In-Network Analytics for Ubiquitous Sensing*

Ittay Eyal[1], Idit Keidar[2], Stacy Patterson[2], and Raphi Rom[2]

[1] Department of Computer Science, Cornell
ittay.eyal@cornell.edu
[2] Department of Electrical Engineering, Technion
{idish,stacyp, rom}@ee.technion.ac.il

Abstract. We address the problem of in-network analytics for data that is generated by sensors at the edge of the network. Specifically, we consider the problem of summarizing a continuous physical phenomenon, such as temperature or pollution, over a geographic region like a road network. Samples are collected by sensors placed alongside roads as well as in cars driving along them. We divide the region into sectors and find a summary for each sector, so that their union is a continuous function that minimizes some global error function. We designate a node (either virtual or physical) that is responsible for estimating the function in each sector. Each node computes its estimate based on the samples taken in its sector and information from adjacent nodes.

The algorithm works in networks with bounded, yet unknown, latencies. It accommodates the addition and removal of samples and the arrival and departure of nodes, and it converges to a globally optimal solution using only pairwise message exchanges between neighbors. The algorithm relies on a weakly-fair scheduler to implement these pairwise exchanges, and we present an implementation of such a scheduler. Our scheduler, which may be of independent interest, is *locally quiescent*, meaning that it only sends messages when required by the algorithm. It achieves quiescence on every link where the algorithm ceases to schedule pairwise exchanges; in particular, if the algorithm converges, it globally quiesces.

1 Introduction

As we enter the era of ubiquitous sensing, we have the opportunity to monitor the world around us with unprecedented resolution and to leverage this vast wealth of data to make our environment smarter. On-board sensors and computers in new vehicles can sense road and traffic conditions and use this information in

* This research was supported by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI), the Israeli Ministry of Trade and Labor Rescue Consortium, the Sidney Goldstein Research Fund, the Israeli Science Foundation, the Arlene & Arnold Goldstein Center at the Technion Autonomous Systems Program, a Technion Fellowship, two Andrew and Erna Finci Viterbi fellowships, the Lady Davis Fellowship Trust, and the Hasso-Plattner Institute for Software Systems Engineering.

Y. Afek (Ed.): DISC 2013, LNCS 8205, pp. 507–521, 2013.

route planning, smart meters enable fine-grained power usage monitoring and can assist in demand-response in the smart grid, and cheap wireless motes that measure noise, light, and air pollution can be used as input into urban planning and public health decisions.

To fully realize this vision, we must be able to process this massive amount of data and generate meaningful summaries that can be used in planning and response. The field of machine learning offers a range of tools for such data analytics, but these tools typically assume all data is present at a centralized location. As this data is generated at the edge of the network at an ever-increasing rate, transmitting the data over a long distance to a central facility is both expensive and energy consuming, especially in wireless networks [1]. Moreover, the high latency incurred by these long-distance transmissions may prove problematic to time-sensitive applications. Finally, even after collecting all data, processing it requires costly and time-consuming computation. Thus, we need distributed solutions for in-network data analytics.

A few very recent works in the field of control theory and machine learning have proposed distributed algorithms for data analytics, however, these works use a naïve model of the distributed system and thus do not offer a realistic solution for our setting. On the other hand, brute force distribution does not work. To develop realistic distributed data-analytics tools requires both an understanding of machine learning and distributed computing.

In this extended abstract, we present a distributed data analytics technique using a novel combined approach and illustrate it using a specific application of in-network analytics, motivated by our work with a major automotive corporation on *vehicular sensor networks*. The objective is to generate a compact estimate of a continuous physical phenomenon from samples measured throughout a region, for example, the road network in Western Europe. These samples are collected by vehicles driving through the region as well as by fixed infrastructure (e.g., roadside units) with compute, storage, and local communication capabilities. We leverage this fixed infrastructure as nodes in a distributed computing platform. This infrastructure may be unreliable, and so our solution must accommodate the arrival and departure of nodes. Moreover, the measured phenomenon changes over time, hence the estimate is generated from a dynamic set of samples. We detail our system model in Section 2.

From these requirements, we generate a formal problem definition for this application setting. We describe how we architect this formal problem in Section 3. Our approach is based on machine learning fundamentals, namely linear regression. Since the data is geographically distributed, care must be taken to ensure that this formalization is amenable to a distributed solution. To achieve this, we employ *selective* learning; each node learns an estimate for its local area, or *sector*, based on its samples and communication with its neighbors. (The division of the region and assignment of nodes can be done with known techniques [2,3,4,5] and is outside the scope of this paper.) We thus define an optimization problem where each node's estimate has to minimize some convex error function related to collected samples, while requiring that the union of these estimates is continuous

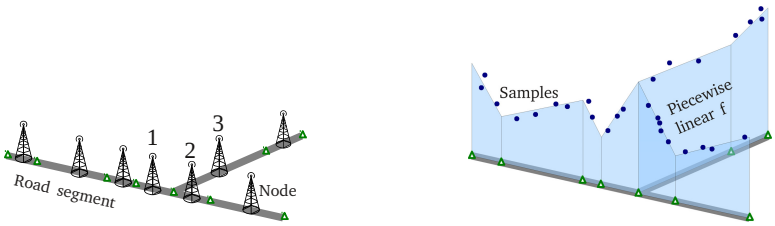
over the entire region. The continuity requirement stems from the fact the sampled phenomenon is known to be continuous. Generating the *global* estimate is a *convex optimization problem* whose objective is to minimize the sum of the local error functions, with equality constraints that match the structure of the network. The nodes' estimates should converge to a global optimal estimate once changes cease. Note that since nodes do not know when changes stop, they must make a best effort to converge at all times. This problem structure opens the door to a solution based on *local* communication. However, it still requires multi-way coordination between all nodes sharing a sector boundary (for details see the technical report [6]). To eliminate such costly coordination, we transform the problem to its dual form to obtain a decomposable (unconstrained) problem that can be solved using only pairwise coordination between neighbors. While transforming to the dual is a common optimization technique, we have not seen it used for this purpose before.

We then present, in Section 4, a novel, distributed optimization algorithm for our formal problem based on the method of coordinate ascent. In general, coordinate ascent is not amenable to distribution, and a naïve implementation requires global synchronization and does not accommodate dynamic behavior (see related work below). In contrast, our distributed coordinate ascent algorithm deals with dynamic inputs and requires neither global information nor synchronization.

The algorithm progresses in steps, and to schedule these steps in the distributed environment we implement a *locally quiescent weakly-fair scheduler*. This scheduler executes pairwise message exchanges in a weakly-fair manner and only when they are required by the algorithm. Unlike with standard synchronizer-based approaches, if the algorithm ceases to schedule pairwise exchanges (i.e., it reaches the optimum), the scheduler achieves quiescence. This scheduler, described in Section 5, may be important in and of itself where communication is expensive and relaxed scheduling is sufficient.

We believe that our approach to distributed, in-network analytics without global communication or synchronization can prove useful in many additional settings. Section 6 concludes the paper and touches on some directions for future research. Some formal details and more elaborate examples are given in the technical report [6]. While, in the sequel, we consider estimation over a road network, our approach can also be applied to estimation over a two-dimensional region, as detailed in the technical report.

Related Work. Previous work on distributed convex optimization can be divided into two categories: averaging-based algorithms (based on the framework of [7]), and sequential algorithms. Averaging-based algorithms have been proposed for unconstrained convex optimization problems [7,8,9], and for constrained convex optimization problems where all constraints are known globally [10,11] or where constraints are purely local [11]. To satisfy the continuity constraints in our problem formulation, these algorithms would require that all nodes know the continuity constraints for the entire network, which induces a prohibitive per node storage cost and a need for global information. In sequential algorithms [12,13], there is



(a) Roads and stations

(b) Piecewise linear estimation

Fig. 1. Piecewise linear estimation in a road system. We see a junction of three roads. (a) The roads are divided into sectors (e.g., 1, 2 and 3), each with its node. The sectors meet at vertices (triangles). (b) The samples are shown as dots, and a curtain above the roads shows the continuous piecewise linear estimate f.

a single copy of the optimization variables, and every node has its own objective function and constraints. Both approaches require storage for *all* variables at every node, which is infeasible for large networks of the type we consider (e.g., a road network of an entire country). Furthermore, none of these algorithms can tolerate node arrivals and departures.

In this work, we present a distributed optimization algorithm based on the method of coordinate ascent. Coordinate ascent is an iterative method where, in each step, a single variable is updated. The algorithm converges to the optimal solution only if the order in which the variables are updated obeys certain properties. A few recent works have proposed parallel implementations of coordinate descent for solving large optimization problems [14,15]. While the update operations occur in parallel, these implementations still require that updates are executed in globally specified order, thus requiring centralized coordination. They also require that the set of participating nodes does not change during the algorithm execution. In contrast, our distributed coordinate ascent algorithm requires no global information or coordination, and it operates correctly even if the set of participating nodes changes.

2 System Model

We consider a finite region, for example, a road network. Sensors collect *samples* in the region, each comprised of a geographic location and a value. There is also a computing network in the region consisting of a dynamic set of nodes. Each node may be part of a physical infrastructure, for example, a base station, or it may be a virtual node implemented by a dynamic set of mobile agents. Neighboring nodes are connected with bidirectional, reliable FIFO links.

We take a hierarchical approach and divide the region into a fixed set of non-overlapping *sectors*. In the road network, a sector is a segment of a road, and its end points are two vertices, as shown in Figure 1(a); a single vertex may be shared by several adjacent sectors. Each vertex has a unique ID. Each sector also has

a unique ID, and these IDs are totally ordered. Each sector is assigned at most one node at any given time, and this node maintains a dynamic set of samples that have been taken in its sector. Each node has a unique ID and it knows the ID of the sector for which it is responsible. We assume a fail-stop model, i.e., a node that fails does not recover, and its neighbors are notified of the failure. A new node may arrive and take responsibility for the failed node's sector. When a node becomes active, its sample set is initially empty. Following its activation, a node is notified of all its neighbors, and they all receive notifications of its arrival.

Notation. We employ the following notation throughout this work:

- The set of vertices for sector i is denoted $\mathcal{V}_i$, and the set of vertices for all sectors is denoted $\mathcal{V}$.
- The dynamic set[1] of active (non-failed) nodes is denoted $\mathcal{N}$.
- The set of active nodes whose sectors share a vertex v is called the *member set* of v, denoted $\mathcal{M}(v)$. We say that nodes i and j are *neighbors* if there exists a vertex v such that both i and j are elements of $\mathcal{M}(v)$.
- Each sample is represented by a tuple (x, z) where x is the distance along the segment from the vertex with the smaller ID, and z is the value of the sample at that location.
- The set of samples at node i is denoted $\mathcal{S}_i$. The dynamic set of all samples, denoted $\mathcal{S}$, is the union of the sample sets belonging to the nodes in $\mathcal{N}$.

Stabilization. For the sake of analysis, we assume that the system eventually stabilizes.

Definition 1 (Global Stabilization Time). *The* ***global stabilization time (GST)*** *is the earliest time after which the following properties hold: (1) no nodes are informed of neighbor changes, (2) samples are neither added nor deleted, and (3) the message latency of all outstanding and future messages is bounded between δ and Δ. Nodes do not know when GST has been reached, nor do they know δ or Δ.*

Before GST (if it exists), there are no time bounds on message latency or on notifications of neighbor set changes or sample set changes.

3 Architecting the Problem

In this section, we develop a formal problem definition that is compatible with the system model, generates a meaningful summary of the collected samples, and is amenable to a distributed solution that scales to the size of an immense road network. Each node generates an estimate for its sector that is optimal with respect to its samples, while also ensuring that the union of these local estimates is a piecewise continuous function over the region. The global estimation problem is defined by a convex optimization problem whose objective is to minimize

[1] The dynamic sets $\mathcal{N}$, $\mathcal{M}(v)$, $\mathcal{S}_i$, and $\mathcal{S}$ are all functions of time. Since the time can always be deduced from the context, we omit the superscript t.

the sum of the local error functions, with equality constraints that match the structure of the network. This convex optimization problem is detailed in Section 3.1. A distributed algorithm that addresses the problem directly requires expensive coordination among all nodes that share a sector boundary (for example, at an intersection). In Section 3.2 we transform the problem to its dual form and obtain a decomposable problem that can be solved using only pairwise communication between neighbors.

3.1 Formal Problem Definition

We define the estimate for sector i to be a real-valued function f_i over the sector. The global estimate f is the union of these individual estimates. We illustrate our algorithm for the case where each f_i is a linear function defined over a single-dimensional road segment. Such a function is shown in Figure 1(b); the estimate f is drawn as a curtain above the roads. It is straightforward to extend our approach to estimation with higher order polynomials, as shown in the technical report [6].

Let u and v be the IDs of the vertices of the sector belonging to node i, with $u < v$. The function f_i is parameterized by the values at its vertices, denoted $\theta_{i,u}$ and $\theta_{i,v}$, and is given by,

$$f_i(x; \theta_{i,u}, \theta_{i,v}) \triangleq (1 - x/d_i)\,\theta_u + (x/d_i)\,\theta_v.$$

Here, d_i is the length of sector i. Define $\theta_i \triangleq [\theta_{i,u}\ \theta_{i,v}]^\mathsf{T}$, and let θ denote the vector of all $\theta_{i,u}$ variables, $i \in \mathcal{N}$, $u \in \mathcal{V}_i$. Each node must generate an optimal estimate of its sector from its samples. Specifically, each node i must determine the values θ_i that minimize a convex error function. As an example, we consider the least squares error,

$$C_i(\theta_i; \mathcal{S}_i) \triangleq \sum_{(x,z)\in\mathcal{S}_i} (f_i(x;\theta_i) - z)^2\,.$$

We also require that the estimates f_i are continuous at the sector boundaries. This requirement means that nodes must collaborate to perform the estimation so that they agree on the values for shared vertices.

The problem of learning the function f after GST can be formulated as a convex optimization problem with a global, separable objective function and linear constraints that capture the continuity requirements,

$$\underset{\theta}{\text{minimize}}\ \ C(\theta; \mathcal{S}) \triangleq \sum_{i\in\mathcal{N}} C_i(\theta_i; \mathcal{S}_i) \tag{1}$$

$$\text{subject to}\ \ \theta_{i,v} = \theta_{j,v}, \quad \text{for } v \in \mathcal{V}, \quad i, j \in \mathcal{M}(v), i \neq j. \tag{2}$$

The constraints (2) state that every pair of nodes in $\mathcal{M}(v)$ has the same value for vertex v, or equivalently, all nodes in $\mathcal{M}(v)$ learn the same value for vertex v. These constraints ensure that the estimate is continuous within each connected component of the network of nodes in $\mathcal{N}$.

Our goal is to design a distributed algorithm for finding the values of the parameters θ that solve the optimization problem defined above. A node i knows only its own sample set $\mathcal{S}_i$ and communicates only with its neighbors. Each node is responsible for obtaining an estimate of its sector by learning values for θ_i. After GST, these estimates must converge to a globally optimal estimate.

In the problem (1)–(2), all members of a vertex must agree on the value of θ for that vertex. An algorithm that addresses the problem directly would require that all members coordinate to maintain this constraint. In the next section, we show that by transforming to the dual, we obtain a problem formulation that only requires coordination between pairs of neighboring nodes.

3.2 Problem Decomposition

We now show how to transform the constrained convex optimization into its unconstrained dual form. We note that, typically, one transforms a problem to its dual because the dual can be solved in a more computationally efficient manner in a centralized setting. Our use of the dual is unconventional; we use the dual problem because it opens the door to a distributed solution with reduced communication costs.

Given a constrained optimization problem, the dual problem is formed by defining the Lagrangian, where the constraints are incorporated into the objective function. The Lagrangian for (1)–(2) is,

$$\mathcal{L}(\theta, \lambda; \mathcal{S}) = \sum_{i \in \mathcal{N}} C_i(\theta_i; \mathcal{S}_i) + \sum_{v \in \mathcal{V}} \sum_{i,j \in \mathcal{M}(v), i \neq j} \lambda_{i,j}^v \left(\theta_{i,v} - \theta_{j,v}\right). \tag{3}$$

Here, each equality constraint $\theta_{i,v} = \theta_{j,v}$ in (2) is assigned a Lagrange multiplier $\lambda_{i,j}^v \in \mathbb{R}$. The dual function is then defined as follows,

$$q(\lambda; \mathcal{S}) \stackrel{\Delta}{=} \inf_{\theta} \; \mathcal{L}(\theta, \lambda; \mathcal{S}), \tag{4}$$

where λ denotes the vector of all Lagrange multipliers.

In our case, (4) can be decomposed as a sum over the nodes in $\mathcal{N}$. Let λ_i denote the vector of Lagrange multipliers associated with a constraint involving a component of θ_i. We can rewrite q as $q(\lambda; \mathcal{S}) = \sum_{i \in \mathcal{N}} q_i(\lambda_i; \mathcal{S}_i)$, where each function q_i is

$$q_i(\lambda_i; \mathcal{S}_i) \;\stackrel{\Delta}{=}\; \inf_{\theta_i} C_i(\theta_i; \mathcal{S}_i) + \sum_{v \in \mathcal{V}_i} \left(\sum_{j \in \mathcal{M}(v), j \neq i} \mathrm{sgn}(j - i) \lambda_{i,j}^v \right) \theta_{i,v}. \tag{5}$$

The function $\mathrm{sgn}(j - i)$ returns 1 if $i < j$ and returns -1 otherwise. The function q_i depends only on information local to node i, specifically, $\mathcal{S}_i$ and the location of of the vertices of sector i. Therefore, given λ_i, each node i can solve for q_i independently. For the least squares error cost function, (5) is a quadratic minimization problem (over θ_i) and thus can be solved analytically. The full expression for q_i is given in the technical report [6].

The dual problem is

$$\underset{\lambda}{\text{maximize}} \quad q(\lambda;\mathcal{S}) = \textstyle\sum_{i\in\mathcal{N}} q_i(\lambda_i;\mathcal{S}_i). \tag{6}$$

For a square error minimization of the form (1)–(2), strong duality holds (see [16]). Therefore, the solution to (6) gives the solution to the primal problem, $\hat{\theta} = \text{argmin}_\theta \ \mathcal{L}(\theta, \hat{\lambda}; \mathcal{S})$.

We note that each Lagrange multiplier $\lambda^v_{i,j}$ appears in both λ_i and λ_j since node i and node j share the vertex v. Therefore, the objective in (6) is not separable over the nodes in $\mathcal{N}$. While the nodes can solve (5) independently for a given vector λ_i, the dual problem contains pairwise dependences between neighbors, and so the nodes must collaborate to find the optimal $\hat{\lambda}$.

4 Distributed Algorithm

We now present our distributed algorithm for generating the optimal estimate defined in the previous section. Our algorithm is based on the coordinate ascent method for nonlinear optimization [17,18]. We briefly review this method in Section 4.1. We then describe the details of our algorithm in Section 4.2 and sketch its correctness in Section 4.3. Formal proofs are given in the technical report [6].

4.1 Preliminaries - The Method of Coordinate Ascent

Consider an unconstrained optimization problem

$$\hat{x} = \underset{x\in\mathbb{R}^m}{\text{argmax}} \quad h(x_1, x_2, \ldots, x_m).$$

The method of coordinate ascent is an iterative optimization algorithm that proceeds as follows. Let $x(k) = [x_1(k) \ \ldots \ x_m(k)]$ be the vector of the values in iteration k. The algorithm begins with an initial $x(1)$. In each step k, a coordinate i is selected, and $x_i(k)$ is updated by finding its maximum while all other values of $x(k)$ are fixed. The update step is,

$$\overline{x}_i = \underset{\xi\in\mathbb{R}}{\text{argmax}} \quad h(x_1(k), \ldots x_{i-1}(k), \xi, x_{i+1}(k), \ldots, x_m(k)) \tag{7}$$

$$x(k+1) = [x_1(k) \quad \ldots \quad x_{i-1}(k) \quad \overline{x}_i \quad x_{i+1}(k) \quad \ldots \quad x_m(k)]. \tag{8}$$

We note that it is possible that the execution of an update step may not result in any change to x (i.e., $x(k+1) = x(k)$) if the selected coordinate is already optimal with respect to the rest of $x(k)$.

The convergence of the above algorithm depends on the properties of h and the order in which the coordinates are evaluated: an arbitrary update order may not converge. In this paper, we consider the *essentially cyclic policy* [19], which states that there exists a constant integer $T > 0$, such that every coordinate $i \in \{1, \ldots, m\}$ is chosen at least once between the r^{th} iteration and the $(r+T-1)^{th}$ iteration, for all r. The following theorem gives the relevant convergence result for the method of coordinate ascent with an essentially cyclic policy (see [17,19]).

Theorem 1. *Let $h(x_1, \ldots, x_m)$ be a concave function that is strictly concave in each x_i when the other variables $x_j, j \neq i$ are held constant, and let h have continuous first partial derivatives. If the coordinate update policy follows an essentially cyclic order, then the algorithm (7)–(8) converges to an optimal solution. Furthermore, if the algorithm executes a cycle of updates, where each coordinate is evaluated at least once, and no evaluation results in a change to x, then the algorithm has found an optimal solution.*

The objective in (6) is concave, has continuous first partial derivatives, and is strictly concave in each $\lambda^v_{i,j}$ when the other values of λ are fixed. So, we can solve this problem using the method of coordinate ascent. We next present a distributed algorithm, based on coordinate ascent, that solves the dual problem (6).

4.2 Distributed Optimization Algorithm

In our distributed algorithm, each node i stores its set of samples $\mathcal{S}_i$, a list of its current neighbors, and its Lagrange multipliers λ_i. Per the problem decomposition, a coordinate $\lambda^v_{i,j}$ appears in two vectors, λ_i and λ_j. Here, each $\lambda^v_{i,j}$ is a *shared variable*, and in the distributed algorithm, and nodes i and j each store a copy of $\lambda^v_{i,j}$. The goal is for nodes to converge to the optimal values for their shared variables, thus solving the dual problem (6).

To implement a distributed version of coordinate ascent for this dual problem, every pair of nodes i and j that share a coordinate $\lambda^v_{i,j}$ must collaborate to update their shared variables for $\lambda^v_{i,j}$, and the distributed algorithm must execute these pairwise updates in an order that guarantees convergence. We now show that a coordinate update can be performed with a pairwise message exchange, where each message contains two coefficients.

Distributed Coordinate Update. For an update of the shared variable $\lambda^v_{i,j}$, its new value γ depends only on the dual functions for nodes i and j,

$$\gamma = \underset{\lambda^v_{i,j}}{\operatorname{argmax}}\, q(\lambda; \mathcal{S}) = \underset{\lambda^v_{i,j}}{\operatorname{argmax}} \left(q_i(\lambda_i; \mathcal{S}_i) + q_j(\lambda_j; \mathcal{S}_j) \right).$$

The value of γ is the root of the equation

$$\frac{\partial}{\partial \lambda^v_{i,j}} (q_i + q_j) = \frac{\partial}{\partial \lambda^v_{i,j}} q_i + \frac{\partial}{\partial \lambda^v_{i,j}} q_j = 0.$$

To find this value, each node sends information about its partial derivative to the other. For the least square error cost function, this information can be encapsulated in two coefficients α^v_i and β^v_i. The values of these coefficients are given in the technical report [6]. After exchanging these coefficients, each node then independently computes $\gamma = -(\beta^v_i + \beta^v_j)/(\alpha^v_i + \alpha^v_j)$ and updates its copy of $\lambda^v_{i,j}$.

The values of the α^v_i and β^v_i are determined by the node's samples and the values of its other shared variables. These shared variables, in turn, depend on additional shared variables with other nodes. For the coordinate update step to be performed correctly, both nodes involved in the update must compute their coefficients using a consistent shared state.

Scheduler. Each node determines which coordinates should be updated by detecting which of its shared variables are not optimal with respect its current state, and it schedules pairwise exchanges on the corresponding links. Our algorithm relies on a *weakly-fair scheduler* (implemented in Section 5) as a black box to implement the pairwise message exchange on scheduled links. We give the specification for the scheduler below, followed by the details of our distributed algorithm execution.

The scheduler provides a `notify`(j) function by which a node i schedules a pairwise message exchange on a link (i, j) to update the shared variable $\lambda^v_{i,j}$. It provides functions `addNeighbor`(j) and `delNeighbor`(j) for indicating when a neighbor has been added or removed, and it provides a `getLink`() function by which a node requests a neighbor for a pairwise message exchange. The node sends messages directly to its neighbor, and the scheduler receives incoming messages. The scheduler provides the function `deliver`(j) by which a calling node delivers a message from node j. If j fails, the scheduler delivers the token FAIL.

We assume that the algorithm is notified on the addition or removal of a neighbor and invokes the appropriate function on the scheduler, `addNeighbor` or `delNeighbor`. We also assume that the algorithm executes the following infinite loop, with only non-blocking operations interleaved (as in our algorithm):

while TRUE **do**
  $j \leftarrow$ `scheduler.getLink`()
  `send`(j, msg_1)
  $msg_2 \leftarrow$ `scheduler.deliver`(j)

In this loop, a node requests the ID of neighbor, and it then exchanges messages with that neighbor. The scheduler must execute this message exchange atomically and provide some weak degree of "fairness" in executing scheduled exchanges. We now make these requirements precise.

We define a *step* of the distributed algorithm to be the delivery of a message at a single node. Without loss of generality, we assume that no two steps are executed concurrently. An *execution* of the distributed algorithm is a sequence of steps. We associate the invocation `scheduler.notify`(j) by a node i with the most recent step s that resulted in a message delivery at node i. In this case we say that the update of link (i, j) was scheduled in step s.

Definition 2 (Weakly-fair scheduler). *A scheduler is a* weakly-fair scheduler *if it guarantees the following for an algorithm that respects the above assumptions:*

1. *Every message is returned by* `scheduler.deliver` *at most once.*
2. *If* `scheduler.deliver`(j) *returns message* m *at time* t *then, if* m *is* FAIL, j *is failed at time* t. *If* m *is not* FAIL, *then* m *was previously sent by* j.
3. *If node* i *invokes* `scheduler.notify`(j), *and neither* i *nor* j *fail, then eventually* i *completes* `scheduler.deliver`(j) *and* j *completes* `scheduler.deliver`(i).

4. *Let* $\overline{s}$ *be the last step before GST. There exists a bound* B *such that, if a node* i *invokes* `scheduler.notify`(j) *at step* s*, then if* i *does not fail,* i *completes* `scheduler.deliver`(j) *by step* $\max(s, \overline{s}) + B$*, and if* j *does not fail,* j *completes* `scheduler.deliver`(i) *by step* $\max(s, \overline{s}) + B$*.*

Algorithm Execution. Each node is responsible for detecting when one of its shared variables is not optimal with respect to its sample set and other shared variables. A shared variable requires an update whenever a sample is added or deleted, a shared variable is added or removed, or any other shared variable is updated by pairwise exchange with another node. When one of these events occurs, the node schedules an update for that shared variable by invoking `scheduler.notify`.

A separate thread at each node handles the scheduled updates of shared variables, one at a time. In an infinite loop, the process requests a node ID from the scheduler by invoking `scheduler.getLink`, and it performs a pairwise message exchange with that node, e.g., node j. It sends its coefficients (as specified above) to j, and it delivers coefficients from j with `scheduler.deliver`. The node then updates its shared variable with j using the coefficients contained in the messages. If j fails before sending its coefficients, then the scheduler eventually delivers FAIL, and the node does not update the shared variable. The pseudocode for the distributed algorithm is given in the the technical report [6].

4.3 Proof Sketch of Algorithm Convergence

We now sketch the convergence proof for our distributed algorithm with a weakly-fair scheduler. The lemmas and theorem stated here are proven in the technical report [6].

We first show that, after GST, the distributed algorithm with a weakly-fair scheduler simulates the execution of the centralized coordinate ascent algorithm on the dual problem in (6). We define a mapping $\mathcal{F}$ between the state of the distributed algorithm and the state of the centralized algorithm; for a given state of the distributed algorithm at step t, $\mathcal{F}$ returns a state where the value of each coordinate $\lambda_{i,j}^v$ in λ is the value at the first node to update the shared variable for $\lambda_{i,j}^v$ in the most recent (possibly incomplete) pairwise exchange.

Lemma 1. *After GST, the algorithm simulates the centralized coordinate ascent algorithm, under the mapping* $\mathcal{F}$*.*

Next we show that any execution C of the centralized algorithm that is generated under the mapping $\mathcal{F}$ from an execution of the distributed algorithm is equivalent to an essentially cyclic execution of the centralized coordinate ascent algorithm. The key to this result is the observation that, at a step s in the distributed execution, if a shared variable is not scheduled for update at any node, then its value is optimal with respect to the values of the other shared variables at step s. This means that in the mapped step of centralized execution, the value of the corresponding value of $\lambda_{i,j}^v$ is also optimal with respect to the rest of λ. The weakly-fair scheduler guarantees that, after GST, all shared variables that are

scheduled for update in step s of the distributed algorithm will be updated by in at most B steps after s (in both the distributed and the mapped centralized executions). Therefore, we can create an equivalent, essentially cyclic centralized execution (whose cycle length depends on B) by adding empty update steps for unscheduled shared variables. This result is formally stated in the following lemma.

Lemma 2. *Let D be an execution of the distributed algorithm starting after GST, consisting of the steps $\{d_1, d_2, d_3, \ldots\}$, and let $C \stackrel{\Delta}{=} \{c_1 = \mathcal{F}(d_1), c_2 = \mathcal{F}(d_2), c_3 = \mathcal{F}(d_3), \ldots\}$ be the corresponding execution of the centralized algorithm. Then, there exists an essentially cyclic centralized execution $\overline{C}$, that is equivalent to C, i.e., $\overline{C}$ contains exactly the same non-empty updates as C, and these updates are executed in the same order.*

Lemmas 1 and 2 show that the shared variables mapped under $\mathcal{F}$ converge to an optimal solution. What remains is to show that the other shared variables converge to the same solution. This result follows directly from the definition of a weakly-fair scheduler (Definition 2). We therefore conclude that the algorithm, run with a weakly-fair scheduler, solves the dual problem.

Theorem 2. *If the scheduler is weakly-fair, then, after GST, the algorithm converges to an optimal solution of the dual problem in (6).*

Since we have convergence in the dual, each node's estimate of θ_i converges to the optimal solution of the primal problem in (1)–(2). We note that, if at a time t after GST, no shared variable is scheduled for update, then every coordinate is optimal with respect to the values of the other coordinates, and thus, the algorithm has found an optimal solution.

5 Locally Quiescent Scheduler

Our algorithm, described above, requires a weakly-fair scheduler to order communication. A naturally appealing approach for such a scheduler is to use a self-stabilizing edge-coloring [20] or other synchronizer-based methods. However, this kind of approach would require the continuous exchange of messages to maintain the synchronization pulses, and these control messages would be sent regardless of whether the algorithm needed to exchange information over the links (unless some external mechanism were used to terminate the synchronizer). Furthermore, before GST, when the network is changing, the resulting schedule might lead to deadlock, which would also require an external mechanism to break.

To complement our novel optimization algorithm described above, we propose here a weakly fair scheduler implementation that we call a *locally quiescent scheduler*. This scheduler is deadlock-free, even before GST, and it only sends messages on a link if the algorithm schedules an exchange for that link. After the algorithm stops scheduling updates, no messages are sent on any link.

The scheduler assigns a master node and slave node to each link; the node with the smaller ID is the master. Note that each node may act as a slave for

Algorithm 1. Locally quiescent scheduler at node i.

```
1  state
2      requestQueue, initially empty
3      notificationQueue, initially empty
4      msgBuffer : ℕ → msg ∪ {⊥, FAIL}
5      notified, initially ∅
6  function notify(j)
7      if min(i, j) ∉ notified then
8          notified ← notified ∪ {min(i, j)}
9          if i < j then
10             push(notificationQueue, j)
11         else
12             send(j, NOTIFY)
13 function getLink()
14     while TRUE do
15         if requestQueue not empty
           then                          (slave)
16             return pop(requestQueue)
17         else if notificationQueue not
           empty then
18             return
               pop(notificationQueue)
19 function deliver(j)
20     wait until msgBuffer(j) ≠ ⊥
21     msg ← msgBuffer(j)
22     msgBuffer(j) ← ⊥
23     return msg
24 function addNeighbor_i(j)
25     no op
26 function delNeighbor_i(j)
27     notified ← notified \ {j}
28     msgBuffer(j) ← FAIL
29 on recv_i(j, msg)
30     msgBuffer(j) ← msg
31     if j < i then    (j is the master)
32         notified ← notified \ {j}
33         push(requestQueue, j)
34 on recv_i(j, ⟨NOTIFY⟩)
35     push(notificationQueue, j)
```

some links and as a master for others. Whenever the algorithm schedules an exchange on a link, it calls the scheduler's `notify` function, which either places the link in the node's notificationQueue (if it is the master), or sends a NOTIFY message to the master of that link (if it is the slave). In this case, when the master receives the NOTIFY message, it places the link in its notificationQueue.

We now explain how the scheduler executes a pairwise message exchange for a scheduled link. Consider a link (i, j), where i is the master and j is the slave. When the master asks for a link with `getLink`, its scheduler processes the next entry in its notificationQueue and returns a neighbor j. The master proceeds by sending a message to the slave, invoking `deliver`(j), and blocking until the message from j is delivered. While the master is blocked, its scheduler queues incoming notifications and requests from masters on other links. When the slave receives the message from the master, its scheduler buffers the message in its msgBuffer and registers it in its requestQueue. Once `getLink` returns the ID of the master, the slave sends its message to the master and delivers the master's message to it. The slave's call to `deliver` returns instantly since the message from the master is already in its scheduler's buffer. Once the slave's message arrives, the `deliver` call at the master returns the message, and the pairwise exchange is complete. The scheduler is given in Algorithm 1.

The following theorem states that the scheduler is weakly-fair. The result follows from the fact that scheduled updates are handled in the order that the notifications arrive at the master and nodes only wait on their slaves. The master-slave

relationship assignments follow the total order of node IDs, prohibiting a deadlock due to cycles. A formal proof is given in the technical report [6].

Theorem 3. *The locally quiescent scheduler is a weakly-fair scheduler.*

We note that our locally quiescent scheduler only sends messages on a link in response to an invocation of `notify` for that link by the algorithm. After the algorithm stops scheduling updates, no messages are sent on any link.

Observation 1. *If a link is not scheduled by the algorithm, the scheduler sends no messages on the link. Therefore, if the algorithm ceases to schedule links, the scheduler achieves quiescence.*

6 Conclusion

We have presented a distributed algorithm for estimating a continuous phenomenon over a geographic region based on samples taken by sensors inside the region. While a straightforward solution to this problem would require expensive coordination among groups of nodes, we have shown how to decompose the problem so that the algorithm requires only pairwise communication between neighbors. We have then provided a novel, distributed implementation of coordinate ascent optimization that solves this estimation problem. Our algorithm accommodates the addition and removal of samples and the arrival and departure of nodes, and it converges to a globally optimal solution without any global coordination or synchronization. The algorithm relies on a weakly-fair scheduler to implement pairwise exchanges, and we have presented an implementation of such a scheduler. Our scheduler only sends message when the algorithm indicates that there are updates to perform, and if the algorithm finds the optimal solution, the scheduler achieves quiescence.

This work demonstrates the benefits and power of distributed *selective* learning, where agents cooperate to calculate a global optimum, while each of them learns only a part of the solution. These results call for future work, studying the possibility of relaxing the communication patterns even further and extending the algorithm to other optimization problems with different objective functions and constraints, for example, estimation of non-continuous phenomena and tracking of phenomena that change over time.

Acknowledgements. The authors thank Isaac Keslassy for his good advice.

References

1. Chong, C., Kumar, S.: Sensor networks: Evolution, opportunities, and challenges. Proceedings of the IEEE 91(8), 1247–1256 (2003)
2. Gilbert, S., Lynch, N., Mitra, S., Nolte, T.: Self-stabilizing mobile robot formations with virtual nodes. In: Kulkarni, S., Schiper, A. (eds.) SSS 2008. LNCS, vol. 5340, pp. 188–202. Springer, Heidelberg (2008)

3. Dolev, S., Tzachar, N.: Empire of colonies: Self-stabilizing and self-organizing distributed algorithm. Theoretical Computer Science 410(6-7), 514–532 (2009)
4. Fernandess, Y., Malkhi, D.: K-clustering in wireless ad hoc networks. In: Proceedings of the Second ACM International Workshop on Principles of Mobile Computing, pp. 31–37. ACM (2002)
5. Lee, D.Y., Lam, S.S.: Efficient and accurate protocols for distributed Delaunay triangulation under churn. In: IEEE International Conference on Network Protocols, pp. 124–136 (2008)
6. Eyal, I., Keidar, I., Patterson, S., Rom, R.: Global estimation with local communication. Technical Report CCIT 809, EE Pub. No. 1766, Technion, Israel Institute of Technology (May 2012)
7. Tsitsiklis, J., Bertsekas, D., Athans, M.: Distributed asynchronous deterministic and stochastic gradient optimization algorithms. IEEE Transactions on Automatic Control 31(9), 803–812 (1986)
8. Nedic, A., Ozdaglar, A.: Distributed subgradient methods for multi-agent optimization. IEEE Transactions on Automatic Control 54(1), 48–61 (2009)
9. Srivastava, K., Nedic, A.: Distributed asynchronous constrained stochastic optimization. IEEE Journal of Selected Topics in Signal Processing 5(4), 772–790 (2011)
10. Ram, S., Nedic, A., Veeravalli, V.: A new class of distributed optimization algorithms: Application to regression of distributed data. Optimization Methods and Software 27(1), 71–88 (2012)
11. Nedic, A., Ozdaglar, A., Parrilo, P.: Constrained consensus and optimization in multi-agent networks. IEEE Transactions on Automatic Control 55(4), 922–938 (2010)
12. Johansson, B., Rabi, M., Johansson, M.: A randomized incremental subgradient method for distributed optimization in networked systems. SIAM Journal on Optimization 20(3), 1157–1170 (2009)
13. Ram, S., Nedic, A., Veeravalli, V.: Asynchronous gossip algorithms for stochastic optimization. In: Proceedings of the 48th IEEE Conference on Decision and Control, pp. 3581–3586 (2009)
14. Elad, M., Matalon, B., Zibulevsky, M.: Coordinate and subspace optimization methods for linear least squares with non-quadratic regularization. Applied and Computational Harmonic Analysis 23, 346–367 (2007)
15. Bradley, J.K., Kyrola, A., Bickson, D., Guestrin, C.: Parallel coordinate descent for l1-regularized loss minimization. In: Proceedings of the 28th International Conference on Machine Learning, pp. 321–328 (2011)
16. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
17. Luenberger, D.G.: Linear and Nonlinear Programming, 2nd edn. Addison-Wesley Publishing Company, Inc. (1984)
18. Bertsekas, D.P.: Nonlinear Programming, 2nd edn. Athena Scientific (1999)
19. Tseng, P.: Convergence of a block coordinate descent method for nondifferentiable minimization. Journal of Optimization Theory and Applications 109(3), 475–494 (2001)
20. Tzeng, C., Jiang, J., Huang, S.: A self-stabilizing ($\delta+ 4$)-edge-coloring algorithm for planar graphs in anonymous uniform systems. Information Processing Letters 101(4), 168–173 (2007)