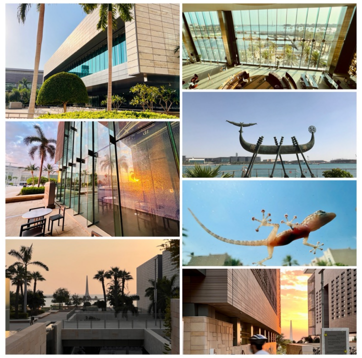
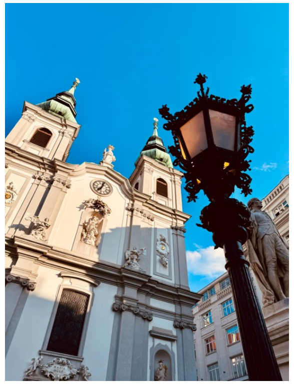


KAUST

King Abdullah University of Science and Technology



Vienna

# The First Optimal Distributed SGD

## (in the Presence of Data, Compute and Communication Heterogeneity)


**Peter Richtárik**

King Abdullah University of Science and Technology

Kingdom of Saudi Arabia


One World Optimization Seminar in Vienna, June 3-7, 2024

ESI Erwin Schrödinger International Institute for Mathematics and Physics

**Peter Richtarik**
@peter_richtarik

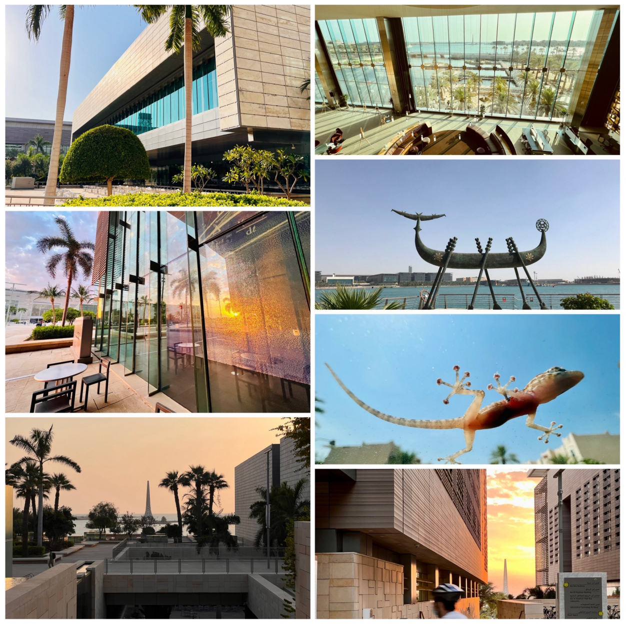
When you get what you didn't know you needed 😂 (courtesy of The Erwin Schroedinger International Institute For Mathematics and Physics, Vienna)

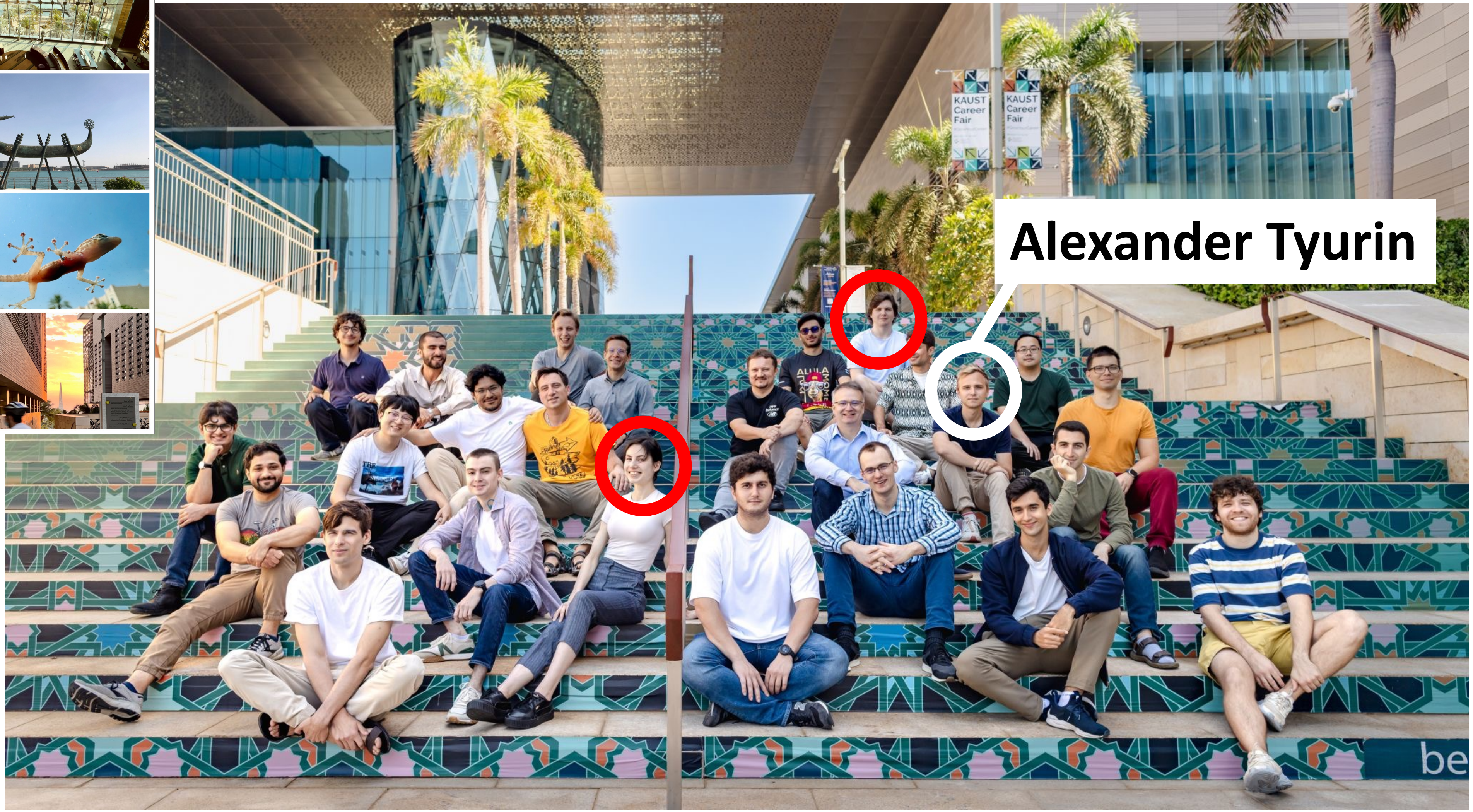
17:02 · 03/06/2024 From Earth · **8,2K** Views

View analytics

**5** Reposts **3** Quotes **73** Likes **8** Bookmarks

# Optimization & Machine Learning Lab @ KAUST





# Part 1
# Introduction

# Optimization Problem

$$\min_{x \in \mathbb{R}^d} f(x) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

# parallel machines

# model parameters / features

Loss on local data $\mathcal{D}_i$ stored on machine $i$

$$f_i(x) := \mathbb{E}_{\xi \sim \mathcal{D}_i} \left[ f(x, \xi) \right]$$

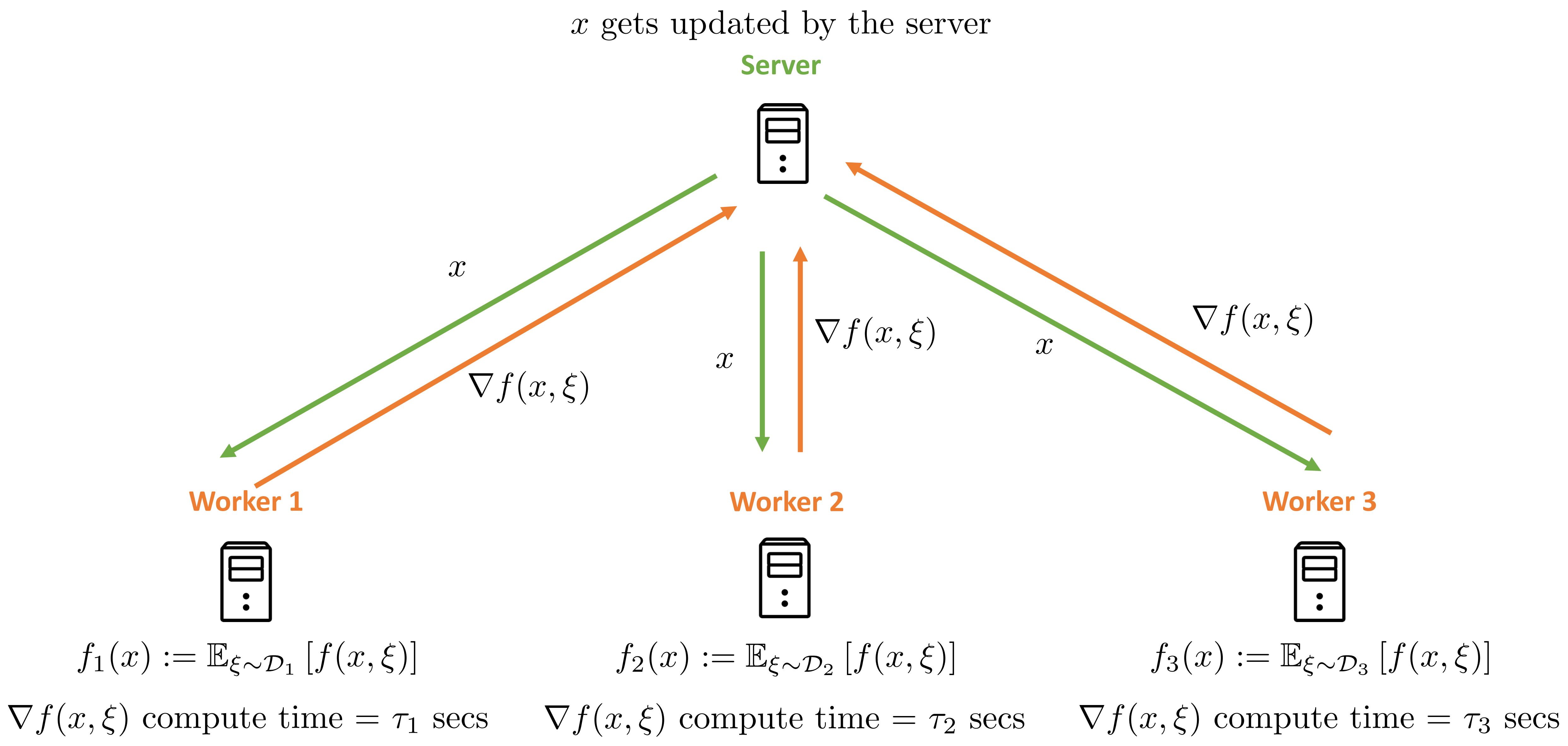
! It takes $\tau_i$ seconds for worker $i$ to compute $\nabla f(x, \xi)$, where $\xi \sim \mathcal{D}_i$

It takes $\theta_i$ seconds for worker $i$ to communicate $g \in \mathbb{R}^d$ to the server

$0 < \tau_1 \leq \tau_2 \leq \cdots \leq \tau_n$

Find a (possibly random) vector $\hat{x} \in \mathbb{R}^d$ such that $\mathbb{E}\left[ \|\nabla f(\hat{x})\|^2 \right] \leq \varepsilon$

# Parallel Computing Architecture



# Three Types of Heterogeneity

| | |
|---|---|
| **Data** | data distributions $\mathcal{D}_1, \ldots, \mathcal{D}_n$ can be different |
| **Compute** | compute times $\tau_1, \ldots, \tau_n$ are nonzero and can be different |
| **Communication** | communication times $\theta_1, \ldots, \theta_n$ are nonzero and can be different |

# Typical Assumptions

1. $\inf f \in \mathbb{R}$

2. $f_i(x) := \mathbb{E}_{\xi \sim \mathcal{D}_i}\left[f(x,\xi)\right]$

Gradient of local functions is Lipschitz:

$$\max_{i\in\{1,\dots,n\}} \sup_{x\neq y} \frac{\|\nabla f_i(x) - \nabla f_i(y)\|}{\|x-y\|} \leq L$$

Stochastic gradients have bounded variance:

$$\max_{i\in\{1,\dots,n\}} \sup_{x\in\mathbb{R}^d} \mathbb{E}_{\xi\sim\mathcal{D}_i}\left[\|\nabla f(x,\xi) - \mathbb{E}_{\xi\sim\mathcal{D}_i}\left[\nabla f(x,\xi)\right]\|^2\right] \leq \sigma^2$$

# Our Papers

***First optimal parallel SGD under…***

5/2023

Rennala SGD
Malenia SGD
Acc. Rennala SGD

Alexander Tyurin and P.R.
**Optimal time complexities of parallel stochastic optimization methods under a fixed computation model**
*NeurIPS 2023*

*… **computation** (and/or data) **heterogeneity***

2/2024

Shadowheart SGD

Alexander Tyurin, Marta Pozzi, Ivan Ilin and P.R.
**Shadowheart SGD: Distributed asynchronous SGD with optimal time complexity under arbitrary computation and communication heterogeneity**
*arXiv:2402.04785, 2024*

*… **communication** (and computation) **heterogeneity***

*[Rennala SGD as a special case]*

5/2024

Freya PAGE
Freya SGD

Alexander Tyurin, Kaja Gruntkowska, and P.R.
**Freya PAGE: First optimal time complexity for large-scale nonconvex finite-sum optimization with heterogeneous asynchronous computations**
*arXiv:2405.1554, 2024*

*… computation heterogeneity for **finite-sum** problems*

in the large-scale regime: $m \geq n^2$

5/2024

Fragile SGD, Amelie SGD
+ accelerated variants

Alexander Tyurin and P.R.
**On the optimal time complexities in decentralized stochastic asynchronous optimization**
*arXiv:2405.16218, 2024*

*… computation and communication heterogeneity in the **decentralized setup***

# Peter, What About the Weird Algorithm Names?



Rennala, Queen of the Full Moon is a Legend Boss in Elden Ring. Though not a demigod, Rennala is one of the shardbearers who resides in the Academy of Raya Lucaria. Rennala is a powerful sorceress, head of the Carian Royal family, and erstwhile leader of the Academy.

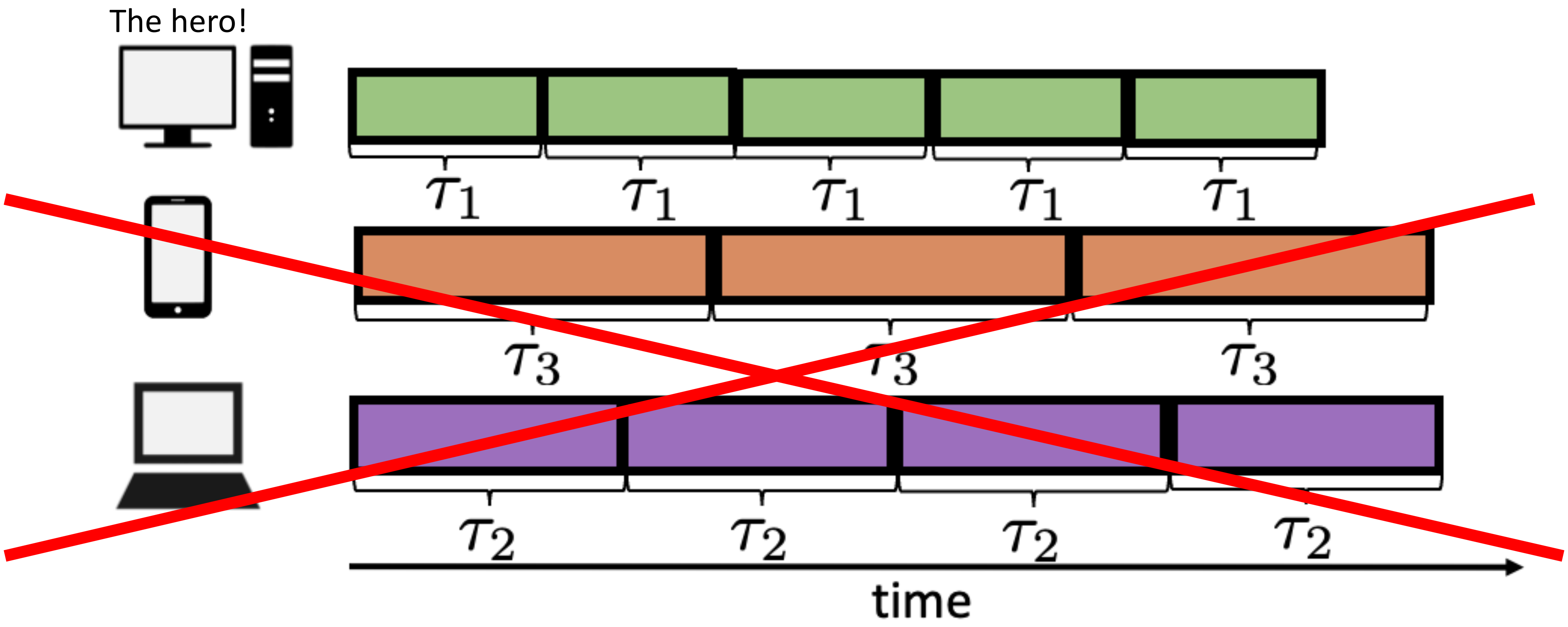

# Optimal Parallel Stochastic Gradient Methods

| | Data Heterogeneity ($\mathcal{D}_i$ different) | Compute Heterogeneity ($\tau_i$ different) | Communication Heterogeneity ($\theta_i$ different) | Smooth Nonconvex | Smooth Convex | Infinite / Finite Sum? | Supports Decentralized Setup? | Optimal Time Complexity? |
|---|---|---|---|---|---|---|---|---|
| **Rennala SGD** Tyurin & R (NeurIPS '23) | ✗ | ✓ | 0 | ✓ | | **Inf** | ✗ | ✓ |
| **Malenia SGD** Tyurin & R (NeurIPS '23) | ✓ | ✓ | 0 | ✓ | | **Inf** | ✗ | ✓ |
| **Accelerated Rennala SGD** Tyurin & R (NeurIPS '23) | ✗ | ✓ | 0 | | ✓ | **Inf** | ✗ | ✓ |
| **Shadowheart SGD** Tyurin, Pozzi, Ilin & R '24 | ✗ | ✓ | ✓ | ✓ | | **Inf** | ✗ | ✓ |
| **Freya PAGE** Tyurin, Gruntkowska & R '24 | ✗ | ✓ | 0 | ✓ | | **Finite** | ✗ | ✓ big data regime |
| **Freya SGD** Tyurin, Gruntkowska & R '24 | ✗ | ✓ | 0 | ✓ | | **Finite** | ✗ | ✗ |
| **Fragile SGD** Tyurin & R '24 | ✗ | ✓ | ✓ | ✓ | | **Inf** | ✓ | nearly |
| **Amelie SGD** Tyurin & R '24 | ✓ | ✓ | ✓ | ✓ | | **Inf** | ✓ | ✓ |

# Part 2
# Previous Approaches
# to Parallelizing SGD

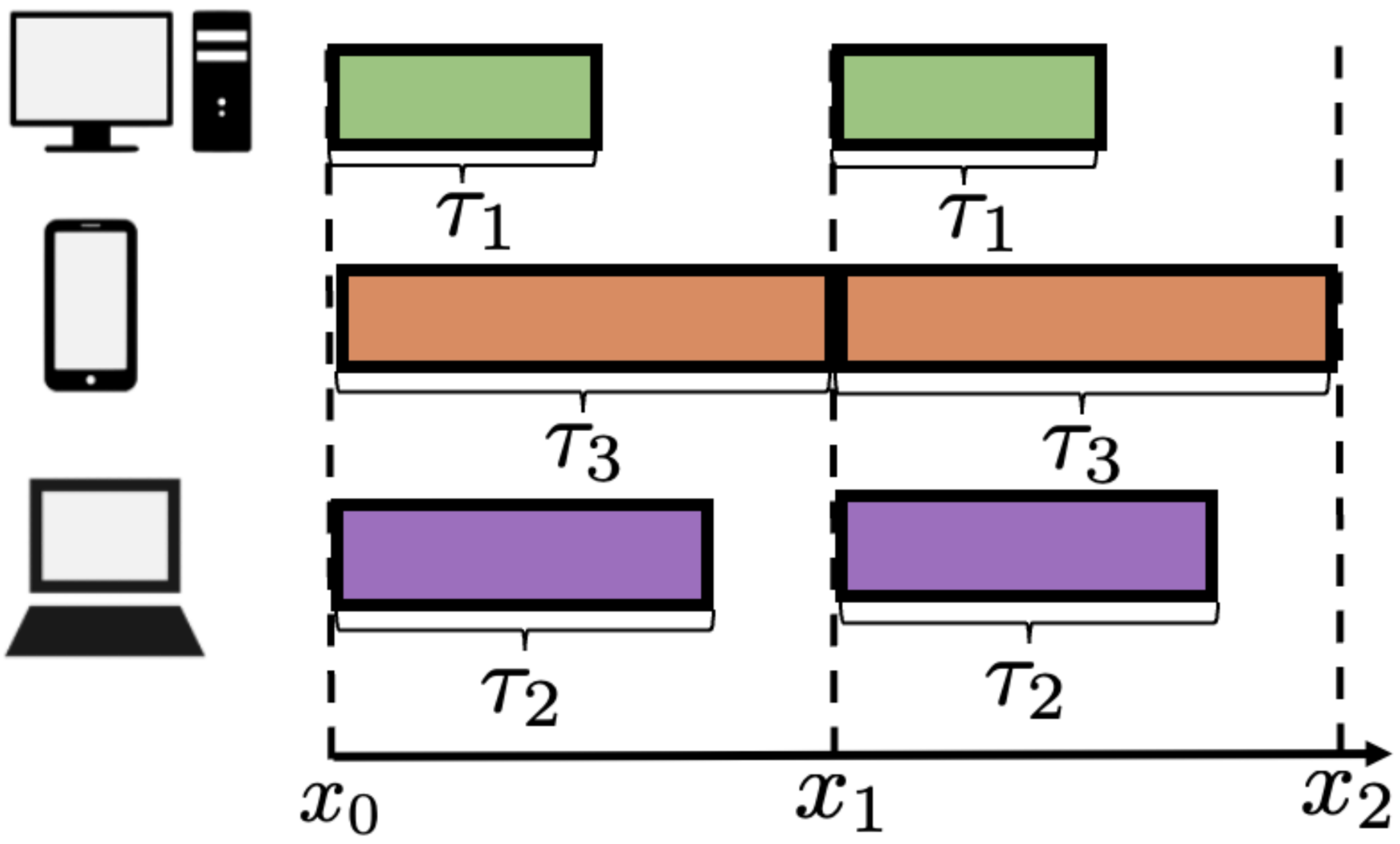
# Hero SGD

Algorithmic idea: The fastest worker does it all!



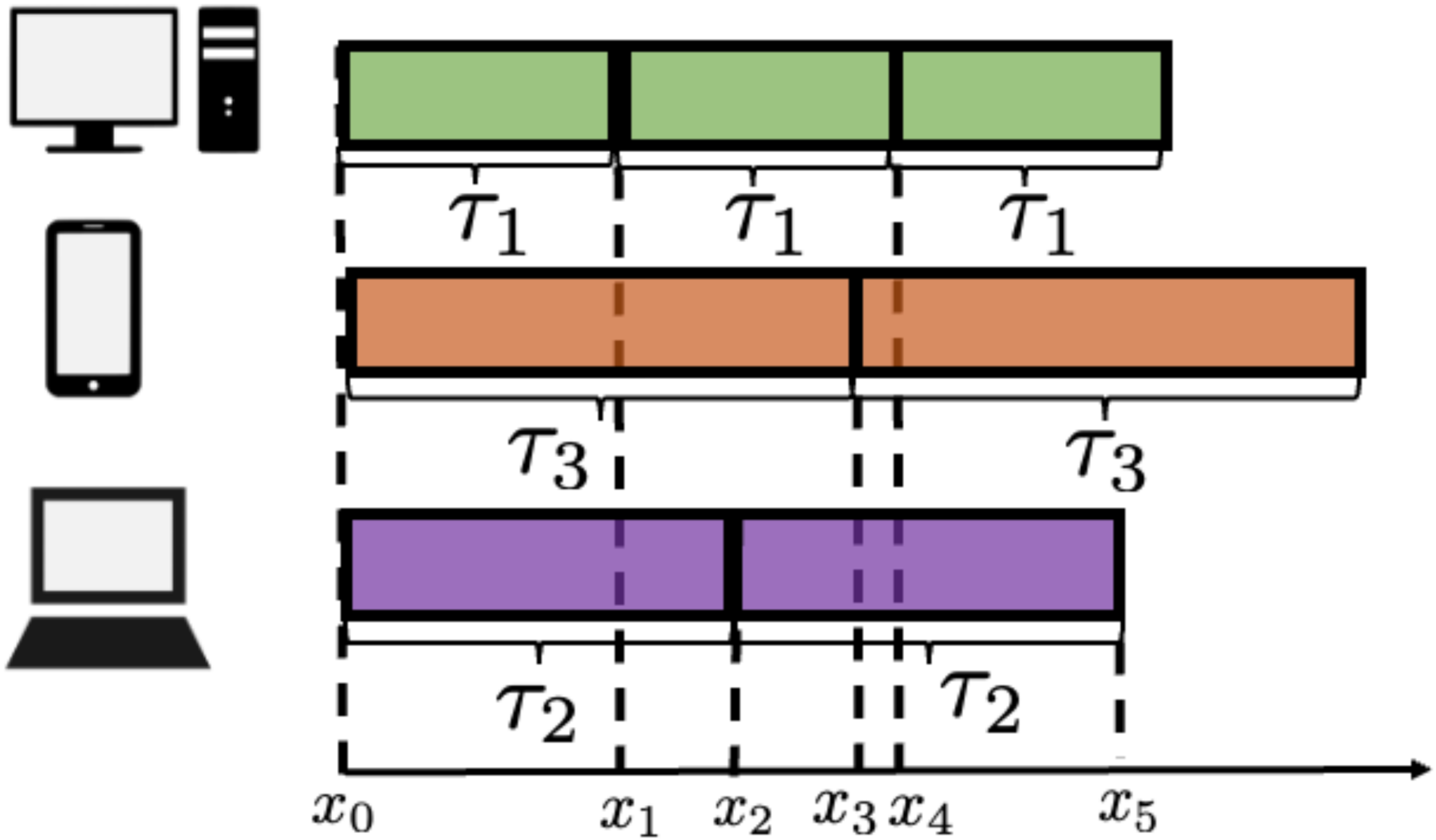
# (Fair) Minibatch SGD

Algorithmic idea: Each worker does one job only!



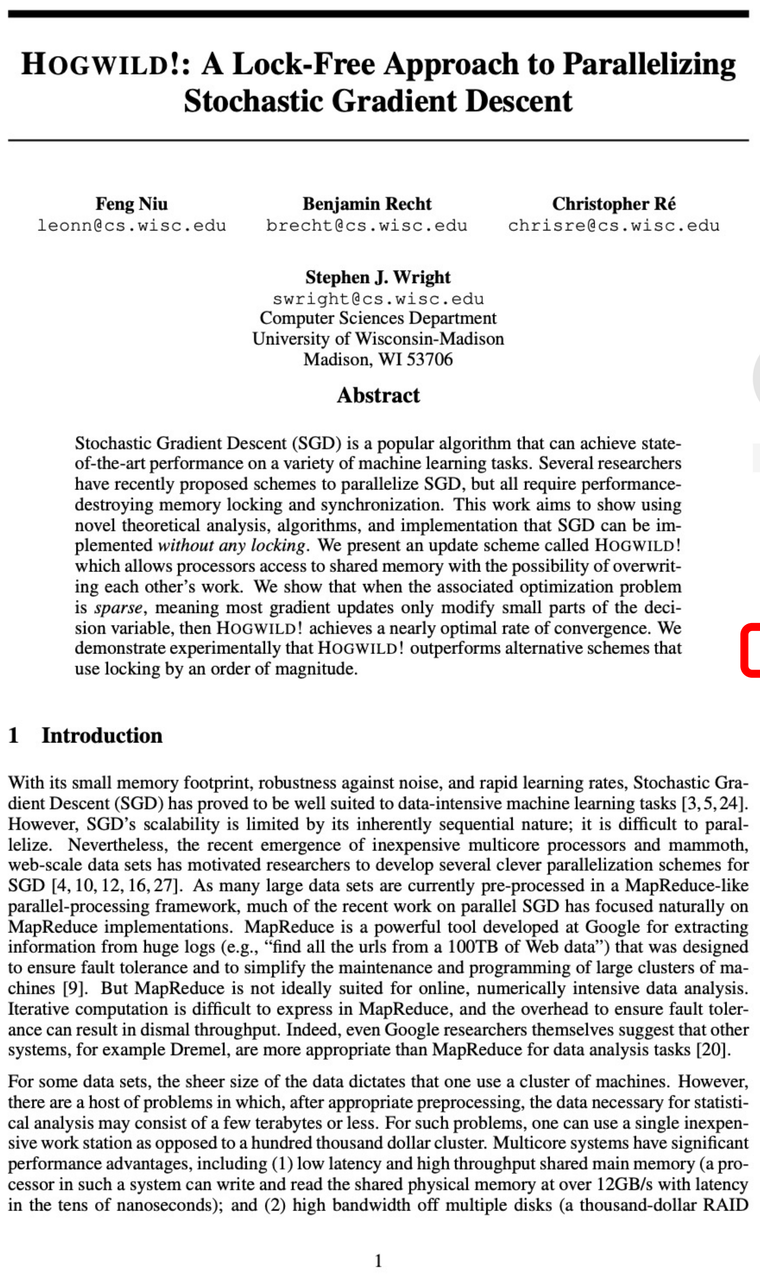
# Asynchronous SGD

Algorithmic idea: All workers are slaves and useful



published in NIPS 2011

**NeurIPS 2020 Test of Time Award**



# HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent


**Feng Niu** leonn@cs.wisc.edu
**Benjamin Recht** brecht@cs.wisc.edu
**Christopher Ré** chrisre@cs.wisc.edu

**Stephen J. Wright**
swright@cs.wisc.edu
Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706



## Abstract

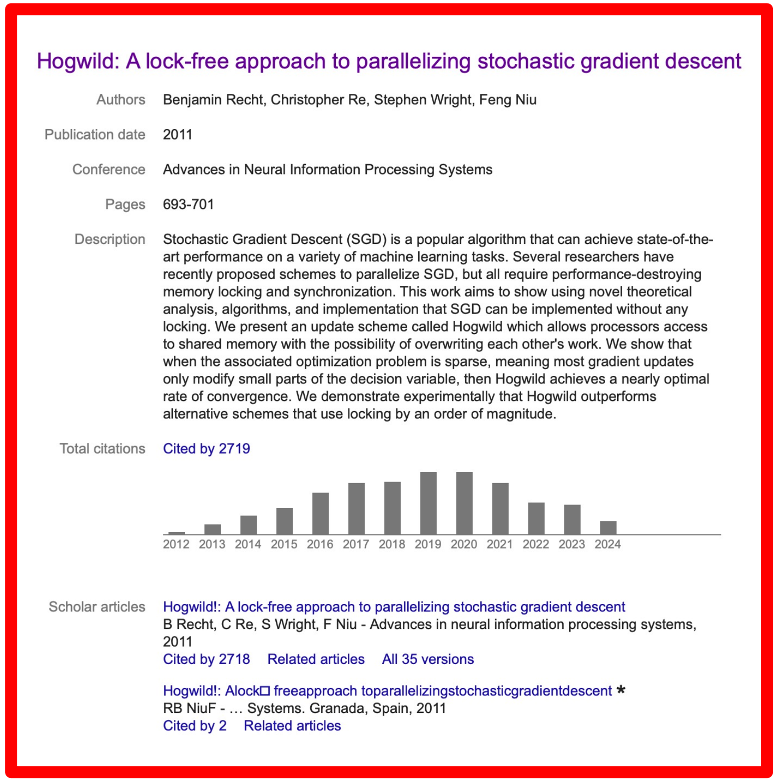
Stochastic Gradient Descent (SGD) is a popular algorithm that can achieve state-of-the-art performance on a variety of machine learning tasks. Several researchers have recently proposed schemes to parallelize SGD, but all require performance-destroying memory locking and synchronization. This work aims to show using novel theoretical analysis, algorithms, and implementation that SGD can be implemented *without any locking*. We present an update scheme called HOGWILD! which allows processors access to shared memory with the possibility of overwriting each other's work. We show that when the associated optimization problem is *sparse*, meaning most gradient updates only modify small parts of the decision variable, then HOGWILD! achieves a nearly optimal rate of convergence. We demonstrate experimentally that HOGWILD! outperforms alternative schemes that use locking by an order of magnitude.


## 1 Introduction

With its small memory footprint, robustness against noise, and rapid learning rates, Stochastic Gradient Descent (SGD) has proved to be well suited to data-intensive machine learning tasks [3, 5, 24]. However, SGD's scalability is limited by its inherently sequential nature; it is difficult to parallelize. Nevertheless, the recent emergence of inexpensive multicore processors and mammoth, web-scale data sets has motivated researchers to develop several clever parallelization schemes for SGD [4, 10, 12, 16, 27]. As many large data sets are currently pre-processed in a MapReduce-like parallel-processing framework, much of the recent work on parallel SGD has focused naturally on MapReduce implementations. MapReduce is a powerful tool developed at Google for extracting information from huge logs (e.g., "find all the urls from a 100TB of Web data") that was designed to ensure fault tolerance and to simplify the maintenance and programming of large clusters of machines [9]. But MapReduce is not ideally suited for online, numerically intensive data analysis. Iterative computation is difficult to express in MapReduce, and the overhead to ensure fault tolerance can result in dismal throughput. Indeed, even Google researchers themselves suggest that other systems, for example Dremel, are more appropriate than MapReduce for data analysis tasks [20].

For some data sets, the sheer size of the data dictates that one use a cluster of machines. However, there are a host of problems in which, after appropriate preprocessing, the data necessary for statistical analysis may consist of a few terabytes or less. For such problems, one can use a single inexpensive work station as opposed to a hundred thousand dollar cluster. Multicore systems have significant performance advantages, including (1) low latency and high throughput shared main memory (a processor in such a system can write and read the shared physical memory at over 12GB/s with latency in the tens of nanoseconds); and (2) high bandwidth off multiple disks (a thousand-dollar RAID



# Our Inspiration: Two Beautiful Papers

## Asynchronous SGD Beats Minibatch SGD Under Arbitrary Delays

Konstantin Mishchenko Francis Bach Mathieu Even Blake Woodworth

DI ENS, Ecole normale supérieure,
Université PSL, CNRS, INRIA
75005 Paris, France

### Abstract

The existing analysis of asynchronous stochastic gradient descent (SGD) degrades dramatically when any delay is large, giving the impression that performance depends primarily on the delay. On the contrary, we prove much better guarantees for the same asynchronous SGD algorithm regardless of the delays in the gradients, depending instead just on the number of parallel devices used to implement the algorithm. Our guarantees are strictly better than the existing analyses, and we also argue that asynchronous SGD outperforms synchronous minibatch SGD in the settings we consider. For our analysis, we introduce a novel recursion based on "virtual iterates" and delay-adaptive stepsizes, which allow us to derive state-of-the-art guarantees for both convex and non-convex objectives.

### 1 Introduction

We consider solving stochastic optimization problems of the form

$$\min_{\mathbf{x}\in\mathbb{R}^d} \{F(\mathbf{x}) := \mathbb{E}_{\xi\sim\mathcal{D}} f(\mathbf{x};\xi)\}, \tag{1}$$

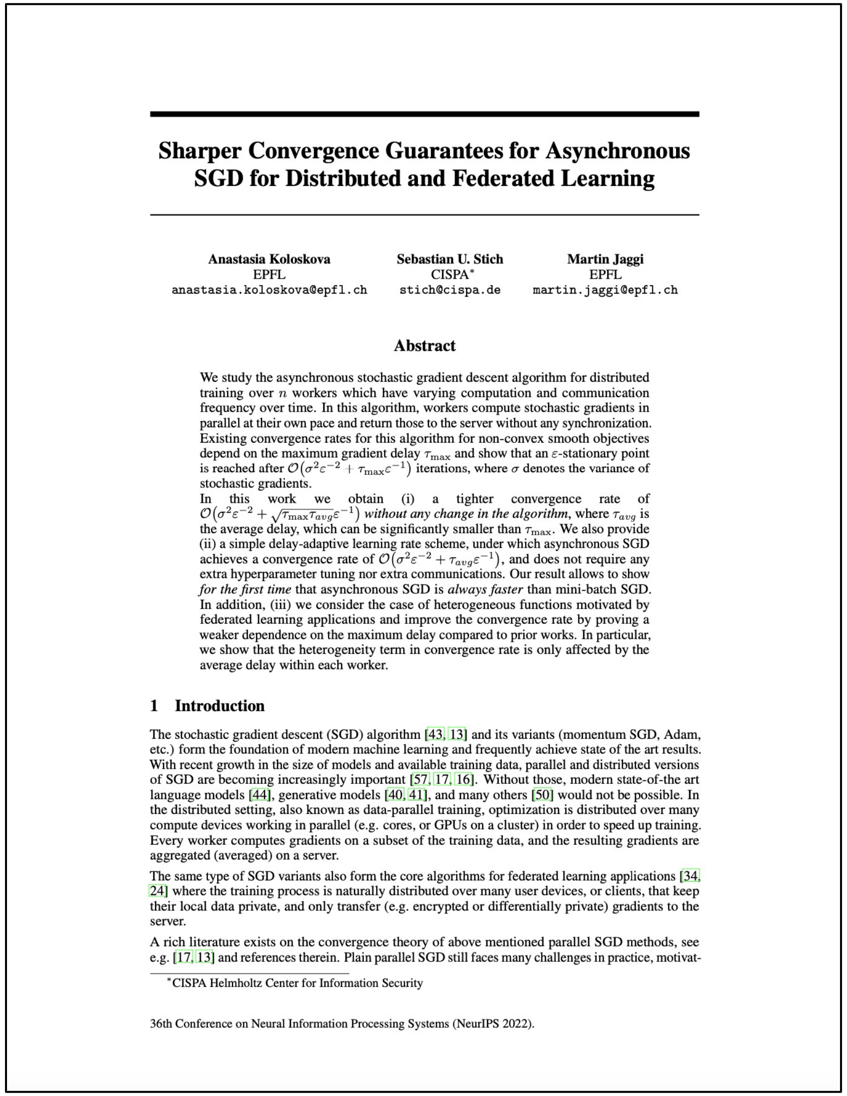
which includes machine learning (ML) training objectives, where $f(\mathbf{x};\xi)$ represents the loss of a model parameterized by $\mathbf{x}$ on the datum $\xi$. Depending on the application, $\mathcal{D}$ could represent a finite dataset of size $n$ or a population distribution. In recent years, such stochastic optimization problems have continued to grow rapidly in size, both in terms of the dimension $d$ of the optimization variable—i.e., the number of model parameters in ML—and in terms of the quantity of data—i.e., the number of samples $\xi_1,\ldots,\xi_n \sim \mathcal{D}$ being used. With $d$ and $n$ regularly reaching the tens or hundreds of billions, it is increasingly necessary to use parallel optimization algorithms to handle the large scale and to benefit from data stored on different machines.

There are many ways of employing parallelism to solve (1), but the most popular approaches in practice are first-order methods based on stochastic gradient descent (SGD). At each iteration, SGD employs stochastic estimates of $\nabla F$ to update the parameters as $\mathbf{x}_k = \mathbf{x}_{k-1} - \gamma_k \nabla f(\mathbf{x}_{k-1};\xi_{k-1})$ for an i.i.d. sample $\xi_{k-1}\sim\mathcal{D}$. Given $M$ machines capable of computing these stochastic gradient estimates $\nabla f(\mathbf{x};\xi)$ in parallel, one approach to parallelizing SGD is what we call "Minibatch SGD." This refers to a synchronous, parallel algorithm that dispatches the current parameters $\mathbf{x}_{k-1}$ to each of the $M$ machines, waits while they compute and communicate back their gradient estimates $\mathbf{g}^1_{k-1},\ldots,\mathbf{g}^M_{k-1}$, and then takes a minibatch SGD step $\mathbf{x}_k = \mathbf{x}_{k-1} - \gamma_k\cdot\frac{1}{M}\sum_{m=1}^M \mathbf{g}^m_{k-1}$. This is a natural idea with long history [16, 18, 55] and it is a commonly used in practice [e.g., 22]. However, since Minibatch SGD waits for all $M$ of the machines to finish computing their gradient estimates before updating, it proceeds only at the speed of the *slowest* machine.

There are several possible sources of delays: nodes may have heterogeneous hardware with different computational throughputs [23, 25], network latency can slow the communication of gradients, and

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

arXiv: June 15, 2022



## Sharper Convergence Guarantees for Asynchronous SGD for Distributed and Federated Learning

Anastasia Koloskova, EPFL, anastasia.koloskova@epfl.ch
Sebastian U. Stich, CISPA*, stich@cispa.de
Martin Jaggi, EPFL, martin.jaggi@epfl.ch

### Abstract

We study the asynchronous stochastic gradient descent algorithm for distributed training over $n$ workers which have varying computation and communication frequency over time. In this algorithm, workers compute stochastic gradients in parallel at their own pace and return those to the server without any synchronization. Existing convergence rates for this algorithm for non-convex smooth objectives depend on the maximum gradient delay $\tau_{\max}$ and show that an $\varepsilon$-stationary point is reached after $\mathcal{O}(\sigma^2\varepsilon^{-2} + \tau_{\max}\varepsilon^{-1})$ iterations, where $\sigma$ denotes the variance of stochastic gradients.

In this work we obtain (i) a tighter convergence rate of $\mathcal{O}(\sigma^2\varepsilon^{-2} + \sqrt{\tau_{\max}\tau_{avg}}\varepsilon^{-1})$ *without any change in the algorithm*, where $\tau_{avg}$ is the average delay, which can be significantly smaller than $\tau_{\max}$. We also provide (ii) a simple delay-adaptive learning rate scheme, under which asynchronous SGD achieves a convergence rate of $\mathcal{O}(\sigma^2\varepsilon^{-2} + \tau_{avg}\varepsilon^{-1})$, and does not require any extra hyperparameter tuning nor extra communications. Our result allows to show *for the first time* that asynchronous SGD is *always faster* than mini-batch SGD. In addition, (iii) we consider the case of heterogeneous functions motivated by federated learning applications and improve the convergence rate by proving a weaker dependence on the maximum delay compared to prior works. In particular, we show that the heterogeneity term in convergence rate is only affected by the average delay within each worker.

### 1 Introduction

The stochastic gradient descent (SGD) algorithm [43, 13] and its variants (momentum SGD, Adam, etc.) form the foundation of modern machine learning and frequently achieve state of the art results. With recent growth in the size of models and available training data, parallel and distributed versions of SGD are becoming increasingly important [57, 17, 16]. Without those, modern state-of-the art language models [44], generative models [40, 41], and many others [50] would not be possible. In the distributed setting, also known as data-parallel training, optimization is distributed over many compute devices working in parallel (e.g. cores, or GPUs on a cluster) in order to speed up training. Every worker computes gradients on a subset of the training data, and the resulting gradients are aggregated (averaged) on a server.

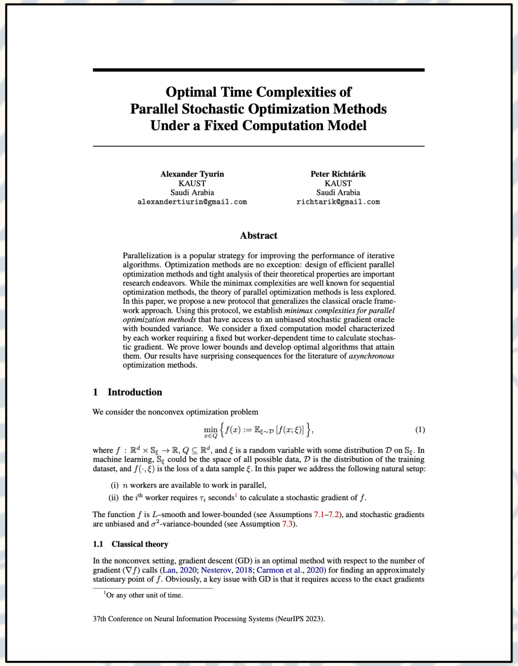
The same type of SGD variants also form the core algorithms for federated learning applications [34, 24] where the training process is naturally distributed over many user devices, or clients, that keep their local data private, and only transfer (e.g. encrypted or differentially private) gradients to the server.

A rich literature exists on the convergence theory of above mentioned parallel SGD methods, see e.g. [17, 13] and references therein. Plain parallel SGD still faces many challenges in practice, motivat-

*CISPA Helmholtz Center for Information Security

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

arXiv: June 16, 2022



Optimal Time Complexities of Parallel Stochastic Optimization Methods Under a Fixed Computation Model


Alexander Tyurin
KAUST
Saudi Arabia
alexandertiurin@gmail.com

Peter Richtárik
KAUST
Saudi Arabia
richtarik@gmail.com


Abstract


Parallelization is a popular strategy for improving the performance of iterative algorithms. Optimization methods are no exception: design of efficient parallel optimization methods and tight analysis of their theoretical properties are important research endeavors. While the minimax complexities are well known for sequential optimization methods, the theory of parallel optimization methods is less explored. In this paper, we propose a new protocol that generalizes the classical oracle framework approach. Using this protocol, we establish *minimax complexities for parallel optimization methods* that have access to an unbiased stochastic gradient oracle with bounded variance. We consider a fixed computation model characterized by each worker requiring a fixed but worker-dependent time to calculate stochastic gradient. We prove lower bounds and develop optimal algorithms that attain them. Our results have surprising consequences for the literature of *asynchronous* optimization methods.


1 Introduction

We consider the nonconvex optimization problem

$$\min_{x \in Q} \left\{ f(x) := \mathbb{E}_{\xi \sim \mathcal{D}} \left[ f(x;\xi) \right] \right\}, \quad (1)$$

where $f : \mathbb{R}^d \times \mathbb{S}_\xi \to \mathbb{R}$, $Q \subseteq \mathbb{R}^d$, and $\xi$ is a random variable with some distribution $\mathcal{D}$ on $\mathbb{S}_\xi$. In machine learning, $\mathbb{S}_\xi$ could be the space of all possible data, $\mathcal{D}$ is the distribution of the training dataset, and $f(\cdot, \xi)$ is the loss of a data sample $\xi$. In this paper we address the following natural setup:

(i) $n$ workers are available to work in parallel,

(ii) the $i^{\text{th}}$ worker requires $\tau_i$ seconds[1] to calculate a stochastic gradient of $f$.

The function $f$ is $L$–smooth and lower-bounded (see Assumptions 7.1–7.2), and stochastic gradients are unbiased and $\sigma^2$-variance-bounded (see Assumption 7.3).

1.1 Classical theory

In the nonconvex setting, gradient descent (GD) is an optimal method with respect to the number of gradient ($\nabla f$) calls (Lan, 2020; Nesterov, 2018; Carmon et al., 2020) for finding an approximately stationary point of $f$. Obviously, a key issue with GD is that it requires access to the exact gradients

[1] Or any other unit of time.


37th Conference on Neural Information Processing Systems (NeurIPS 2023).


# Part 3
# Rennala SGD

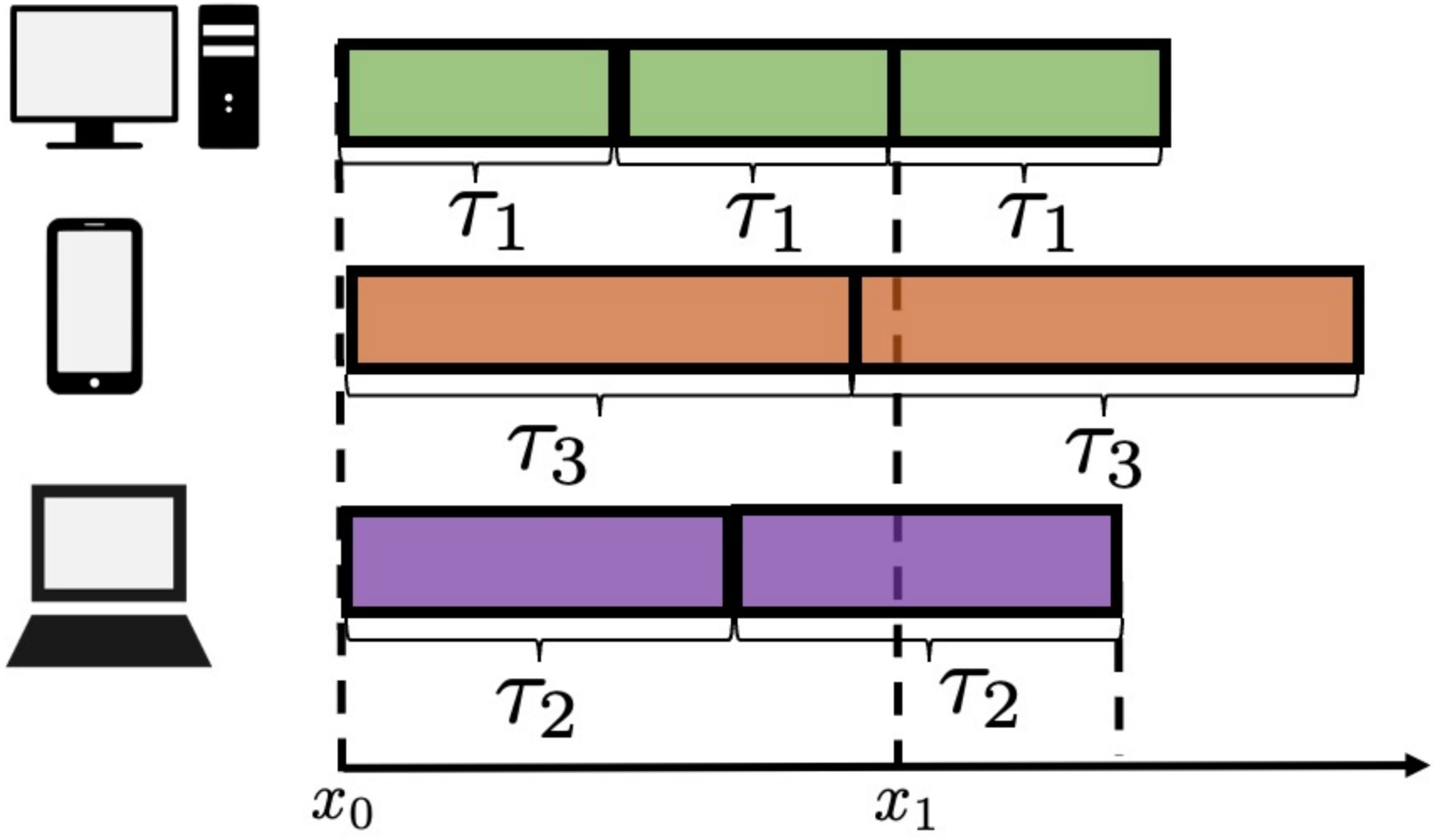
Alexander Tyurin and P.R.

**Optimal time complexities of parallel stochastic optimization methods under a fixed computation model**

*NeurIPS 2023*

# Rennala SGD

Algorithmic idea: Minibatch SGD with asynchronous minibatch collection



# Upper Bound

**Theorem (informal)**

Assume data homogeneity and zero communication times.
Then Rennala SGD solves the problem in

$$96 \times \min_{m \in \{1,\ldots,n\}} \left(\frac{1}{m}\sum_{i=1}^{m}\frac{1}{\tau_i}\right)^{-1}\left(\frac{L\Delta}{\varepsilon} + \frac{L\Delta\sigma^2}{\varepsilon^2 m}\right)$$

seconds.

Number of parallel machines

Gradient of $f$ is $L$-Lipschitz

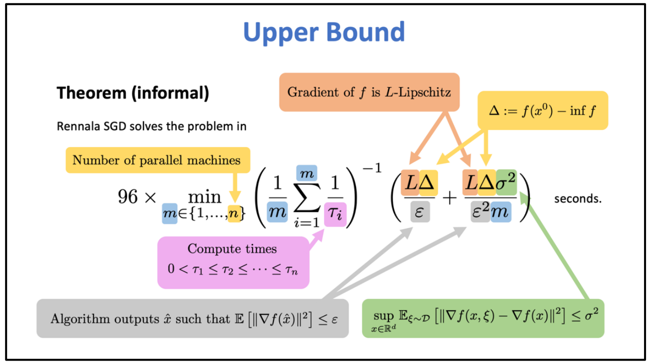
$\Delta := f(x^0) - \inf f$

Compute times
$0 < \tau_1 \leq \tau_2 \leq \cdots \leq \tau_n$

Algorithm outputs $\hat{x}$ such that $\mathbb{E}\left[\|\nabla f(\hat{x})\|^2\right] \leq \varepsilon$

$\sup_{x \in \mathbb{R}^d} \mathbb{E}_{\xi \sim \mathcal{D}}\left[\|\nabla f(x,\xi) - \nabla f(x)\|^2\right] \leq \sigma^2$

# Matching Lower Bound




## Theorem (informal)

It is not possible to design a method that will find a solution faster than in

$$\Omega\left(\min_{m \in \{1,\dots,n\}} \left(\frac{1}{m}\sum_{i=1}^{m}\frac{1}{\tau_i}\right)^{-1}\left(\frac{L\Delta}{\varepsilon} + \frac{L\Delta\sigma^2}{\varepsilon^2 m}\right)\right)$$

seconds.

**Rennala SGD = first optimal parallel SGD**

# Classical Oracle: Keeps Track of # Iterations

Function class

Distribution governing noise

Oracle class

Algorithm class

**Protocol 1** Classical Oracle Protocol

**Input:** function $f \in \mathcal{F}$ oracle and distribution $(O, \mathcal{D}) \in \mathcal{O}(f)$ algorithm $A \in \mathcal{A}$
**for** $k = 0, \ldots, \infty$ **do**
  $x^k = A^k(g^1, \ldots, g^k)$   $\triangleright$ $x^0 = A^0$ for $k = 0$.
  $g^{k+1} = O(x^k, \xi^{k+1})$,   $\xi^{k+1} \sim \mathcal{D}$
**end for**

Typically, stochastic gradient:
$g^{k+1} = \nabla f(x^k, \xi^{k+1})$

Natural for sequential methods, where a single worker does all the work!

**Iteration complexity** (classical complexity measure):

$$\mathfrak{m}_{\text{oracle}}(\mathcal{A}, \mathcal{F}) := \inf_{A \in \mathcal{A}} \sup_{f \in \mathcal{F}} \sup_{(O, \mathcal{D}) \in \mathcal{O}(f)} \inf \left\{ k \in \mathbb{N} \,\middle|\, \mathbb{E}\left[\|\nabla f(x^k)\|^2\right] \le \varepsilon \right\}$$

[Nemirovsky and Yudin, 1983] [Nesterov, 2018]
[Carmon et al, 2020] [Arjevani et al, 2022]

# New Oracle: Keeps Track of Time

**Protocol 2** Time Oracle Protocol

1: **Input:** functions $f \in \mathcal{F}$, oracle and distribution $(O, \mathcal{D}) \in \mathcal{O}(f)$, algorithm $A \in \mathcal{A}$
2: $s^0 = 0$
3: **for** $k = 0, \dots, \infty$ **do**
4: $(t^{k+1}, x^k) = A^k(g^1, \dots, g^k),$ $\triangleright\ t^{k+1} \geq t^k$
5: $(s^{k+1}, g^{k+1}) = O(t^{k+1}, x^k, s^k, \xi^{k+1}), \quad \xi^{k+1} \sim \mathcal{D}$
6: **end for**

Natural for parallel methods!

**Iteration complexity** (classical complexity measure):

$$\mathfrak{m}_{\text{oracle}}(\mathcal{A}, \mathcal{F}) := \inf_{A \in \mathcal{A}} \sup_{f \in \mathcal{F}} \sup_{(O,\mathcal{D}) \in \mathcal{O}(f)} \inf \left\{ k \in \mathbb{N} \,\middle|\, \mathbb{E}\left[\|\nabla f(x^k)\|^2\right] \leq \varepsilon \right\}$$

$$S_t := \left\{ k \in \mathbb{N} \cup \{0\} \,\middle|\, t^k \leq t \right\}$$

**Time complexity** (new complexity measure):

$$\mathfrak{m}_{\text{time}}(\mathcal{A}, \mathcal{F}) := \inf_{A \in \mathcal{A}} \sup_{f \in \mathcal{F}} \sup_{(O,\mathcal{D}) \in \mathcal{O}(f)} \inf \left\{ t \geq 0 \,\middle|\, \mathbb{E}\left[ \inf_{k \in S_t} \|\nabla f(x^k)\|^2 \right] \leq \varepsilon \right\}$$

# Data Homogeneous Regime

| Method | Time Complexity |
| --- | --- |
| Minibatch SGD | $\tau_n \left( \frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{n\varepsilon^2} \right)$ |
| Asynchronous SGD (Cohen et al., 2021) (Koloskova et al., 2022) (Mishchenko et al., 2022) | $\left( \frac{1}{n} \sum_{i=1}^{n} \frac{1}{\tau_i} \right)^{-1} \left( \frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{n\varepsilon^2} \right)$ |
| Rennala SGD (Theorem 7.5) | $\min_{m \in [n]} \left[ \left( \frac{1}{m} \sum_{i=1}^{m} \frac{1}{\tau_i} \right)^{-1} \left( \frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{m\varepsilon^2} \right) \right]$ |
| Lower Bound (Theorem 6.4) | $\min_{m \in [n]} \left[ \left( \frac{1}{m} \sum_{i=1}^{m} \frac{1}{\tau_i} \right)^{-1} \left( \frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{m\varepsilon^2} \right) \right]$ |

# Experimental Results (Sample)

$$\tau_i = \sqrt{i} \text{ seconds}$$

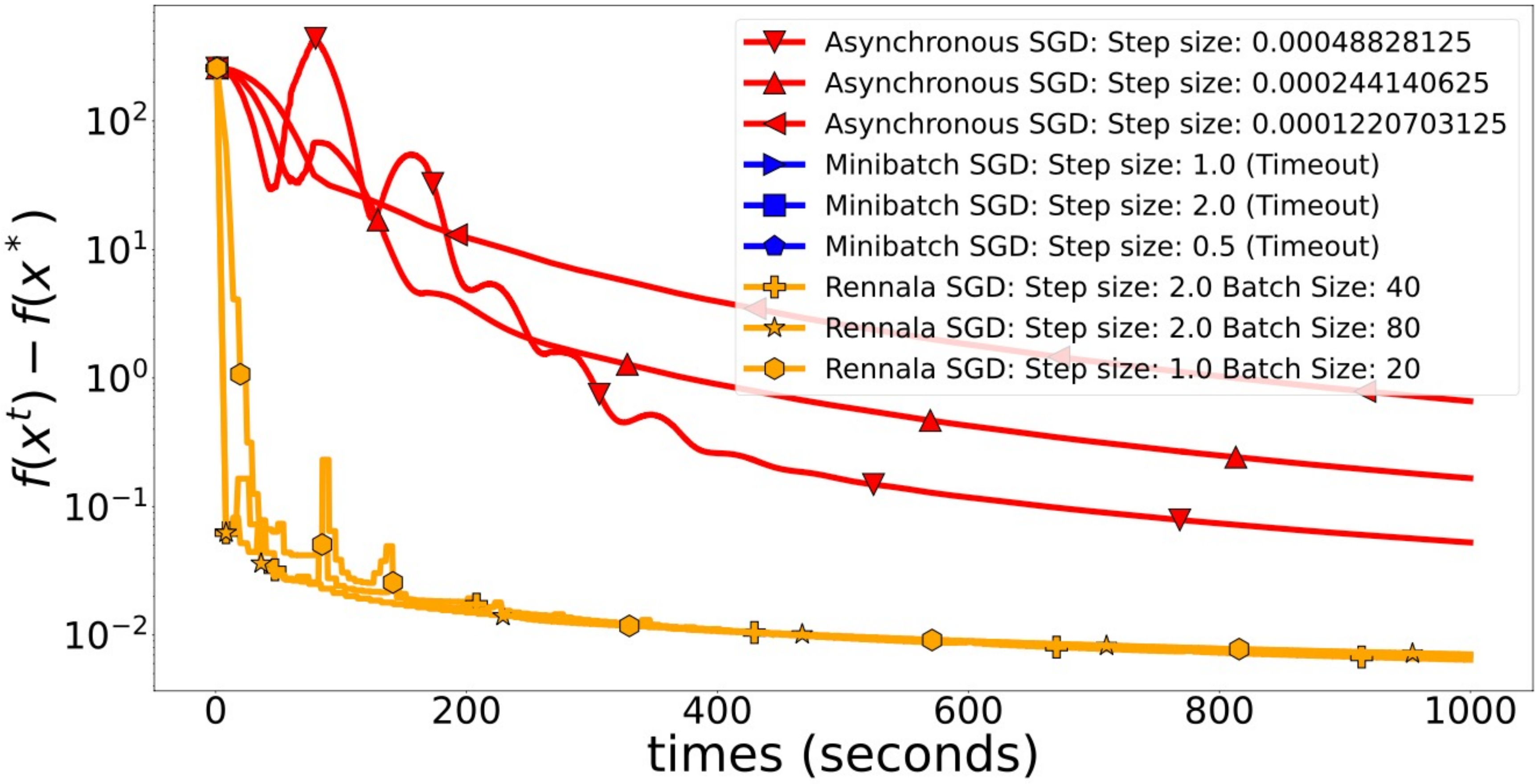



Figure 3: # of workers $n = 10000$.

# Part 4
# Two Extensions

# Extension 1: Data Heterogeneous Regime

| Method | Time Complexity |
|---|---|
| Minibatch SGD | $\tau_n \left( \frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{n\varepsilon^2} \right)$ |
| Malenia SGD (Theorem A.4) | $\tau_n \frac{L\Delta}{\varepsilon} + \left( \frac{1}{n} \sum_{i=1}^{n} \tau_i \right) \frac{\sigma^2 L\Delta}{n\varepsilon^2}$ |
| Lower Bound (Theorem A.2) | $\tau_n \frac{L\Delta}{\varepsilon} + \left( \frac{1}{n} \sum_{i=1}^{n} \tau_i \right) \frac{\sigma^2 L\Delta}{n\varepsilon^2}$ |

# Extension 2: Convex (Data Homogeneous) Regime

| Method | Time Complexity |
|---|---|
| Minibatch SGD | $\tau_n \left(\min\left\{\frac{\sqrt{L}R}{\sqrt{\varepsilon}}, \frac{M^2R^2}{\varepsilon^2}\right\} + \frac{\sigma^2R^2}{n\varepsilon^2}\right)$ |
| Asynchronous SGD (Mishchenko et al., 2022) | $\left(\frac{1}{n}\sum_{i=1}^{n}\frac{1}{\tau_i}\right)^{-1}\left(\frac{LR^2}{\varepsilon} + \frac{\sigma^2R^2}{n\varepsilon^2}\right)$ |
| (Accelerated) Rennala SGD (Theorems B.9 and B.11) | $\min_{m\in[n]}\left[\left(\frac{1}{m}\sum_{i=1}^{m}\frac{1}{\tau_i}\right)^{-1}\left(\min\left\{\frac{\sqrt{L}R}{\sqrt{\varepsilon}}, \frac{M^2R^2}{\varepsilon^2}\right\} + \frac{\sigma^2R^2}{m\varepsilon^2}\right)\right]$ |
| Lower Bound (Theorem B.4) | $\min_{m\in[n]}\left[\left(\frac{1}{m}\sum_{i=1}^{m}\frac{1}{\tau_i}\right)^{-1}\left(\min\left\{\frac{\sqrt{L}R}{\sqrt{\varepsilon}}, \frac{M^2R^2}{\varepsilon^2}\right\} + \frac{\sigma^2R^2}{m\varepsilon^2}\right)\right]$ |
| Lower Bound (Section M) (Woodworth et al., 2018) | $\tau_1 \min\left\{\frac{\sqrt{L}R}{\sqrt{\varepsilon}}, \frac{M^2R^2}{\varepsilon^2}\right\} + \left(\frac{1}{n}\sum_{i=1}^{n}\frac{1}{\tau_i}\right)^{-1}\frac{\sigma^2R^2}{n\varepsilon^2}$ |

$\nabla f$ is $L$-Lipschitz, $f$ is $M$-Lipschitz, and $\|x^0 - x^\star\| \leq R$

# The End

# Part 5
# Further Extensions

# Optimal Parallel Stochastic Gradient Methods

| | Data Heterogeneity ($\mathcal{D}_i$ different) | Compute Heterogeneity ($\tau_i$ different) | Communication Heterogeneity ($\theta_i$ different) | Smooth Nonconvex | Smooth Convex | Infinite / Finite Sum? | Supports Decentralized Setup? | Optimal Time Complexity? |
|---|---|---|---|---|---|---|---|---|
| **Rennala SGD**<br>Tyurin & R (NeurIPS '23) | ✗ | ✓ | 0 | ✓ | | **Inf** | ✗ | ✓ |
| **Malenia SGD**<br>Tyurin & R (NeurIPS '23) | ✓ | ✓ | 0 | ✓ | | **Inf** | ✗ | ✓ |
| **Accelerated Rennala SGD**<br>Tyurin & R (NeurIPS '23) | ✗ | ✓ | 0 | | ✓ | **Inf** | ✗ | ✓ |
| **Shadowheart SGD**<br>Tyurin, Pozzi, Ilin & R '24 | ✗ | ✓ | ✓ | ✓ | | **Inf** | ✗ | ✓ |
| **Freya PAGE**<br>Tyurin, Gruntkowska & R '24 | ✗ | ✓ | 0 | ✓ | | **Finite** | ✗ | ✓<br>big data regime |
| **Freya SGD**<br>Tyurin, Gruntkowska & R '24 | ✗ | ✓ | 0 | ✓ | | **Finite** | ✗ | ✗ |
| **Fragile SGD**<br>Tyurin & R '24 | ✗ | ✓ | ✓ | ✓ | | **Inf** | ✓ | nearly |
| **Amelie SGD**<br>Tyurin & R '24 | ✓ | ✓ | ✓ | ✓ | | **Inf** | ✓ | ✓ |

# Shadowheart SGD:

# Optimal Parallel SGD under Compute and Communication Heterogeneity

# Shadowheart SGD

$$x^{k+1} = x^k - \gamma \cdot \frac{\sum_{i=1}^{n} w_i \sum_{j=1}^{m_i} \mathcal{C}_{ij} \left( \sum_{l=1}^{b_i} \nabla f(x^k, \xi_{il}^k) \right)}{\sum_{i=1}^{n} w_i m_i b_i}$$

**Algorithm 1** Shadowheart SGD

**Input:** starting point $x^0 \in \mathbb{R}^d$, stepsize $\gamma > 0$, the ratio $\sigma^2/\varepsilon$, computation times $h_i > 0$, and communication times $\tau_i > 0$ for $i \in [n]$

Find the equilibrium time $t^*$ using Def. 4.2

Set $b_i = \left\lfloor \frac{t^*}{h_i} \right\rfloor$ and $m_i = \left\lfloor \frac{t^*}{\tau_i} \right\rfloor$ for all $i \in [n]$

Find active workers $S_{\text{A}} = \{i \in [n] \,:\, b_i \wedge m_i > 0\}$

**for** $k = 0, 1, \ldots, K-1$ **do**

  Run Alg. 2 in all active workers $S_{\text{A}}$

  Broadcast $x^k$, $b_i$, and $m_i$ to all active workers $S_{\text{A}}$

  Initialize $g^k = 0$

  **for** $i \in S_{\text{A}}$ **in parallel do**

    $w_i \overset{(a)}{=} \left(b_i\omega + \omega\frac{\sigma^2}{\varepsilon} + m_i\frac{\sigma^2}{\varepsilon}\right)^{-1}$

    **for** $j = 1, \ldots, m_i$ **do**

      Receive $\mathcal{C}_{ij}\left(g_i^k\right)$ from worker $i$

      $g^k = g^k + w_i \mathcal{C}_{ij}\left(g_i^k\right)$

    **end for**

  **end for**

  $g^k = g^k / \left(\sum_{i=1}^n w_i m_i b_i\right)$

  $x^{k+1} = x^k - \gamma g^k$

**end for**

$(a)$ : If $\omega = 0$ and $\frac{\sigma^2}{\varepsilon} = 0$, then $w_i = 1$

**Algorithm 2** Strategy of Worker $i$

Receive $x^k$, $b_i$, and $m_i$ from the server

Init $g_i^k = 0$

**for** $l = 1, \ldots, b_i$ **do**

  Calculate $\nabla f(x^k; \xi_{il}^k), \quad \xi_{il}^k \sim \mathcal{D}_\xi$

  $g_i^k = g_i^k + \nabla f(x^k; \xi_{il}^k)$

**end for**

**for** $j = 1, \ldots, m_i$ **do**

  Send $\mathcal{C}_{ij}\left(g_i^k\right) \equiv \mathcal{C}\left(g_i^k; \nu_{ij}^k\right)$ to the server, $\nu_{ij}^k \sim \mathcal{D}_\nu$, $\mathcal{C}_{ij} \in \mathbb{U}(\omega)$

**end for**

Table 1: **Time Complexities of Centralized Distributed Algorithms.** Assume that it takes at most $h_i$ seconds to worker $i$ to calculate a stochastic gradient and $\dot{\tau}_i$ seconds to send *one coordinate/float* to server. Abbreviations: $L$ = smoothness constant, $\varepsilon$ = error tolerance, $\Delta = f(x^0) - f^*$, $n$ = # of workers, $d$ = dimension of the problem. We take the Rand$K$ compressor with $K = 1$ (Def. C.1) (as an example) in QSGD and Shadowheart SGD. Due to Property 5.2, the choice $K = 1$ is optimal for Shadowheart SGD up to a constant factor.

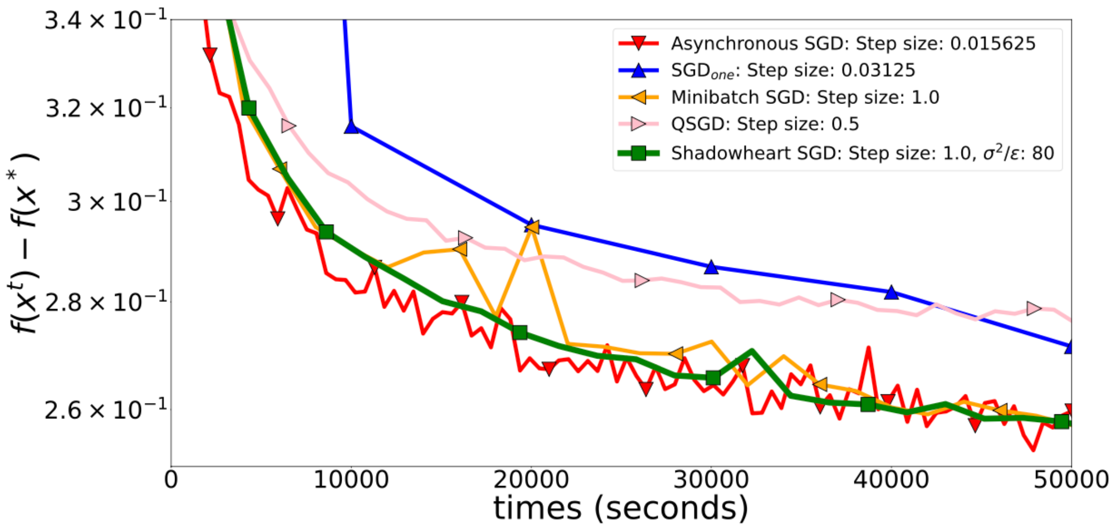
| Method | Time Complexity | $\max\{h_n, \dot{\tau}_n\} \to \infty$, $\max\{h_i, \dot{\tau}_i\} < \infty \, \forall i < n$ (the last worker is slow) | Time Complexities in Some Regimes: $h_i = h, \dot{\tau}_i = \dot{\tau} \, \forall i \in [n]$ (equal performance) | Numerical Comparison[b] $\sigma^2/\varepsilon = 1$ | $10^3$ | $10^6$ |
|---|---|---|---|---|---|---|
| Minibatch SGD (see (3)) | $\max_{i\in[n]} \max\{h_i, d\dot{\tau}_i\} \left(\frac{L\Delta}{\varepsilon} + \frac{\sigma^2 L\Delta}{n\varepsilon^2}\right)$ | $\infty$ (non-robust) | $\max\{h, d\dot{\tau}, \frac{d\dot{\tau}\sigma^2}{n\varepsilon}, \frac{h\sigma^2}{n\varepsilon}\} \frac{L\Delta}{\varepsilon}$ (worse, e.g., when $\dot{\tau}$, $d$ or $n$ large) | $\times 10^3$ | $\times 10^3$ | $\times 10^4$ |
| QSGD (see (7)) (Alistarh et al., 2017) (Khaled & Richtárik, 2020) | $\max_{i\in[n]} \max\{h_i, \dot{\tau}_i\} \left(\left(\frac{d}{n} + 1\right) \frac{L\Delta}{\varepsilon} + \frac{d\sigma^2 L\Delta}{n\varepsilon^2}\right)$ | $\infty$ (non-robust) | $\geq \frac{dh\sigma^2}{n\varepsilon} \frac{L\Delta}{\varepsilon}$ (worse, e.g., when $\varepsilon$ small) | $\times 3$ | $\times 10^2$ | $\times 10^4$ |
| Rennala SGD (Tyurin & Richtárik, 2023c), Asynchronous SGD (e.g., (Mishchenko et al., 2022)) | $\geq \min_{j\in[n]} \max\left\{h_{\bar{\pi}_j}, d\dot{\tau}_{\bar{\pi}_j}, \frac{\sigma^2}{\varepsilon}\left(\sum_{i=1}^{j} \frac{1}{h_{\bar{\pi}_i}}\right)^{-1}\right\} \frac{L\Delta}{\varepsilon}$ [a] | $< \infty$ (robust) | $\geq \max\left\{h, d\dot{\tau}, \frac{h\sigma^2}{n\varepsilon}\right\} \frac{L\Delta}{\varepsilon}$ (worse, e.g., when $\dot{\tau}$, $d$ or $n$ large) | $\times 10^2$ | $\times 10$ | $\times 1.5$ |
| Shadowheart SGD (see (9) and Alg. 1) (Corollary 4.4) | $t^*(d-1, \sigma^2/\varepsilon, [h_i, \dot{\tau}_i]_1^n) \frac{L\Delta}{\varepsilon}$ [c] | $< \infty$ (robust) | $\max\left\{h, \dot{\tau}, \frac{d\dot{\tau}}{n}, \sqrt{\frac{d\dot{\tau}h\sigma^2}{n\varepsilon}}, \frac{h\sigma^2}{n\varepsilon}\right\} \frac{L\Delta}{\varepsilon}$ | $\times 1$ | $\times 1$ | $\times 1$ |

The time complexity of Shadowheart SGD is not worse than the time complexity of the competing centralized methods (see Sec. 6), and is *strictly* better in many regimes.
We show that (12) is the *optimal time complexity* in the family of centralized methods with compression (see Sec. 7).

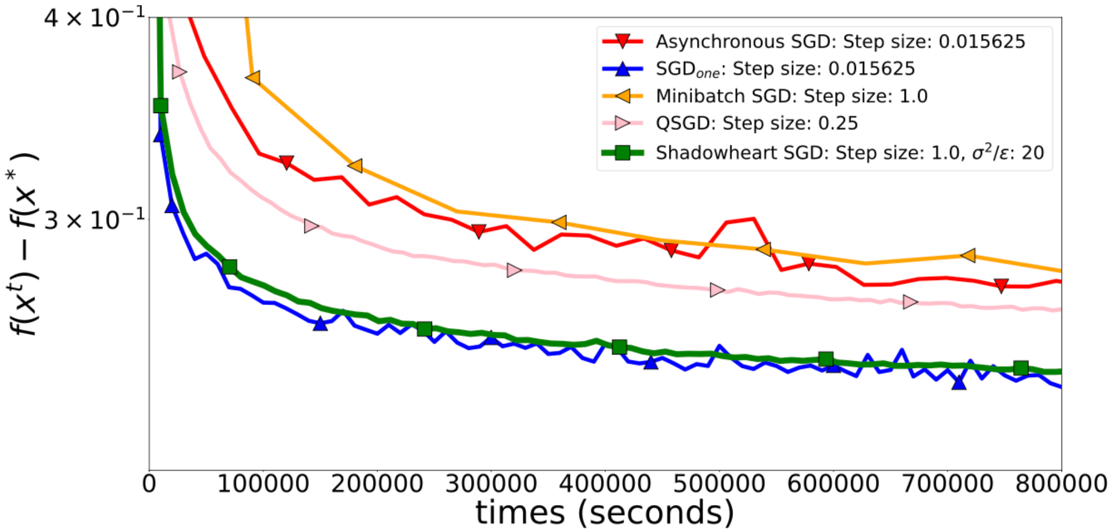
[a] Upper bound time complexities are not derived for Rennala SGD and Asynchronous SGD. However, we can derive the lower bound using Theorem N.5 with $\omega = 0$. One should take $d\dot{\tau}_i$ instead of $\tau_i$ when apply Theorem N.5 because these methods send $d$ coordinates. $\bar{\pi}$ is a permutation that sorts $\max\{h_i, d\dot{\tau}_i\} : \max\{h_{\bar{\pi}_1}, d\dot{\tau}_{\bar{\pi}_1}\} \leq \cdots \leq \max\{h_{\bar{\pi}_n}, d\dot{\tau}_{\bar{\pi}_n}\}$

[b] We numerically compute time complexities for $d = 10^6$, $n = 10^3$, $h_i \sim U(0.1, 1)$, $\dot{\tau}_i \sim U(0.1, 1)$ (uniform i.i.d.), and three noise regimes $\sigma^2/\varepsilon \in \{1, 10^3, 10^6\}$. We report the factors by which the time complexities of the competing methods are worse compared to the time complexity of our method Shadowheart SGD. So, for example, Minibatch SGD, QSGD and Asynchronous SGD can be worse by the factors $\times 10^4$, $\times 10^4$, and $\times 10^2$, respectively.

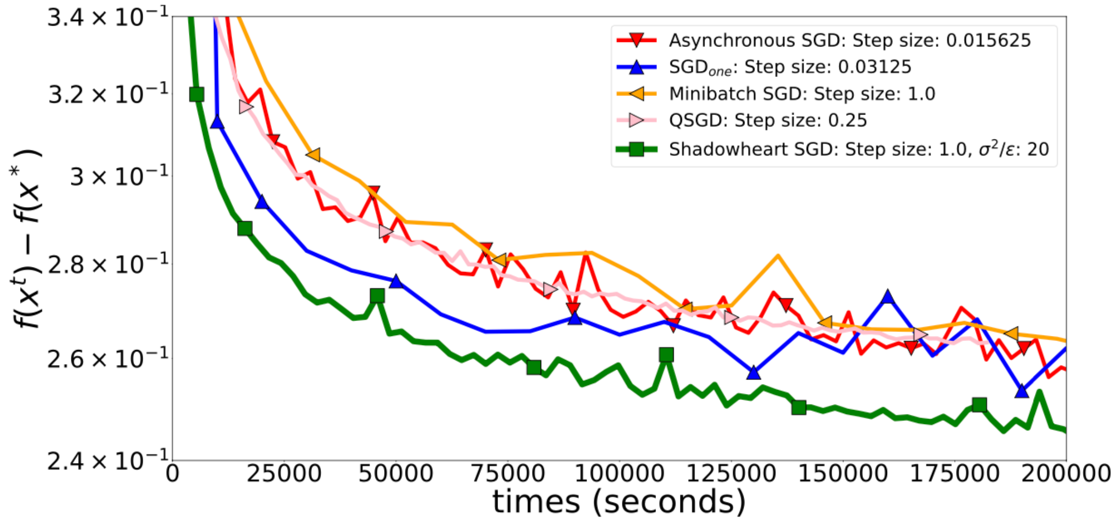
[c] The mapping $t^*$ is defined in Def. 4.2.



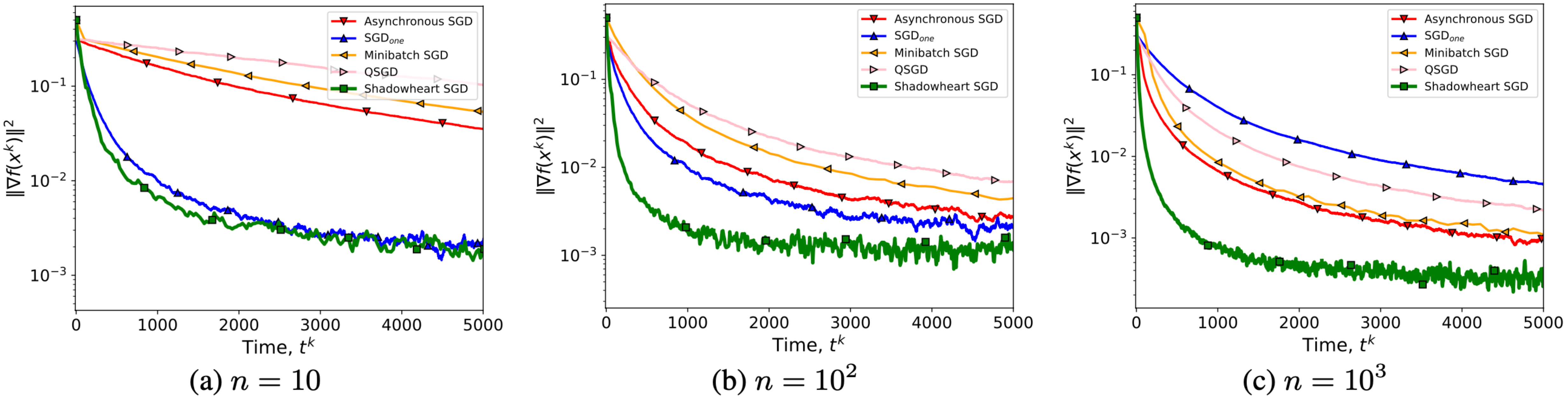

(a) Experiment with computation speeds $h_i = \sqrt{i}$ and **high** communications speeds $\dot{\tau}_i = \sqrt{i}/d$




(b) Experiment with computation speeds $h_i = \sqrt{i}$ and **low** communications speeds $\dot{\tau}_i = \sqrt{i}/d^{1/2}$




(c) Experiment with computation speeds $h_i = \sqrt{i}$ and **medium** communications speeds $\dot{\tau}_i = \sqrt{i}/d^{3/4}$



(a) $n = 10$ (b) $n = 10^2$ (c) $n = 10^3$

Figure 5: $h_i^k, \dot{\tau}_i^k \sim U(0.1, 1)$

# Amelie SGD:

# Optimal SGD under Computation and Communication Heterogeneity in the Decentralized Setup

# Decentralized Setup: Amelie SGD

| Method | The Worst-Case Time Complexity Guarantees | Comment |
|---|---|---|
| Minibatch SGD | $\frac{L\Delta}{\varepsilon} \max\left\{\left(1+\frac{\sigma^2}{n\varepsilon}\right)\max\{\max_{i,j\in[n]} \tau_{i\to j}, \max_{i\in[n]} h_i\}\right\}$ | suboptimal if $\sigma^2/\varepsilon$ is large |
| RelaySGD, Gradient Tracking (Vogels et al., 2021) (Liu et al., 2024) | $\geq \frac{\max_{i\in[n]} L_i\Delta}{\varepsilon} \frac{\sigma^2}{n\varepsilon} \max_{i\in[n]} h_i$ | requires local $L_i$-smooth. of $f_i$, suboptimal if $\sigma^2/\varepsilon$ is large (even if $\max_{i\in[n]} L_i = L$) |
| Asynchronous SGD (Even et al., 2024) | — | requires similarity of the functions $\{f_i\}$, requires local $L_i$-smooth. of $f_i$ |
| Amelie SGD and Lower Bound (Thm. 7 and Cor. 2) | $\frac{L\Delta}{\varepsilon} \max\left\{\max_{i,j\in[n]} \tau_{i\to j}, \max_{i\in[n]} h_i, \frac{\sigma^2}{n\varepsilon}\left(\frac{1}{n}\sum_{i=1}^{n} h_i\right)\right\}$ | Optimal up to a constant factor |

# The End
# (for real)