–6873, Dec 2013.

[23] Niki Trigoni and Bhaskar Krishnamachari. Sensor network algorithms and applications Introduction. *Philosophical Transactions of the Royal Scoeity A - Mathematical, Physical, and Engineering Sciences*, 370(1958, SI):5–10, JAN 13 2012.

[24] H. Uzawa. Iterative methods in concave programming. *Studies in Linear and Non-Linear Programming*, 1958.

[25] H. Uzawa. The kuhn-tucker theorem in concave programming. *Studies in Linear and Non-Linear Programming*, 1958.

[26] Minyi Zhong and C.G. Cassandras. Asynchronous distributed optimization with event-driven communication. *Automatic Control, IEEE Transactions on*, 55(12):2735–2750, Dec 2010.