The Role of Network Topology for Distributed Machine Learning

Giovanni Neglia, Gianmarco Calbi
Université Côte d'Azur, Inria, France
{giovanni.neglia,gianmarco.calbi}@inria.fr

Don Towsley, Gayane Vardoyan
University of Massachusetts - Amherst, MA, USA
{towsley,gvardoyan}@cs.umass.edu

***Abstract*—Many learning problems are formulated as minimization of some loss function on a training set of examples. Distributed gradient methods on a cluster are often used for this purpose. In this paper, we study how the variability of task execution times at cluster nodes affects the system throughput. In particular, a simple but accurate model allows us to quantify how the time to solve the minimization problem depends on the network of information exchanges among the nodes. Interestingly, we show that, even when communication overhead may be neglected, the clique is not necessarily the most effective topology, as commonly assumed in previous works.**

I. Introduction

A basic step in large scale machine learning is to compute parametric models minimizing a loss function. To take advantage of parallelism or in-memory computation, the dataset may be split among different executors, each using its data subset to perform the same task: computing a noisy estimate of the gradient of the loss function. The gradient estimates are then averaged during a synchronization phase. This improved estimate is used to update model parameters. The synchronization phase can be time consuming due to the need to wait for the slowest tasks (called *stragglers*). Asynchronous solutions have been proposed, where nodes may work on stale data and read-write race conditions may arise [1]. Empirically, on single-node systems, these asynchronous algorithms have yielded order-of-magnitude increases in system throughput (number of tasks executed per time unit) [1].

Unfortunately, asynchronicity may impede convergence to the optimal model, and a higher throughput does not necessarily guarantee faster convergence [2]. Large-scale data-parallel computation frameworks, like Spark, usually rely on the bulk synchronous parallel model with high synchronization overhead. [3] shows how asynchronous primitives can be introduced in Spark in order to implement stochastic gradient or alternating direction method of multipliers, but convergence is not guaranteed. [2] proposes ASAP, a computational model introducing a form of fine-grained synchronization, where each executor only requires state updates from a few other executors to proceed in the computation. Executors are then nodes of a *dependency graph*, where an edge (i, j) indicates that executor j needs to wait for executor i. A sparse graph reduces communication overhead and may mitigate the effect of stragglers, because only nodes waiting for a straggler are slowed down. This increases the overall completion rate. At the same time the "quality" of the updates can suffer, because each node receives information only from a limited neighbourhood. One might then expect that more iterations will be required for convergence. This raises a number of questions: can a sparse dependency graph significantly increase throughput (the rate at which iterations complete) or will stragglers slow down the whole network, anyhow? and which effect prevails? does higher throughput compensate for the need for more iterations? To the best of our knowledge, apart from experimental results in [2], our paper is the first attempt to answer these questions and to quantify this interesting tradeoff.

ASAP computational model fits the general framework of optimization problems defined over networks. Such network-structured optimization problems arise in a variety of application domains within the information sciences and engineering, like multi-agent coordination, distributed tracking and localization, estimation problems in sensor networks and packet routing. In some of these applications, network topology is imposed by specific communication constraints (e.g. limited sensor transmission range), while in our case we have flexibility to define the topology of the dependency graph. In the field of consensus-based (also called gossip-based) distributed optimization over networks there are some results [4], [5], [6], [7] that relate convergence rate to level of network connectivity, but they consider convergence rate in terms of *number of iterations* to achieve a given precision. Our work builds on these results to characterize the *time* required to achieve this precision. Analyses in previous studies reach the (expected) conclusion that a more connected network can only improve the convergence rate (in terms of number of iterations) unless communication delays increase with the level of connectivity [6]. In this paper, we conclude that, even ignoring communication delays, a sparser network can sometimes lead to smaller convergence time.

The paper is organized as follows. Sect. II provides required background. The analysis of the system throughput is developed in Sect. III, and presents both exact results and an approximate model. In Sect. IV we exploit our model to determine the best dependency graph—i.e. the graph leading to the fastest convergence time—as a function of the number of nodes and the distribution of the computation times. The numerical results in Sect. V confirm our theoretical findings for specific machine learning problems. Finally, Sect. VI discusses related work.

II. Notation and Background

Supervised learning aims to learn a function that maps an input to an output using m examples from a training dataset $\mathcal{S} = \{(\chi_l, y_l), l = 1, \dots m\}$. Each example (χ_l, y_l) is a pair consisting of an input object χ_l and a desired output value y_l. In order to find the best statistical model, machine learning techniques often find the best set of parameters $\mathbf{x} \in \mathbb{R}^p$ by solving the following optimization problem:

$$\begin{aligned} &\underset{\mathbf{x}}{\text{minimize}} && \sum_{l=1}^{m} f(\mathbf{x}, \chi_l, y_l) + R(\mathbf{x}) \\ &\text{subject to} && \mathbf{x} \in X \subset \mathbb{R}^p \end{aligned} \tag{1}$$

where the function $f(\mathbf{x}, \chi_l, y_l) :\to \mathbb{R}$ represents the error the model commits on the l-th element of the dataset $\mathcal{S}$ when vector of parameters $\mathbf{x}$ is used, while $R(\mathbf{x})$ is a regularization term that enforces some "simplicity" (e.g. sparseness) of $\mathbf{x}$. Both functions f_l and R are often convex. For example, when χ_l is a vector in $\mathbb{R}^p$ and $y_l \in \mathbb{R}$, the standard linear regression method determines the optimal parameter vector considering $f(\mathbf{x}, \chi_l, y_l) = (\mathbf{x}^\intercal \chi_l - y_l)^2$ and $R(\mathbf{x}) = 0$. When $R(\mathbf{x})$ is the squared Euclidean norm or the 1-norm, one obtains respectively ridge and lasso regression ([8] provides a simple introduction to these methods and those we mention below). Support Vector Machine methods for classification consider $y_l \in \{-1, 1\}$, $f(\mathbf{x}, \chi_l, y_l) = \max(0, 1 - y_l \mathbf{x}^\intercal \chi_l)$ (the hinge loss function) and again the Euclidean norm for regularization. Logistic regression (another classification technique) considers $f(\mathbf{x}, \chi_l, y_l) = \log(1 + \exp(-y_l \mathbf{x}^\intercal \chi_l))$.

Different iterative techniques have been developed to solve problem (1) (see [9] for a nice introduction). Due to increases in available data and complexity of statistical models, distributed solutions are often required to determine the parameter vector in a reasonable time. The dataset in this case is divided among n computing nodes ($\mathcal{S} = \dot\bigcup_{i=1}^{n} \mathcal{S}_i$), and problem (1) can be restated as minimization of the sum of functions local to each node:

$$\underset{\mathbf{x}}{\text{minimize}} \quad F(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} F_i(\mathbf{x}), \quad \text{subject to} \quad \mathbf{x} \in X \tag{2}$$

where $F_i(\mathbf{x}) = n \sum_{(\chi_l, y_l) \in \mathcal{S}_i} f(\mathbf{x}, \chi_l, y_l) + R(\mathbf{x})$. An immediate distributed implementation using subgradients and the popular *parameter server* framework [10] is the following: at iteration k each node i computes a subgradient[1] $\mathbf{G}_i(\mathbf{x}(k))$ of its local function and sends it to a server. The server then updates the current parameter vector (the model) as follows:

$$\mathbf{x}(k+1) = \Pi_X\left(\mathbf{x}(k) - \alpha(k) \frac{1}{n} \sum_{i=1}^{n} \mathbf{G}_i(\mathbf{x}(k)) \right), \tag{3}$$

and sends the new updated model to all nodes, which then start the $(k+1)$-th computation. The parameter $\alpha(k) > 0$ is the (potentially time-varying) learning rate, and Π_X denotes the projection operator on the feasible set X. Note that the parameter server needs to wait for all executors to compute their subgradients. This requirement makes the system particularly sensitive to stragglers, which can significantly reduce computation speed of a distributed system [11], [12], [13].

A different computational model is the *bulk synchronous parallel* (BSP) one, which is shared by many general-purpose computation frameworks like Pregel, Giraph or Spark, and relies on synchronization barriers to guarantee consistency and fault-tolerance. Under BSP, the executors do not communicate with a parameter server, but each executor broadcasts its local subgradient and aggregates the subgradients of all the nodes to compute the new model. Despite the different communication patterns, the parameter vector evolves according to (3) and each update still requires that all nodes finish computing their subgradient.

In order to mitigate the effect of stragglers, ASAP [2] proposes a form of fine-grained synchronization, where each executor only needs to wait for the state updates of a few other executors to proceed the computation. Experimental results on an infiniBand implementation stack show up to 2-10X speedups in convergence time. The model in this paper helps to understand under what conditions (e.g. distribution of the computation times, characteristics of the executors) similar improvements can be achieved and moreover how to estimate them without the need to carry out expensive experiments.

We now formally describe a general Distributed Subgradient Method (DSM) to solve problem (1). The distributed system can be represented by a directed *dependency graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, 2, \dots n\}$ is the set of executors and an edge $(i, j) \in \mathcal{E}$ indicates that at each iteration node j needs to wait for the previous iteration from node i. We assume that the graph is strongly connected. Let $N_i = \{j | (j, i) \in \mathcal{E}\}$ denote the in-neighbourhood of node i, i.e. the set of predecessors of node i in $\mathcal{G}$. Each node i maintains a local estimate of the parameter vector $\mathbf{x}_i(k)$ and broadcasts it to its successors. Then the local estimate is updated as follows:

$$\mathbf{x}_i(k+1) = \Pi_X\left(\sum_{j \in N_i \cup \{i\}} p_{i,j} \mathbf{x}_j(k) - \alpha(k) \mathbf{G}_i(\mathbf{x}_i(k)) \right), \tag{4}$$

i.e., apart from the projection, the node computes a weighted average (the consensus/gossip component) of the estimates of its neighbours and itself, and then corrects it taking into account the subgradient of the local function. $P_n = (p_{i,j})$ is an $n \times n$ matrix of non-negative weights. We call it the *consensus matrix*. It is clear that (4) closely describes ASAP computational model, but it also encompasses the BSP model or the parameter-server framework. In fact, if *i)* the graph is an undirected clique, *ii)* the consensus weights are all equal ($p_{i,j} = 1/n$) and *iii)* nodes start with an identical parameter vector ($\mathbf{x}_i(0) = \mathbf{x}_j(0)$ for each $i, j \in \mathcal{V}$), then the parameter vectors remain equal at any iteration and (4) reduces to (3).

References [14], [5] study the convergence of DSM. We state some of their results under the following assumptions:

[1]Given a function $f(\mathbf{x})$, a subgradient of $f()$ in $\mathbf{x}$ is a vector $\mathbf{g}$, such that $f(\mathbf{y}) - f(\mathbf{x}) \geq \mathbf{g}^\intercal(\mathbf{y} - \mathbf{x})$. In general a function can have many subgradients in a point $\mathbf{x}$. When the function f is differentiable in $\mathbf{x}$, the only subgradient is the gradient. With some abuse of notation, we indicate a subgradient in $\mathbf{x}$ as $\mathbf{g}(\mathbf{x})$, even if $\mathbf{g}$ is not a function.

A1) all functions F_i are convex and their subgradients in set X are bounded by a constant L, *A2)* there is a (global) minimizer $\mathbf{x}^* \in X$, *A3)* the matrix P is doubly stochastic, *A4)* the graph $\mathcal{G}$ is strongly connected. In particular, [5, Thm 8] states that if the total number of iterations is K and $\alpha(k) = 1/\sqrt{K}$

$$F\left(\frac{\sum_{k=0}^{K-1} y(k)}{K}\right) - F(x^*) \qquad (5)$$
$$\leq \frac{(y(0)-x^*)^2 + L^2}{2\sqrt{K}} + \frac{2L^2}{\sqrt{K}\gamma(P_n)},$$

where $y(k) = 1/n \sum_{i=1}^{n} x_i(k)$, $\gamma(P_n) = 1 - \sigma_2(P_n)$ is the spectral gap of the matrix P_n, and $\sigma_2(P_n)$ is its second largest singular value. The first term on the right hand side depends on the initial distance from the minimizer and shows the classic $1/\sqrt{K}$ convergence speed of centralized subgradient methods. The second term[2] instead is an additional cost due to the distributed nature of the computation. In particular, $1/\gamma(P_n)$ is minimal for a clique network ($\gamma(P_n) = 1$), while it can be arbitrarily large for a generic network. In fact, $\gamma(P_n)$ converges to 0 as the number of nodes n increases for many families of graphs with bounded degree like grids, rotationally invariant graphs, random geometric graphs (see also Sect. IV).

The bound (5) is proven in [5] *i)* when all nodes start with the same parameter vector, i.e. $x_i(0) = x_j(0)$ for each i and j, and *ii)* for the one-dimensional case, i.e. $F_i : \mathbb{R} \to \mathbb{R}$. The authors claim that the result can be generalized to the case when $F_i : \mathbb{R}^p \to \mathbb{R}$ by using techniques developed in previous papers like [14]. If one selects $\alpha(k) = \sqrt{\gamma(P_n)/K}$, it is possible to obtain

$$F\left(\frac{\sum_{k=0}^{K-1} y(k)}{K}\right) - F(x^*) \leq \frac{\left((y(0)-x^*)^2 + 3L^2\right)}{2\sqrt{K}\sqrt{\gamma(P_n)}}. \quad (6)$$

To the best of our knowledge, this is the tightest bound available in terms of the spectral gap of P. It was first observed for the Dual Averaging Distributed method (DADM) proposed in [4] under assumptions similar to A1-A4 above.

Motivated by error bounds such as (6), the existing literature has concluded that a more connected network topology leads to faster convergence with the only downside being the larger amount of messages to be exchanged among the nodes (roughly proportional to the number of edges). We will show in Sect. IV that the conclusion can radically change when one takes into account that different executors may require different times to carry out the basic iteration in (4), and one may want to deploy less-connected networks!

III. Convergence speed: homogeneous executors

In this section we focus the convergence rate over *time* for the case that nodes are homogeneous, i.e. there is no systematic difference in the time each of them needs to carry out one iteration. Still, task execution times can exhibit variability across nodes and across time, e.g. due to other processes running in parallel, as may happen in a shared cluster or a cloud environment. This variability can be significant: [11] reports that in the production clusters at Facebook and Microsoft Bing straggling tasks can be eight times slower than the median task.

Let $\tau_i(k)$ be the time required for node i to compute (4) during the k-th iteration. We assume that $\{\tau_i(k), i = 1, \dots n, k = 0, 1, \dots\}$ is a set of i.i.d. random variables (rvs) with finite expectation. It follows from (6) that the number of iterations needed to achieve a given accuracy ϵ is

$$K_\epsilon \in \mathcal{O}\left(\left(\frac{1}{\epsilon^2 \gamma(P_n)}\right)\right). \quad (7)$$

In what follows, we characterize the *time* to reach a given accuracy ϵ. We focus first on the case of a clique.

A. Clique

When graph $\mathcal{G}$ is a clique, all nodes start computing at the same time. Let t_k denote the time they start iteration k. Nodes terminate their calculations at different times $\{t_k + \tau_i(k)\}$, but all need to wait for the last one to finish before being able to move to the calculations of the $(k+1)$-th iteration. The expected duration of one iteration is then

$$\theta_{\text{clique}} \triangleq \mathbb{E}[\max\{\tau_1, \tau_2, \dots \tau_n\}] = \mathbb{E}\left[\tau_{(n)}\right], \quad (8)$$

where we have used the standard notation $\tau_{(l)}$ to indicate the l-th smallest order statistic of the variables $\{\tau_1, \tau_2, \dots \tau_n\}$.

Once we have computed θ, the time to achieve a given accuracy ϵ is

$$T_\epsilon = K_\epsilon \theta. \quad (9)$$

Across the examples in this paper, we will consider $\tau_i = (1-v) + v\phi_i$, where $v \in [0, 1]$ is a constant and ϕ_i follows one of the following three distributions: *i)* uniform over the interval $[0, 2]$, *ii)* exponential with parameter one, and *iii)* Pareto type II with range $[0, \infty)$, shape parameter β and scale parameter $\zeta = \beta - 1$ (in the numerical examples we consider $\beta = 3$ so that both the first and second moments of τ_i are finite). Note that in all three cases $\mathbb{E}[\tau_i] = 1$. Constant v represents the amount of random variability of the computation time. We refer to these three cases simply as the uniform, exponential, and Pareto cases, independent of the value of v. These examples span three different situations: in particular for $v = 1$ the hazard function is respectively increasing, constant and decreasing. Then, as t increases, the expected residual computation time given that $\tau_i > t$: *i)* decreases in the uniform case, *ii)* does not change in the exponential one, and *iii)* increases in the Pareto one. Computing (8) reduces to calculating the expected value of the maximum of independent rvs. After some standard calculations, we obtain:

$$\theta_{\text{clique}}(n) = (1-v) + v \times g_{\text{clique}}(n), \text{ with} \quad (10)$$
$$g_{\text{clique}}(n) = \begin{cases} 2\frac{n}{n+1}, & \text{uniform,} \\ H(n), & \text{exponential,} \\ (\beta-1)\left(\frac{1}{\binom{n-1/\beta}{n}} - 1\right) & \text{Pareto,} \end{cases}$$

[2]We have added a factor 2 missing in [5].

where $H(n) = \sum_{i=1}^{n} 1/i$ is the harmonic function and $\binom{\beta}{n}$ is the generalized binomial coefficient defined for any non negative integer n and any real value β as $\binom{\beta}{n} = \frac{\beta(\beta-1)\dots(\beta-n+1)}{n!}$. Combining (10), (9), and (7) allows us to evaluate the effect of the variability in the computation time. We observe that $\theta_{\text{clique}}(n)$ increases in v and in n (for $v > 0$). Even a small variability of the computation time like $v = 0.1$ increases the iteration time θ_{clique} by about 10% in the uniform case, by 23% in the exponential case, and by 100% in the Pareto case for a system with $n = 100$ nodes. The larger the number of nodes, the larger the increase of θ_{clique}. We will observe that convergence time is in practice more sensitive to variability in the computation time, than to network connectivity. It may then be worthwhile to sacrifice connectivity if this allows one to mitigate the effect of variability of computation times on convergence rate.

B. Regular graphs: exact analysis

In what follows we consider dependency graphs where each node has in-degree d, although many of the results hold even when nodes have different in-degrees. When the graph is not a clique, different nodes can be executing different iterations at a given time t. Moreover, during one iteration, say k, node i may be computing or waiting for inputs from predecessors currently processing at earlier iterations. We denote the state of node i at time t as $s_i(t) \in \{1, 2, \dots\}$. $s_i(t)$ identifies whether node i is computing or waiting. In particular if $s_i(t) = 2k-1$, then node i is currently performing its k-th computation. If $s_i(t) = 2k$, then node i has completed the k-th computation, and is currently waiting for (at least) one of its predecessors to provide the input needed to start the $(k+1)$-th computation. In both cases, the number of iterations completed by node i by time t is $\lfloor s_i(t)/2 \rfloor$. The difference between the states of any two nodes can be bounded in terms of the graph diameter. A detailed analysis (whose proof we omit due to space constraints) shows that:

Proposition III.1. *At time t the states of any two nodes i and j may differ by at most $2D-1$, where D is the diameter of the graph $\mathcal{G}$, i.e.*

$$|s_i(t) - s_j(t)| \leq 2D - 1, \forall t, i, j.$$

We define the *average duration of an iteration* in the case of a general dependency graph $\mathcal{G}$ as

$$\theta = \lim_{t\to\infty} \frac{t}{\lfloor s_i(t)/2 \rfloor} \tag{11}$$

that does not depend on the specific node i considered because of Proposition III.1. We will show that the limit in (11) is well defined (it exists almost surely).

Our computation framework can be described as a Fork, Join, Queueing Network with Blocking (FJQN/B) [15]. A FJQN-B is a queueing network that can be represented by a directed graph where nodes are servers and buffers are associated with edges. In general, a server has multiple input buffers on the incoming edges and multiple outgoing edges. A server is allowed to start service whenever there is at least one job in each of its input buffers (the server is not starved) and space for at least one job in each of its output buffers (the server is not blocked). Upon completion of service at a node, one job is removed from each of its input buffers and one job is added to each of its outgoing buffers. A complete description of a FJQN-B requires specifying, besides the buffer sizes and service time distributions, an initial "marking," i.e. the number of jobs in each buffer at time $t = 0$. It is easy to check that our system can be represented as a FJQN/B where a job in the buffer at link (i, j) maps to an estimate $\mathbf{x}_i$ computed by node i, but not processed yet by node j. In particular, the graph associated to the FJQN/B is $\mathcal{G}$ itself, the initial marking has one job at each buffer, and buffer sizes can be set equal to $B = D + 1$.[3]

Let $T_i(k) = \inf\{t | s_i(t) \geq 2k\}$ be the time instant by which node i terminates iteration k. We can prove:

Proposition III.2. *Assume that computation times $\{\tau_i(k)\}$ form jointly a stationary and ergodic sequence of integrable rvs. Then there exists a constant θ such that, almost surely,*

$$\theta = \lim_{t\to\infty} \frac{t}{\lfloor s_i(t)/2 \rfloor} = \lim_{k\to\infty} \frac{T_i(k)}{k}, \ \forall i.$$

Proof. We start proving that the FJQN/B of our system is necessarily deadlock-free [15, Def. 4.2], that means, roughly speaking, that it is not possible that all nodes are blocked or starved at a given time. We define a chain of length l to be a sequence of undirected contiguous edges, i.e. (i_1, i_2), (i_2, i_3), $\dots (i_l, i_{l+1})$ with $(i_h, i_{h+1}) \in E$ or $(i_{h+1}, i_h) \in E$ for each $h = 1, 2, \dots l$. A cycle is a chain with $i_{h+1} = i_1$. Consider the system at time 0. Given a cycle $\mathcal{C}$ of length l, define an arbitrary orientation of it. Let $I_+(\mathcal{C})$ denote the total number of jobs in all the buffers on edges that are oriented according to the reference orientation plus the total number of holes (places available) in all the buffers on edges that are oriented according to the reverse direction. It is possible to show that $I_+(\mathcal{C})$ does not change over time. Given that the initial marking of our FJQN/B has a single job in any buffer, it follows

$$l \leq I_+(\mathcal{C}) \leq l(B-1),$$

where $B(= D+1)$ is the size of each buffer. The lower (resp. upper) bound corresponds to the case where all the edges in the cycle are oriented according to the reference (resp. reverse) direction. Being that $0 < I_+(\mathcal{C}) < lB$ for each cycle, it follows from [15, Theorem 4.1] that the FJQN/B is deadlock-free.

We now observe that because of Prop. III.1

$$\lim_{t\to\infty} \frac{t}{\lfloor s_i(t)/2 \rfloor} = \lim_{t\to\infty} \frac{t}{\lfloor \min_i s_i(t)/2 \rfloor},$$

if any of them exists. Moreover, for any deadlock-free FJQN/B it can be checked that

$$\frac{\max_i T_i(k)}{k} \leq \frac{t}{\lfloor \min_i s_i(t)/2 \rfloor} < \frac{k+1}{k} \frac{\max_i T_i(k+1)}{k+1}.$$

[3]This value guarantees that there is no buffer blocking in the system but only synchronization blocking.

Then, if $\lim_{k\to\infty} \frac{\max_i T_i(k)}{k}$ exists, θ exists and is equal to such limit. The result then follows from [15, Thm. 5.2].

□

We can also prove the following bounds for θ (for simplicity we state them only for i.i.d. computation times):

Proposition III.3. *If computation times are bounded from above by $\tau_{\max}$, the following holds:*

$$1 \le \theta \le 4\tau_{\max}. \tag{12}$$

If they are unbounded but have finite moment generating function $M_\tau(t)$, the following holds:

$$\max\left(1, \frac{\mathbb{E}\left[\tau_{(d)}\right]}{D}\right) \le \theta \le \frac{\frac{2\ln(4dM_\tau(t))}{D} + \ln(M_\tau(t)) + \ln 2}{t/2}, \tag{13}$$

for any $t > 0$ such that $M_\tau(t) < \infty$.

Proof. We rely on [16, Thms 3, 4]. The bounds in Prop. III.3 follow almost immediately once we prove that both the minimum level of our FJQN/B and the deterministic cycle time equal one. We refer the reader to [16] for the formal definitions of these two quantities and related ones.

In order to compute the level, one needs to first consider the circuit-free variant of the FJQN/B [16, Sect. 3.1]. In our case, because the initial marking has one job at each edge and the size of the buffers is $D+1 > 1$, the circuit-free variant is a new FJQN/B with graph $(\mathcal{V}\cup\mathcal{V}', \mathcal{E}')$, where $\mathcal{V}'$ is a new set of $|\mathcal{E}|$ nodes, one for each edge in the original graph. We indicate the node in $\mathcal{V}'$ corresponding to edge $(i,j) \in \mathcal{E}$ as $u'(i,j)$. The set of edges $\mathcal{E}'$ is defined as follows: $\mathcal{E}' = \{(i, u'(i,j)) \;\; \forall (i,j) \in \mathcal{E}\} \cup \{(j, u'(i,j)) \;\; \forall (i,j)\in\mathcal{E}\}$. Buffer sizes are equal to $b^- = 1$ for the first set of edges and to $b^+ = D$ for the second group. We observe that the graph $(\mathcal{V}\cup\mathcal{V}', \mathcal{E}')$ is bipartite, with links only from the original nodes in $\mathcal{V}$ to nodes in $\mathcal{V}'$. It follows that the function $l : \mathcal{V}\cup\mathcal{V}' \to \mathbb{N}$, such that $l(v) = 1$ if $v \in \mathcal{V}$ and $l(v) = 2$ if $v \in \mathcal{V}'$ is trivially the optimal topological labelling [16, Sect. 4.1] of the graph with maximum label gap over the edges equal to 1. Then the minimum level associated to our FJQN/B is indeed 1.

For the deterministic cycle time, we need to consider the deterministic non-blocking variant of our FJQN/B [16, Sect. 3.2]. The corresponding graph has the same set of nodes $\mathcal{V}$. The set of edges is increased by adding for each edge in $\mathcal{E}$ a reversed edge with initial marking equal to the number of holes in the original edge, and a self-loop edge for each node with initial marking equal to one job. Because in our case $\mathbb{E}[\tau] = 1$, the deterministic cycle time can then be computed as $\max_{\xi\in\Xi} \frac{S(\xi)}{J(\xi)}$, where Ξ is the set of all the circuits[4] in the deterministic non-blocking variant, $S(\xi)$ and $J(\xi)$ denote respectively the number of nodes and the total number of jobs in the circuit ξ. In our case every edge has at least one job, so that the cycle time is at most one, but this bound is achieved on any circuit made by the original edges. We conclude then that the deterministic cycle time for our FJQN/B is indeed 1.

□

[4]A path is a sequence of directed contiguous edge, i.e. (i_1, i_2), (i_2, i_3), $\ldots (i_l, i_{l+1})$ with $(i_h, i_{h+1}) \in \mathcal{E}$ for each $h = 1, 2, \ldots l$. A circuit is a path with $i_{h+1} = i_1$.

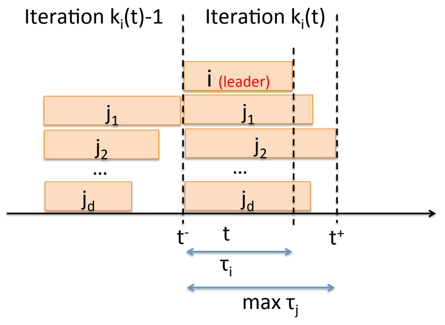


Fig. 1. Reference behaviour for a leading node (i) and its predecessors as considered to approximate θ.

Unfortunately, the above bounds are not useful for our goals. For example (12), for the case of uniform rvs and $v = 1$, leads to $1 \le \theta \le 8$ independent of topology. The bound (13) for the case of exponential rvs exhibits a dependence on the topology, but is very loose: for the clique, it predicts ($t = 1/2$) $H(n-1) \le \theta \le 8\ln(n-1) + 32\ln(2)$.

These considerations justify our approximate approach in the following section.

C. Regular graphs: approximate analysis

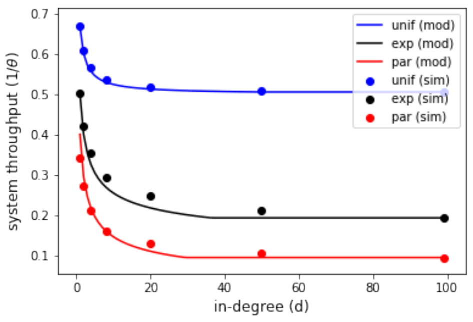
Proposition III.2 guarantees that the iteration duration θ is well defined and in particular does not depend on the specific node considered. We focus on a node for which it is easy to compute an approximate value of θ. In particular, it is convenient to consider a node i that at time t has the largest value of $s_i(t)$, i.e. $i \in \arg\max_{j\in\mathcal{V}}\{s_j(t)\}$. We say that i is a *leading* node (at time t). Let $t_i^- = \min_\xi\{s_i(\xi) \ge 2k_i(t) - 1\}$ be the time at which i started the $k_i(t)$-th computation and $t_i^+ = \min_\xi\{s_i(\xi) \ge 2k_i(t)+1\}$ the time at which it starts computation $k_i(t)+1$. We observe that $t_i^- \le t < t_i^+$. The expected value of $t_i^+ - t_i^-$ can be used to estimate θ. At time t_i^- node i has received the last result of iteration $k_i(t) - 1$ from its predecessors and starts computing iteration $k_i(t)$. Because i is a leading node, we assume that i terminates the computation before any of its predecessors have sent its new input (the results of iteration $k_i(t)$) and then needs to wait for them. Moreover, we assume that the predecessors started computing iteration $k_i(t)$ at time t_i^-. Figure 1 illustrates this hypothetical behaviour. Under the above assumptions, we can approximate

$$\theta \approx \mathbb{E}\left[t_i^+ - t_i^-\right] \approx \mathbb{E}_{\tau_i}\left[\mathbb{E}_{\{\tau_j, j\in N_i\}}\left[\max_{j\in N_i}\tau_j | \tau_j > \tau_i\right]\right],$$

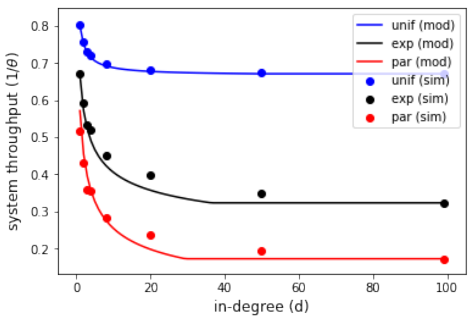
or in a more compact form:

$$\theta \approx \mathbb{E}_{\tau_0}\left[\mathbb{E}_{\{\tau_i, i=1,\ldots d\}}\left[\tau_{(d)} | \tau_{(1)} \ge \tau_0\right]\right] \tag{14}$$

where $\tau_0, \tau_1, \ldots \tau_d$ are i.i.d. rvs and $\tau_{(d)}$ and $\tau_{(1)}$ are respectively their maximum and minimum. The above reasoning is not exact for several reasons: at time t_i^- some predecessors may not start computing $k_i(t)$ because they are blocked



(a) v = 1



(b) v = 0.5

Fig. 2. Model (17) vs simulations for $n = 100$ nodes.

waiting for inputs from their predecessors or, on the contrary they may have already started computing $k_i(t)$ and may end up finishing earlier than node i (that would then no longer be a leading node at time t_i^+). We tried a number of different approximations to account for some of these effects, but found that (14) provided the best results. It is possible to refine (14), by considering that for a generic graph $\theta \leq \theta_{\text{clique}}$—as it may be rigorously proven through a path coupling argument—leading to

$$\theta \approx \min\left\{\mathbb{E}_{\tau_0}\left[\mathbb{E}_{\{\tau_i\}}\left[\tau_{(d)}|\tau_{(1)} \geq \tau_0\right]\right], \mathbb{E}\left[\tau_{(n)}\right]\right\} \tag{15}$$

Calculations similar to those used to derive (10) lead to:

$$\theta \approx (1-v) + vg(d), \text{ with} \tag{16}$$

$$g(d) = \begin{cases} \frac{2d+1}{d+1}, & \text{uniform,} \\ H(d) + 1, & \text{exponential,} \\ \beta\left(\frac{1}{\binom{d-1/\beta}{d}} - 1\right) + 1 & \text{Pareto.} \end{cases}$$

The refinement (15) leads to

$$\theta \approx (1-v) + v\tilde{g}(d), \tag{17}$$

where $\tilde{g}(d) = \min(g(d), g_{\text{clique}}(n))$.

a) Validation: In order to check the quality of our approximation, we have simulated the system for $t_{\text{sim}} = 10^4$ time units[5] and compared the predictions of (17) with the empirical estimation $\frac{t_{\text{sim}}}{\lfloor s_i(t_{\text{sim}})/2 \rfloor}$. Figure 2 shows the results for the three different distributions when the dependency

[5]Remember that the reference is a task expected execution time ($\mathbb{E}[\tau_i] = 1$).

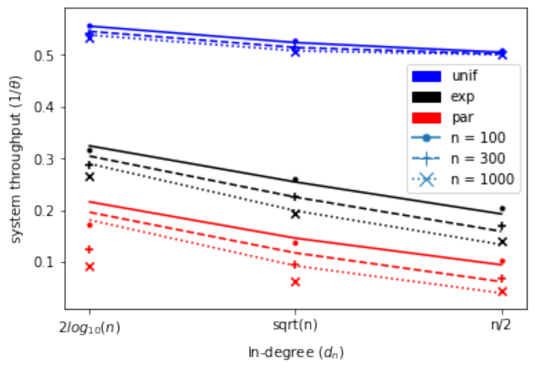


Fig. 3. Model (17) vs simulations for $n = 100, 300, 1000$ nodes. The degree is set in three different ways: 1) $d_n = 2\log_{10} n$, 2) $d_n = \sqrt{n}$, 3) $d_n = 0.5n$.

graph $\mathcal{G}$ is a *directed regular ring lattice*, i.e. each node i is connected to nodes $i+1, i+2, \ldots, i+d$, where the sums are modulo n. The figure shows throughput (the number of iterations per time unit, equal to $1/\theta$) versus the degree d of the ring. The approximation is quite accurate in all cases. We can now quantitatively answer the first question we posed in the introduction: a sparse dependency graph indeed significantly increases system throughput. The heavier the tail of the computing time distribution, the larger the throughput speedup in comparison to the clique: in particular, the figure shows that the throughput of the cycle is 20% larger than the clique for the uniform case and $v = 0.5$ and up to almost 400% larger for the Pareto case and $v = 1$. Figure 3 shows that the approximation is reasonably good for different network sizes n with the error being larger for Pareto and for smaller degree values d.

IV. The Best Dependency Graph

By combining (7), (9) and (16)[6] and considering the bound in (7) to be strict we obtain:

$$T_\epsilon \sim \frac{1 - v + vg(d)}{\epsilon^2\gamma(P_n)}. \tag{18}$$

This equation summarizes the tradeoff between achieving a higher throughput with less connected topologies (the numerator increases in d) and improving the quality of parameter-vector updates with more connected ones (the denominator in general increases in d). Which effect prevails? In order to answer, we need to specify not only the topology, but also the consensus weights. In what follows, we assume that each node gives equal weight $(1/(d+1))$ to the $d+1$ parameter vectors it averages according to (4). Then it holds:

$$P_n = \frac{1}{d+1}\left(I_n + A_n\right), \tag{19}$$

where A_n is the adjacency matrix of the graph $\mathcal{G}$ and I_n is the $n \times n$ identity matrix.

[6]For θ we are considering Eq. (16) and not the more accurate (17), because the max appearing in (16) makes the optimal degree change abruptly with n in an artificial way.

We examine some topologies for which we can get closed form expressions (or good estimates) for the spectral gap $\gamma(P_n)$ and then use (18) to determine the degree d leading to the smallest convergence time.

A. *Rings*

We consider the special case of directed regular ring lattices (introduced in Sect. III-C), which we refer to simply as *rings*. For rings, geometric calculations yield the following expression for the spectral gap: $\gamma(P_{n,r}) = 1 - \frac{1}{d+1}\frac{\sin\left(\frac{\pi(d+1)}{n}\right)}{\sin\left(\frac{\pi}{n}\right)}$. We observe that the spectral gap increases in d. For a given n, we can determine the degree d that minimizes (18). Numerical studies produce a result that is invariant across the three distributions and for different values of v: the clique ($d = n-1$) yields the minimum convergence time. Hence, convergence time for rings is decreased more effectively through improving the quality of the updates, rather than increasing the throughput of the system.

An asymptotic analysis can provide more insight into this finding. We allow the degree to be a function of n with $d_n \in o(n)$. For large n, the Taylor series of $\gamma(P_{n,r})$ leads to

$$\gamma(P_{n,r}) \sim \frac{\pi^2}{6n^2} d_n^2.$$

For $v > 0$, the asymptotic scaling of $1 - v + vg(d_n)$ is determined by $g(\cdot)$. We can then simply consider $v = 1$. For large d_n, $g(d_n)$ is in $\Theta(1)$, $\Theta(\ln d_n)$, and $\Theta(d_n^{1/\beta})$ respectively for the uniform, the exponential and the Pareto case. We obtain:

$$T_\epsilon \sim \begin{cases} n^2/d_n^2, & \text{uniform,} \\ n^2(\ln d_n)/d_n^2, & \text{exponential,} \\ n^2/d_n^{2-1/\beta}, & \text{Pareto.} \end{cases} \tag{20}$$

This shows that it is not possible to maintain a bounded convergence time by scaling d_n sublinearly. If we consider $d_n \in \Theta(n)$, e.g. $d_n = \delta n$, then $\sigma_2(P_{n,r}) \in \Theta(1)$ and the convergence time still diverges except in the uniform case. We remark that, on the contrary, the number of iterations K_ϵ is always $\Theta(1)$, showing once more the importance of considering the convergence time T_ϵ rather than K_ϵ.

B. *Expander graphs*

For rings, the clique is always the best topology. This appears to be a consequence of the fact that the spectral gap $\gamma(P_n)$ converges to 0 as n increases. In what follows we restrict ourselves to d-regular undirected non-bipartite graphs. For such graphs the largest eigenvalue of the adjacency matrix, A_n, is d. One can define their spectral gap as $\gamma(A_n) = d - \max_{|\lambda_i|<d} |\lambda_i(A_n)|$, where $\{\lambda_i(A_n)\}$ are the eigenvalues of A_n. If we select the weight matrix according to (19), one can show that $\gamma(P_n) \geq \gamma(A_n)/(d+1)$. Then, in general, the larger the spectral gap of the graph, the larger the spectral gap of the matrix P_n. Expander graphs with degree d are families of d-regular graphs with arbitrarily large numbers of nodes n, for which the spectral gap is bounded away from zero, i.e. there exists a constant $b > 0$ such that $\liminf_n \gamma(A_n) \geq b > 0$. Assuming we deal with an expander, at least asymptotically $\gamma(P_n)$ does not play an important role and only $g(d_n)$ matters. We can then expect that the slowdown due to nodes' dependencies has the largest effect on the computation time suggesting that the clique may not guarantee the fastest convergence. With high probability, random d-regular graphs are expanders with $\gamma(A_n) = d - 2\sqrt{d-1}$ [17]. Moreover, for random d-regular graphs our choice of weights (19) leads to $\gamma(P_n) = (d - 2\sqrt{d-1})/(d+1)$. We can then look for the value of d that minimizes the convergence time in (18). For example for $v = 1$ and n large enough, we obtain

$$d_{\text{opt}} \approx \begin{cases} n-1, & \text{uniform,} \\ 68, & \text{exponential,} \\ 21, & \text{Pareto.} \end{cases} \tag{21}$$

We can then conclude that in the exponential and Pareto cases there is a non-trivial sweet spot for the convergence time: the optimal topology is neither a clique nor a cycle.

Figure 4 shows how the optimal degree changes as the variability of the computation time increases (v ranges from 0 to 1) in a network with $n = 100$ nodes.

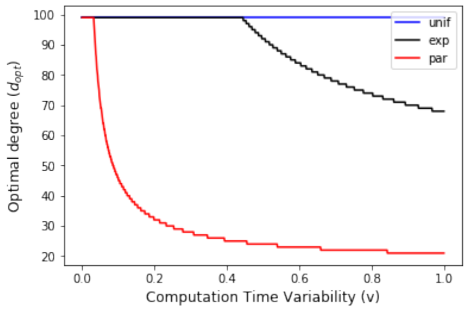


Fig. 4. The optimal degree for an expander versus the variability of the computation time. $n = 100$ nodes.

V. Numerical Evaluation

In this section we apply DSM to solve minimization problems that appear in machine learning tasks. Our numerical results have been obtained through a discrete event simulator developed in Python. We compare these results with our theoretical findings in the previous sections.

As typical supervised learning problems, we have considered a linear regression and a linear classifier respectively minimizing the mean squared error (MSE) $1/m \sum_{i=1}^{m} (\mathbf{x}^\intercal \chi_l - y_l)^2$ and the sum of hinge loss functions $1/m \sum_{i=1}^{m} \max(0, 1 - y_l \mathbf{x}^\intercal \chi_l)$. We carried out experiments both with synthetic datasets we generated (for which the underlying statistical model is known) and the following two datasets from the UCI Machine Learning Repository [18]:

a) CT: (named "Relative location of CT slices on axial axis" in the repository) has $m = 53500$ samples each with $p = 385$ features extracted from computer tomography images [19]. It is used for the regression problem to predict the relative location of the image on the axial axis.

b) SUSY: has $m = 5 \times 10^5$ samples each with $p = 18$ features consisting of kinematic properties measured by the particle detectors in an accelerator [20]. It is used for the classification problem to distinguish between a signal process that produces supersymmetric particles and a background process that does not.

In the minimization process we have considered $\alpha(k) \propto \sqrt{\gamma(P_n)}$, and undirected d-regular expander topologies[7] with $n = 1000$ nodes and different degrees d.

Figure 5 quantifies the tradeoff discussed in the paper when computation times are Pareto-distributed and $v = 1$. The number of iterations completed per node grows faster the less connected is the topology (Fig. 5 (a)), but for a given number of iterations, a more connected topology achieves a smaller MSE (Fig. 5 (b)). The overall effect is shown in Fig. 5 (c): the clique is far from optimal and expanders with degrees between $d = 16$ and $d = 40$ achieve the fastest decrease of MSE. Note how the optimal value predicted by our model ($d_{\text{opt}} = 21$) falls in this interval.

Figure 6 a) plots average convergence time needed to achieve an MSE equal to 1.5×10^4 calculated over 12 independent runs for the three different distributions. The empirical optimal degrees are $d = 999$, $d = 30$ and $d = 16$ respectively for the uniform, exponential and Pareto cases. Our analysis correctly predicts that the clique is optimal in the uniform case and overestimates the optimal degree for the exponential case ($d = 68$ vs $d = 30$). At the same time, setting a topology with degree $d = 68$ leads to a small increase (about 3%) of the convergence time in comparison to the empirical optimal setting $d = 30$. As already observed for Fig. 5 (c), our prediction is very accurate for the Pareto case. Overall, it seems that our model can effectively be used to tune the topology of the dependency graph. The figure shows that the convergence time for the clique is about 18% and 33% larger than for the best topology respectively in the exponential and the Pareto case. Considering also the advantage of a smaller communication overhead, sparse topologies seem to provide interesting advantages!

Figure 6 b) shows the convergence time needed to achieve a sum of hinge loss functions equal to 0.2. The conclusions are qualitatively the same of the regression problem, and even the relative improvements are almost the same. The results are also confirmed for the synthetic datasets. Our findings are then consistent across the different datasets and machine learning problems considered.

VI. Related work and Discussion

References [4], [5] have been discussed in Sect. II, because their bounds on the convergence rate in terms of iterations are the starting point of our work. Chapter 3 of [6] also concludes that a sparse dependency graph can minimize the convergence time, but for a different reason: reducing the connectivity reduces the amount of data to be transmitted which then can decrease the duration of one iteration. In this paper we show that a sparse topology may be preferable even if communication times are negligible. If not, the improvement from a sparse topology would be even larger. Reference [7] proves the optimal convergence rates for strongly convex and smooth distributed optimization considering a constant communication time. Our analysis can be easily adapted to extend the results in [6], [7] to take into account the variability of the computation and communication times.

Recently, [22] has studied the effect of a communication delay on the convergence rate of distributed gradient methods

[7]The graphs have been generated using the `NetworkX` implementation of the algorithm in [21].

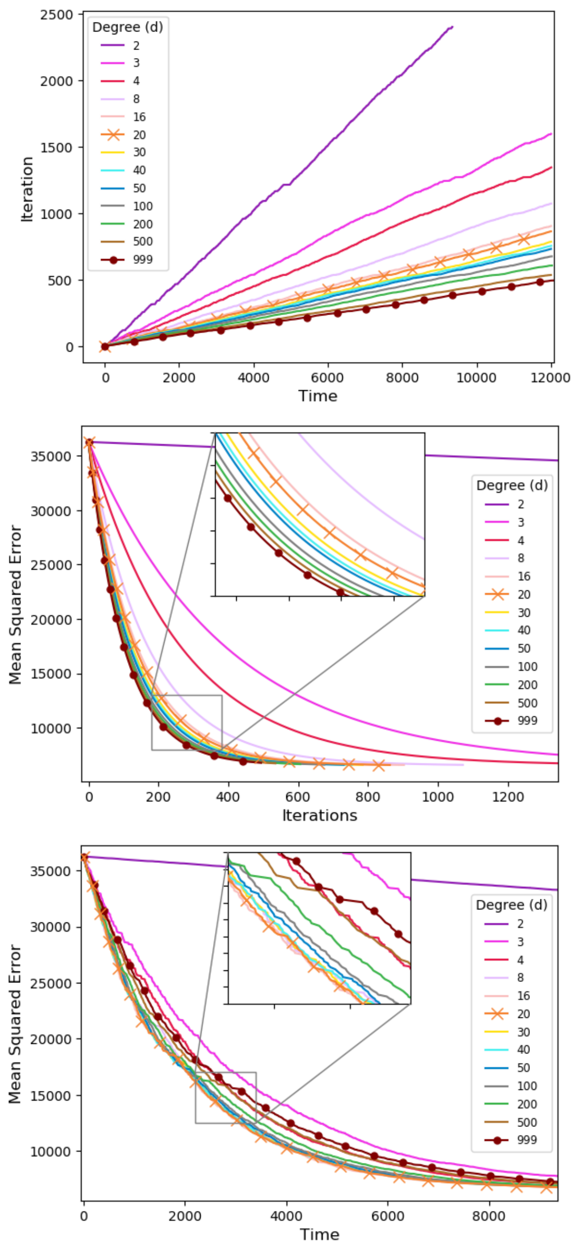
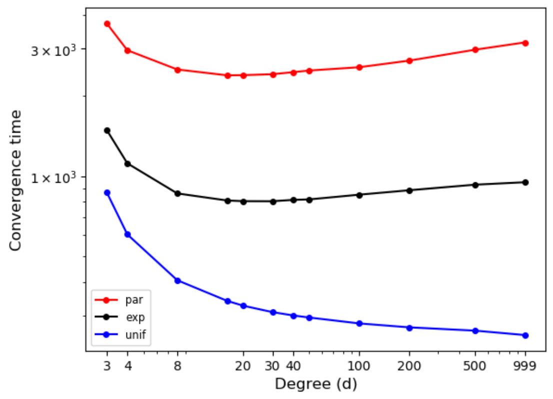


Fig. 5. Effect of network connectivity (degree d) on the convergence for linear regression on dataset CT with Pareto computation times. (a) Throughput, (b) Convergence in terms of number of iterations, (c) Convergence in terms of time.



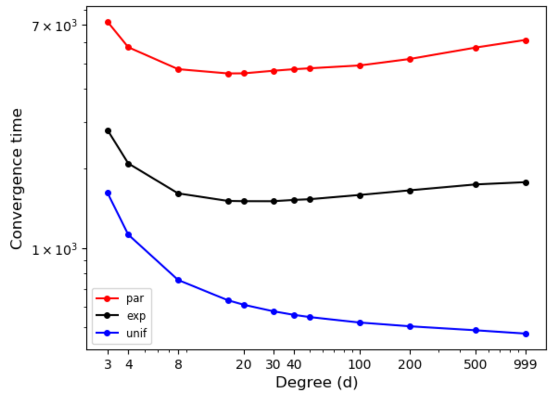


Fig. 6. Convergence time versus network connectivity (degree d) for different distributions of the computation times. (a) Linear regression on dataset CT, (b) Linear classification on dataset SUSY.

in terms of time, but the communication delay is constant and the operation is fully asynchronous with nodes working on stale information from their neighbours. In a heterogeneous setting where nodes compute at different speeds, asynchronicity can lead to convergence to a point that is not a minimizer. Our approach maintains a form of partial synchronism (a node waits for its predecessors), that prevents this problem.

Recently some straggling mitigation techniques for distributed gradient methods have been proposed [13], [12]. They have both been designed for asynchronous operation in a parameter server framework. Moreover, they rely on data replication across the executors [13] and some form of data encoding [12]. Our approach requires a form of partial synchronicity but does not need to replicate or pre-process the dataset.

VII. Conclusions

In this paper we have presented a model for distributed subgradient methods with fine-grained synchronization. We have characterized the throughput of the system (number of tasks completed per time unit) as a function of the network topology and of the distribution of the computation times. In particular, we have proved bounds as well as provided an approximate expression for the throughput. Using the results of our analysis, we have then determined the topology leading to the smallest convergence time. Our conclusion, confirmed by numerical results with real datasets, shows that limiting the network connectivity can lead to significant reduction of the convergence time. It then contradicts the common belief that a clique would always lead to the best performance.

This work was supported in part by the NSF under grant CNS-1617437 and by ARL under Cooperative Agreement W911NF-17-2-0196.

References

[1] F. Niu, B. Recht, C. Re, and S. J. Wright, “Hogwild!: A lock-free approach to parallelizing stochastic gradient descent,” in *Proc. of the 24th Intl Conference on Neural Information Processing Systems*, ser. NIPS’11. USA: Curran Associates Inc., 2011, pp. 693–701.

[2] A. Kadav and E. Kruus, “ASAP: asynchronous approximate data-parallel computation,” *CoRR*, vol. abs/1612.08608, 2016.

[3] J. E. Gonzalez, P. Bailis, M. I. Jordan, M. J. Franklin, J. M. Hellerstein, A. Ghodsi, and I. Stoica, “Asynchronous complex analytics in a distributed dataflow architecture,” *CoRR*, vol. abs/1510.07092, 2015.

[4] J. C. Duchi, A. Agarwal, and M. J. Wainwright, “Dual averaging for distributed optimization: Convergence analysis and network scaling,” *IEEE Trans. on Automatic Control*, vol. 57, no. 3, pp. 592–606, 2012.

[5] A. Nedić, A. Olshevsky, and M. G. Rabbat, “Network topology and communication-computation tradeoffs in decentralized optimization,” *Proc. of the IEEE*, vol. 106, no. 5, pp. 953–976, May 2018.

[6] K. I. Tsianos, “The role of the network in distributed optimization algorithms,” Ph.D. dissertation, McGill University, 2013.

[7] K. Scaman, F. Bach, S. Bubeck, Y. T. Lee, and L. Massoulié, “Optimal algorithms for smooth and strongly convex distributed optimization in networks,” in *Proc. of the 34th ICML*, 2017.

[8] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*. Springer, 2014.

[9] S. Bubeck, “Convex optimization: Algorithms and complexity,” *Foundations and Trends in Machine Learning*, vol. 8, no. 3-4, 2015.

[10] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server,” in *Proc. of the 11th USENIX OSDI Conf.*, 2014.

[11] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, “Effective straggler mitigation: Attack of the clones,” in *Proc. of the 10th USENIX Conf. NSDI*, 2013.

[12] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, “Straggler mitigation in distributed optimization through data encoding,” in *Proc. of NIPS)*, 2017, pp. 5434–5442.

[13] S. Li, M. Mousavi Kalan, A. S. Avestimehr, and M. Soltanolkotabi, “Near-Optimal Straggler Mitigation for Distributed Gradient Methods,” in *Proc. of the 7th Intl. Workshop ParLearning*, May 2018.

[14] A. Nedic and A. E. Ozdaglar, “Distributed subgradient methods for multi-agent optimization,” *IEEE Trans. Automat. Contr.*, vol. 54, no. 1, pp. 48–61, 2009.

[15] Y. Dallery, Z. Liu, and D. Towsley, “Equivalence, reversibility, symmetry and concavity properties in fork-join queuing networks with blocking,” *J. ACM*, vol. 41, no. 5, pp. 903–942, Sep. 1994.

[16] Y. Zeng, A. Chaintreau, D. Towsley, and C. H. Xia, “Throughput scalability analysis of fork-join queueing networks,” *Operations Research*, 2018.

[17] B. D. McKay, “The expected eigenvalue distribution of a large regular graph,” *Linear Algebra and its Applications*, vol. 40, pp. 203 – 216, 1981.

[18] “UCI Machine Learning Repository,” archive.ics.uci.edu/ml/datasets/.

[19] F. Graf, H.-P. Kriegel, M. Schubert, S. Pölsterl, and A. Cavallaro, “2D Image Registration in CT Images Using Radial Image Descriptors,” in *Proc. of MICCAI*. Berlin, Heidelberg: Springer, 2011, pp. 607–614.

[20] P. Baldi, P. Sadowski, and D. O. Whiteson, “Searching for exotic particles in high-energy physics with deep learning.” *Nature communications*, vol. 5, p. 4308, 2014.

[21] B. D. McKay and N. C. Wormald, “Uniform generation of random regular graphs of moderate degree,” *J. Algorithms*, vol. 11, no. 1, pp. 52–67, Feb. 1990.

[22] T. Doan, C. Beck, and R. Srikant, “Impact of communication delays on the convergence rate of distributed optimization algorithms,” in *Proc. of ACM SIGMETRICS 2018*, Jun. 2018.