On Achieving Scalability through Relaxation

by

Giorgi Nadiradze

December, 2021

A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

I S T AUSTRIA
Institute of Science and Technology

The thesis of Giorgi Nadiradze, titled *On Achieving Scalability through Relaxation*, is approved by:

Supervisor: Dan Alistarh, IST Austria, Klosterneuburg, Austria

Signature: ______________________________

Committee Member: Krishnendu Chatterjee, IST Austria, Klosterneuburg, Austria

Signature: ______________________________

Committee Member: Thomas Sauerwald, University of Cambridge, Cambridge, UK

Signature: ______________________________

Committee Member: Christopher De Sa, Cornell University, Ithaca, USA

Signature: ______________________________

Defense Chair: Marco Mondelli, IST Austria, Klosterneuburg, Austria

Signature: ______________________________

Signed page is on file

IST Austria Thesis, ISSN: 2663-337X

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: ______________________________

Giorgi Nadiradze
December, 2021

Signed page is on file

Abstract

The scalability of concurrent data structures and distributed algorithms strongly depends on reducing the contention for shared resources and the costs of synchronization and communication. We show how such cost reductions can be attained by relaxing the strict consistency conditions required by sequential implementations.

In the first part of the thesis, we consider relaxation in the context of concurrent data structures. Specifically, in data structures such as priority queues, imposing strong semantics renders scalability impossible, since a correct implementation of the remove operation should return only the element with highest priority. Intuitively, attempting to invoke remove operations concurrently creates a race condition. This bottleneck can be circumvented by relaxing semantics of the affected data structure, thus allowing removal of the elements which are no longer required to have the highest priority. We prove that the randomized implementations of relaxed data structures provide provable guarantees on the priority of the removed elements even under concurrency. Additionally, we show that in some cases the relaxed data structures can be used to scale the classical algorithms which are usually implemented with the exact ones.

In the second part, we study parallel variants of the stochastic gradient descent (SGD) algorithm, which distribute computation among the multiple processors, thus reducing the running time. Unfortunately, in order for standard parallel SGD to succeed, each processor has to maintain a local copy of the necessary model parameter, which is identical to the local copies of other processors; the overheads from this perfect consistency in terms of communication and synchronization can negate the speedup gained by distributing the computation. We show that the consistency conditions required by SGD can be relaxed, allowing the algorithm to be more flexible in terms of tolerating quantized communication, asynchrony, or even crash faults, while its convergence remains asymptotically the same.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dan Alistarh, for always finding time for me, and for always steering me in the right direction, whenever I felt stuck on a project. He accepted me as his PhD student, even though I did not have background in Distributed Computing, and was very helpful, supportive and patient as I was learning new concepts.

I am grateful to Krishnendu Chatterjee, Thomas Sauerwald and Christopher De Sa for agreeing to be on my thesis committee, and to Marco Mondelli for chairing the defense.

Many thanks to George Giakkoupis for hosting me in Rennes, and for giving very useful feedback, on one of my papers, during the visit.

I was lucky to be affiliated with two different research groups during my PhD studies. I would like to thank Peter Widmayer for welcoming me into his group at ETH, and all members of the group for making my stay at ETH enjoyable. Similarly, I am grateful to all my colleagues from the group Alistarh at IST.

I would also like to thank the lunch “team” of the groups Alistarh and Mondelli for their company during the pandemic.

I gratefully acknowledge funding from Swiss National Fund Ambizione Project PZ00P2 161375, and European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 805223 ScaleML).

Last but not least, I would like to thank my family and friends for their encouragement and support.

About the Author

Giorgi Nadiradze completed a BSc in Informatics at Tbilisi State University and a MSc in Applied Mathematics at Central European University. In September 2016, he started his PhD studies at ETH Zurich, under the supervision of Dan Alistarh. In October 2018, he joined the Alistarh Group at IST Austria. His research mainly focuses on concurrent data structures and distributed optimization, besides that, he has also worked on various topics, such as load balancing on graphs, shared-memory leader election and the strip packing problem.

List of Collaborators and Publications

The list of all publications which appear in this thesis:

1. Dan Alistarh, Trevor Brown, Justin Kopinsky, Jerry Z. Li, and Giorgi Nadiradze. Distributionally linearizable data structures. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, SPAA '18, pages 133–142, New York, NY, USA, 2018. ACM

2. Dan Alistarh, Giorgi Nadiradze, and Nikita Koval. Efficiency guarantees for parallel incremental algorithms under relaxed schedulers. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 145–154. ACM, 2019

3. Giorgi Nadiradze, Ilia Markov, Bapi Chatterjee, Vyacheslav Kungurtsev, and Dan Alistarh. Elastic consistency: A practical consistency model for distributed stochastic gradient descent. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9037–9045, May 2021

4. Giorgi Nadiradze, Amirmojtaba Sabour, Peter Davies, Shigang Li, and Dan Alistarh. Asynchronous decentralized SGD with quantized and local updates. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021

Table of Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

The performance thresholds reached by single-processor computing, together with advances in parallel hardware, have brought up the need to design scalable algorithms and data structures, which allow computation to be distributed *efficiently* among multiple nodes. Scalability can be hindered by resource contention, as well as synchronization and communication costs. Some data structures, such as queues and stacks, have inherently strict semantics, which induce race conditions when concurrent threads attempt to retrieve the element with highest priority (the element which was inserted last in stack and the element which was inserted first in queue), without breaking correctness. This creates necessity to relax the strict semantics and allow threads to remove an element which does not necessarily have the highest priority. Reducing contention at the expense of relaxing semantics gives rise to two important questions, which this thesis aims to answer. First, can we assess the quality of the removed element by providing provable rank guarantees? Second, is it possible to apply relaxed data structures to the problems which are known to be solvable using their exact counterparts?

Further, some algorithms, for example mini-batch stochastic gradient descent are naturally parallel. More precisely, mini-batch stochastic gradient is an iterative algorithm, where at each iteration, parameter is updated using the average of n stochastic gradients. In the parallel version, computation is distributed among n processors: each processor maintains local copy of the parameter and computes stochastic gradient using the local copy. In order to achieve the same convergence rate as the sequential version, processors need to keep their local parameters consistent: at each iteration processors need to receive all the stochastic gradients computed by the other processors, then they apply average of all the stochastic gradients to their local parameter and proceed to the next iteration. Maintaining the local parameters consistent requires all to all communication and synchronization among the processors, which in practice is costly and becomes a bottleneck which limits scalability. We show that this issue can be circumvented by relaxing parameter consistency. Specifically, we show that under asynchrony and reduced communication, even though the local parameters are no longer consistent, they do not have a large deviation, and this allows us to prove that the convergence is not affected.

This thesis consists of four chapters: the first two contain results about concurrent data structures with relaxed semantics, and third and fourth address the distributed optimization (distributed stochastic gradient descent in particular) under the relaxed consistency setting. We go over the results in more detail below.

1.1 Concurrent Data Structures

In Chapter 2 we consider relaxed concurrent data structures. As an illustration, let us consider the following implementation of a relaxed counter algorithm. Recall that an exact counter is a data structure which supports the following two methods: $Increment()$, which, as the name suggests, increments the value of counter, and $Read()$, which returns the current value of the counter. In a sequential setting, it is easy to see that in order for the counter implementation to be correct, every $Read()$ operation should return a value which is equal to the number of $Increment()$ operations which were invoked before the $Read()$. The classic correctness measure used for the concurrent implementations of the counter is linearizability [HW90]: that is, each method should atomically happen at some point between its invocation and response. The point at which the method is executed is called a linearization point, and every $Read()$ operation should return the value which is equal to the number of $Increment()$ operations whose linearization points precede the linearization point of the $Read()$. The naive implementation which uses single sequential counter with synchronization primitive is not scalable at higher thread count, because of the contention for the counter, and this can even be proved theoretically [AACH$^+$14, EHS12].

Instead, we consider a relaxed version of the counter, which uses n sequential counters, instead of a single one. The $Increment()$ operation picks two counters uniformly at random, atomically reads their values and atomically increments the one with the smaller value. The $Read()$ operation chooses one counter uniformly at random and returns its value multiplied by n. The linearization point of $Increment()$ operation corresponds to the atomic write(increment) it performs, and the linearization point of $Read()$ operation corresponds to its atomic read. Since the relaxed counter is randomized and no longer returns the exact value of the counter we need to introduce the new correctness condition - Distributional Linearizability. Distributional Linearizability can be seen as the version of classical linearizability, but with additional probabilistic costs which are caused by relaxation (The exact versions of data structures have cost zero). In the case of relaxed counter, if the values of counters are $x_1, x_2, ...x_n$ at the point $Read()$ does atomic read (linearization point), then if $Read()$ chooses counter i then the cost is $n\Big|x_i - \frac{\sum_{j=1}^n x_j}{n}\Big|$. Notice that in the absence of concurrency the relaxed counter is the analogue of the classical balls into bins process with two choices [ABKU99]. Hence for any i, the expected cost $n\,\mathbb{E}\Big|x_i - \frac{\sum_{j=1}^n x_j}{n}\Big|$ can be upper bounded by $O(n\log n)$ and the cost is $O(n\log n)$ with high probability as well [PTW15, BCSV00]. The main challenge introduced by the concurrency is that the values of the counters are no longer consistent: at the point when $Increment()$ operation performs atomic write, we are no longer guaranteed that it increments the counter with the smaller value. We are first to show that in this setting the cost can be upper bounded by $O(n\log^2 n)$, both in expectation and with high probability. We also provide empirical results which show that the relaxed counter does indeed scale well. Next, we consider relaxed queues [RSD15] and show that the expected cost is upper bounded by $O(n\log^2 n)$ as well. Here, the cost is defined in terms of the rank of returned element, which is equal to one plus the number of elements which were enqeueud before it. The result can be extended to the relaxed priority queues as long as the elements are inserted in decreasing priority order. In the proof we use techniques derived in [AKLN17]; an important and non-trivial distinction is that we have to deal with concurrency between operations.

In Chapter 3 we show that in some algorithms relaxed data structures can be efficiently used instead of exact ones. We are given a relaxed priority scheduler which contains tasks with priorities. The rank of the task is one plus the number of tasks with higher priority, which are currently in the scheduler. For a task u, let $inv(u)$ be the number of tasks with higher rank (lower priority) which are returned by the scheduler during the period when u is the highest priority task and let $rank(t)$ be the rank of the returned task at time step t (here, timestep corresponds to the number of tasks returned by the scheduler).

We require that the relaxed scheduler has the following properties. For any task u, $inv(u) < k$ and for any $t > 0$, $rank(t) \le k$. The results from Chapter 2 can be used to show that MultiQueue [RSD15] satisfies the above properties with high probability. Algorithms such as single source shortest path algorithm [Dij59] can be implemented using the exact scheduler. For example, in the case of [Dij59], each node corresponds to task and the rank of a task is a current distance from the source node to the node it corresponds to. While the scheduler is not empty, the algorithm gets the task with the highest priority (shortest distance) from the scheduler and executes it by relaxing the outgoing edges of the retrieved node. The parallel implementations of Dijkstra's algorithm have limited scalability since we again have the unavoidable bottleneck caused by high contention on the node with the shortest distance.

Hence the need to use the relaxed scheduler. Notice that it is not clear how the running time of the algorithm is affected by priority inversions caused by relaxation. That is, the nodes returned by the scheduler are no longer at the optimal distance and the algorithm is forced to process them multiple times (we count such occurrences towards extra/wasted work performed by the algorithm). We show that the number of times the nodes are processed in total is at most $n + O(\text{poly}(k)\frac{d_{max}}{w_{min}})$, where n is the number of the nodes in the graph, d_{max} is the maximum shortest distance from the source node to some other node and w_{min} is the smallest edge weight. Here, $O(\text{poly}(k)\frac{d_{max}}{w_{min}})$ is the waste or overhead caused by the relaxation, since the exact version processes each node exactly once.

The argument can be used to roughly count the running time of the parallel version. Assume that $k = O(\text{poly}(\log n))$, $\frac{d_{max}}{w_{min}} = O(\text{poly}(\log n))$ and the largest degree is $O(\text{poly}(\log n))$ as well, then the running time of sequential Dijkstra's algorithm is $O(n\,\text{poly}(\log n))$ while the total work performed by the parallel version when using the relaxed scheduler is $O(n \log^2 n) + O(\text{poly}(\log n))$ ($O(n\log^2 n)$ is the actual work and $O(poly(\log n))$ is the wasted work). If the algorithm uses $\log n$ threads and assuming (optimistically) that the actual work is distributed evenly among the threads, we get that the running time of the parallel algorithm is $O(n \log n) + O(\text{poly}(\log n))$, giving us $\log n$ factor speedup compared to the sequential version. We experimentally show that the SSSP algorithm which uses MultiQueues [RSD15] is indeed scalable. Additionally, experimental results suggest that for the graphs with low diameter and low variance of the edge weights the overhead from relaxation is even less than what the upper bound provided by the theoretical results implies. Finally, in Chapter 3 we derive upper and lower bounds on the extra work caused by the relaxation for the incremental algorithms such as Delaunay Triangulation and Comparison Sorting.

1.2 Distributed Optimization

Consider the following optimization problem. We wish to minimize function $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is a compact subset of $\mathbb{R}^d$. For the simplicity we will assume that $\mathcal{X}$ is actually $\mathbb{R}^d$ and f is non-convex, even though we will show how to deal with the convex case in Chapter 4. In the context of *supervised learning*, we are given m data samples $\{S_1, S_2, ..., S_m\}$ and

$$f(x) = \sum_{j=1}^{m} \frac{\ell_j(x)}{m},$$

where ℓ_j is a loss function at data sample S_j. The classical stochastic gradient descent is an iterative method which starts at some initial point $x_0 \in \mathbb{R}^d$ and the iteration is given by

$$x_{t+1} = x_t - \eta \tilde{G}(x_t).$$

Here, $\tilde{G}(x_t)$ is a stochastic gradient which is gradient of the loss function at a randomly chosen data sample ($\tilde{G}(x_t)$ is a gradient of ℓ_j at x_t, where $1 \leq j \leq m$ is chosen uniformly at random) and $0 < \eta < 1$ is a learning rate used to control the convergence of SGD. Given that function f is L-smooth and the data samples have variance σ^2 (for any $x \in \mathbb{R}^d$, $\mathbb{E}\,\|\tilde{G}(x) - \nabla f(x)\|^2 \leq \sigma^2$), it is very well known (e.g [GL13]) that after T iterations of SGD:

$$\sum_{t=0}^{T-1} \frac{\nabla f(x_t)}{T} = O\Big(\frac{f(x_0) - f(x^*)}{\eta T} + \sigma^2 \eta\Big).$$

In the above, O hides parameters such as L and $f(x^*)$ is a lower bound on a function f. By setting $\eta = \frac{1}{\sqrt{T}}$ we get the convergence rate of $O\Big(\frac{f(x_0)-f(x^*)}{\sqrt{T}} + \frac{\sigma^2}{\sqrt{T}}\Big)$. For the parallel version we consider two types of systems: message-passing which is covered in Chapter 4 and shared-memory which we cover in Chapters 4 and 5.

1.2.1 Message Passing

Given n processors which communicate with each other using message passing, one round (step) of the parallel version of SGD can be written as

$$x_{t+1} = x_t - \sum_{i=1}^{n} \frac{\eta}{n} \tilde{G}^i(x_t).$$

where each node i computes stochastic gradient $\tilde{G}^i(x_t)$ independently (we assume that each node has access to the entire data) and then sends the result to all other nodes, in order to maintain the value of x_t consistent. Since $\sum_{i=1}^{n} \frac{\tilde{G}^i(x_t)}{n}$ is also stochastic gradient, but with the reduced variance $\frac{\sigma^2}{n}$, after T rounds (steps) we have that

$$\sum_{t=0}^{T-1} \frac{\nabla f(x_t)}{T} = O\Big(\frac{f(x_0) - f(x^*)}{\eta T} + \frac{\sigma^2 \eta}{n}\Big).$$

Which allows us to set $\eta = \frac{\sqrt{n}}{\sqrt{T}}$, to get the convergence rate of $O\Big(\frac{f(x_0)-f(x^*)}{\sqrt{nT}} + \frac{\sigma^2}{\sqrt{nT}}\Big)$. Even though this implies that the parallel version is theoretically scalable (with a speedup factor of $\Omega(\sqrt{n})$), the algorithm itself is not robust since it needs all to all communication and synchronization, which are required to make sure that all processors receive all the stochastic gradients computed by the other processors. For the algorithm to scale in practice, we need to take care of the following issues:

- For very large d, processors need to compress the stochastic gradients, in order to reduce the communication costs.
- Some processors can be slow at computing stochastic gradients, sent stochastic gradients can be delayed because of the network issues, or processors can just crash before sending their stochastic gradient to all the other processors. Thus, synchonization becomes costly or even impossible, and we need to reduce synchronization overhead or make the algorithm entirely asynchronous.

This means that processors are not able to keep x_t consistent. We relax the consistency condition by assuming that each processor i has a local view of x_t, which we denote by v_t^i.

In Chapter 4 we prove that the relaxed version has asymptotically the same convergence rate (Hence, it is scalable) as the exact one as long as the elastic consistency criterion is satisfied. That is, there exists a constant B, such that for each processor i and step t:

$$\mathbb{E}\,\|v_t^i - x_t\|^2 \leq \eta^2 B^2.$$

More precisely, the convergence is affected by the fact that stochastic gradients are computed over local views (models) and not over x_t. But, using Cauchy-Schwarz inequality, L-smoothness, variance bound and elastic consistency we can show that for any processor i and time step t

$$\mathbb{E}\,\|\tilde{G}^i(v_t^i) - \tilde{G}^i(x_t)\|^2 \leq 3L^2\eta^2B^2 + 6\sigma^2.$$

This allows us to prove that in this case convergence rate is $O\Big(\frac{f(x_0)-f(x^*)}{\sqrt{nT}} + \frac{\sigma^2}{\sqrt{nT}} + \frac{B^2L^2n}{T}\Big)$. Note that for large enough T, the third term is dominated by the first two terms, hence if we run the relaxed SGD for long enough we are able to asymptotically match the converge rate of the exact version.

The crucial point is that our analysis encapsulates all the settings which caused us to relax consistency. That is, it can be cleanly split into two parts: First part proves that elastic consistency condition is satisfied for the concrete setting we consider, and the second part provides unified convergence analysis for all the settings.

1.2.2 Shared Memory

The scalability in message passing systems was achieved through the variance reduction. This in turn was accomplished by grouping local SGD iterations together to create one parallel round (In chapter 4 we show that this can be done even when the algorithm is asynchronous). Next, we consider shared memory setting, which is mainly used for asynchronous algorithms. That is, processors access shared memory asynchronously and we do not have a notion of parallel rounds.

We start by describing the algorithm studied in [ADSK18]. The shared parameter vector $x \in \mathcal{R}^d$ is stored in a shared memory, so that processors can atomically read each of its coordinates and are able to atomically add value to coordinates as well. Note that by limiting atomic addition to the coordinates only, we are able to avoid the bottleneck which would be caused by multiple processors trying to update the shared model. This can be seen as the analogue of multiple threads attempting to remove the highest priority element from the concurrent data structure. In the case of concurrent data structures, we reduced contention by allowing removal of the elements which are no longer required to have the highest priority. Here, since the dimension of the shared parameter is usually large, we reduce contention by distributing update operations among the coordinates. Subsequently, the views of the shared parameter read by processors are inconsistent (This would be the case even if they were able to read the entire parameter atomically). First we define the notion of time step: one time step corresponds to one atomic update of the first coordinate of the shared parameter. Let i be a processor which performs $t+1$-th atomic update of the first coordinate. One round of SGD can be written as:

$$x_{t+1} = x_t - \eta\tilde{G}(v_t^i).$$

In the above, v_t^i is the inconsistent view read by processor i, which it used to compute stochastic gradient. We assume that processors have access to the entire data, thus we do not use notation for the local stochastic gradients ($\tilde{G}^i$ for the processor i). In Chapter 4 we show that elastic consistency condition is satisfied in this case as well. That is, for each step t and processor i which performs SGD iteration at step $t+1$, there exists parameter $B > 0$

such that:

$$\mathbb{E}\,\|x_t - v_t^i\|^2 \leq \eta^2 B^2.$$

Here, B depends on d (the dimension of the shared parameter), interval contention (maximum number of concurrent operations observed by the algorithm) and the second moment bound. The second moment bound M is a parameter such that for any vector $x \in R^d$, $\mathbb{E}\,\|\tilde{G}(x)\| \leq M^2$. Also, in Chapter 4 we show that the convergence analysis from the message passing systems can also be used in this case, to obtain the convergence rate $O\Big(\frac{f(x_0)-f(x^*)}{\sqrt{T}} + \frac{\sigma^2}{\sqrt{T}} + \frac{B^2L^2}{T}\Big)$. Note that we no longer have speedup, since we have only one SGD iteration per round (step). Instead, we can argue that the speedup comes from the observation that even though we consider single SGD iteration per round, in practice the linear number of iterations can happen concurrently. Hence, Tn sequential rounds which contain single SGD iteration, can be seen as $O(T)$ parallel rounds which contain the linear number of SGD iterations. Thus, if T is the number of parallel rounds, then the convergence rate becomes $O\Big(\frac{f(x_0)-f(x^*)}{\sqrt{nT}} + \frac{\sigma^2}{\sqrt{nT}} + \frac{B^2L^2}{nT}\Big)$. This means that we again achieve $\Omega(\sqrt{n})$ speedup, given that B is negligible and T is large enough.

In Chapter 5 we relax the consistency even further. We consider scenario where processors are nodes of a graph, each node is only able to communicate with the neighbouring nodes and the data is split among the nodes.

We devise an GD algorithm which is fully asynchronous and uses compressed communication. Each node maintains its own local model, which is stored in the corresponding local register. Nodes communicate by updating local registers of each other. Recall that in the previous case nodes shared one global model, and contention was reduced by having per coordinate atomic update operations. Here, we require the entire model to be updated atomically, but we would like to point out that contention is reduced by using one local model per node and limiting interaction partners of nodes to a randomly chosen neighbours. At node i, our algorithm can roughly be described as follows (for the simplicity we omit the full description). Node i computes the random number of local stochastic gradients using the data samples available to it, then it communicates with the random neighbour and retrieves its approximate local model by reading from the local register (approximation is caused by asynchrony and quantized communication). Next, it averages the approximate local model of the neighbour with its own local model to get the vector v^{avg}. Then it writes quantized value of v^{avg} in the local register of the neighbour. Finally, it updates v^{avg} using the local stochastic gradients it computed beforehand and writes the quantized version of the updated vector in its own local register. The time steps in this model correspond to the communication among the nodes: at each time step some node finishes computation of the local stochastic gradients and interacts with a random neighbour. In the theory of gossip type algorithms it is common to assume that nodes communicate with neighbouring nodes once a Poisson clock with rate 1 ticks. We make the same assumption and it implies that, at each step, a node which contacts its neighbour is chosen uniformly at random [GNW16]. Next, we provide the intuition behind the scalability of our algorithm.

Unlike the settings in Chapter 4, the elastic consistency condition does not hold, since nodes use the local models of adjacent nodes in order to update theirs. Instead, we exploit graph topology and load balancing tools to show that for any step t there exists a parameter B such that:

$$\sum_{i=1}^{n} \mathbb{E}\,\|v_t^i - \mu_t\|^2 \leq n\eta^2 B^2,$$

where v_t^i is a local model of node i after t steps and $\mu_t = \frac{\sum_{i=1}^n v_t^i}{n}$. This can be seen as the elastic consistency condition which holds for the node which is chosen uniformly at random. But, since we have shown that at each step the node which performs SGD iteration is chosen uniformly at random as well, we can follow the convergence proof used in Chapter 4 for asynchronous shared memory setting, and also bound the error caused by additional relaxations to prove that our SGD algorithm achieves $\Omega(\sqrt{n})$ speedup.

CHAPTER 2

Distributionally Linearizable Data Structures

2.1 Introduction

Consider a system of n threads, which share a set of n distinct atomic counters. We wish to implement a *scalable approximate (relaxed) counter*, which we will call a *MultiCounter*, by distributing the contention among these n distinct instances: to *increment* the global counter, a thread selects two atomic counters i and j uniformly at random, reads their values, and (atomically) increments by 1 the value of the one which has *lower value* according to the values it read. To *read* the global counter, the thread returns the value of a randomly chosen counter i, multiplied by n. [1] Notice that the *read* operation is not guaranteed to return the correct value of the global counter, which is equal to the total number of increments. Throughout this chapter, we will refer to the absolute value of the difference between the returned value and the correct value as the *skew* (of the relaxed counter).

The astute reader will have noticed that this process is similar to the classic two-choice load balancing process [ABKU99], in which a sequence of balls are placed into n initially empty bins, and, in each step, a new ball is placed into the less loaded of two randomly chosen bins. Here, the individual atomic counters are the *bins*, and each increment corresponds to a new *ball* being added. This sequential load balancing process is extremely well studied [RMS01, Mit96]: a series of deep technical results established that the difference between the most loaded bin and the average is $O(\log \log n)$ both in expectation and with high probability [ABKU99, Mit96], and that this difference remains stable as the process executes for increasingly many steps [BCSV00, PTW15]. In [PTW15], it is shown that the similar result holds for the difference between the average and least loaded bin, albeit in this case, the difference is upper bounded by $O(\log n)$. Subsequently, in the absence of concurrency, the *skew* of the relaxed counter can be bounded by $O(\log n)$. We would therefore expect the above relaxed concurrent counter to have relatively low and stable *skew*, and to scale well, as contention is distributed among the n counters.

However, there are several technical issues when attempting to analyze this natural process in a concurrent setting.

[1] This multiplication serves to maintain the same magnitude as the total number of updates to the distributed counter up to a point in time.

- First, concurrency interacts with classic two-choice load balancing process in non-trivial ways. The key property of the two-choice process which ensures good load balancing is that trials are *biased towards less loaded bins*—equivalently, operations are biased towards incrementing counters of lesser value. However, this property may break due to concurrency: *at the time of the update*, a thread may end up updating the counter of *higher* value among its two choices if the counter of smaller value is updated concurrently since it was read by the thread, thus surpassing the other counter.
- Second, perhaps suprisingly, it is currently unclear how to even *specify* such a concurrent data structure. Despite a significant amount of work on specifying *deterministic relaxed* data structures [HKP$^+$13, AKY10, HHH$^+$16] , none of the existing frameworks cover relaxed *randomized* data structures.
- Finally, assuming such a data structure can be analyzed and specified, it is not clear whether it would be in any way *useful*: many existing applications are built around data structures with deterministic guarantees, and it is not obvious how scalable, relaxed data structures can be leveraged in standard concurrent settings.

One may find it surprising that analysing such a relatively simple concurrent process is so challenging. Beyond this specific instance, these difficulties reflect wider issues in this area: although these constructs are reasonably popular in practice due to their good scalability, e.g. [BFK$^+$11, NLP13, WGTT15, RSD15], their properties are non-trivial to pin down [AKLN17], and it is as of yet unclear how they interact with the higher-order algorithmic applications they are part of [LNP15].

Contribution. In this chapter, we take a step towards addressing these challenges. Specifically:

- We provide the first analysis of a two-choice load balancing process in an asynchronous setting, where operations may be interleaved, and the interleaving is decided by an adversary. We show that the resulting process is robust to concurrency, and continues to provide strong balancing guarantees in potentially infinite executions, as long as the ratio between the number of bins and the number of threads is above a large constant threshold.
- We introduce a new correctness condition for *randomized relaxed* data structures, called *distributional linearizability*. Intuitively, a concurrent data structure D is distributionally linearizable to a *sequential random process* R, defined in terms of a sequential specification S, a cost function $cost$ measuring the deviation from the sequential specification, and a distribution $\mathcal{P}$ on the values of the cost function, if every execution of D can be mapped onto an execution of the relaxed sequential process R, respecting the outputs and the costs incurred, as well as the order of non-overlapping operations.
- We prove that the randomized *MultiCounter* data structure introduced above is distributionally linearizable to a (sequential) variant of the classic two-choice load balancing process. This allows us to formally define the properties of MultiCounters. Moreover, we show that this analytic framework also covers variants of *MultiQueues* [RSD15], a popular family of concurrent data structures implementing relaxed concurrent priority queues. This yields the first analytical guarantees for MultiQueues in concurrent executions.
- We implement the MultiCounters, and show that they can provide a highly scalable approximate timestamping mechanism, with relatively low *skew*. We build on this, and

show that MultiCounters can be successfully applied to timestamp-based concurrency control mechanisms such as the TL2 software transactional memory protocol [DSS06]. This usage scenario presents an unexpected trade-off: assuming low contention, the resulting TM protocol scales almost linearly, but may break correctness with very low probability. In particular, we show that there exist workloads and parameter settings for which this relaxed TM protocol scales almost linearly, improving the performance of the TL2 baseline by more than $3\times$, without breaking correctness.

Techniques. Our main technical contribution is the concurrent analysis of the classic two-choice load balancing process, in an asynchronous setting, where the interleaving of low-level steps is decided by an oblivious adversary. The core of our analysis builds on the elegant potential method of Peres, Talwar and Wieder [PTW15], which we render robust to asynchronous updates based on potentially stale information. To achieve this, we overcome two key technical challenges. The first is that, given an operation op, as more and more other operations execute between the point where it reads and the point where it updates, the more stale its information becomes, and so the probability that op makes the "right" choice at the time of update, inserting into the less loaded of its two random choices, *decreases*. Moreover, operations updating with stale information will "stampede" towards lower-weight bins, effectively skewing the distribution. The second technical issue we overcome is that long-running operations, which experience a lot of concurrency, may in fact be adversarially biased towards the wrong choice, inserting into the more loaded of its two choices with non-trivial probability. We discuss these issues in detail in Section 2.5.1.

In brief, our analysis circumvents these issues by upper bounding the expected number of concurrent operations which increment the counter with the lesser load, thus bounding the error caused by making the wrong choice. Note that the adversary can control the expected number of such operations through increased concurrency. The critical property which we leverage in our analysis is that, while individual operations can be arbitrarily contended (and therefore biased), there is a bound of n on the *average* contention per operation, which in turn bounds an average error adversary can induce over a period of time. Our argument formalizes this intuition, and phrases it in terms of the evolution of the potential function.

We show that this result has implications beyond "parallelizing" the classic two-choice process, as we can leverage it to obtain probabilistic bounds on the *skew* of the MultiCounter. Using the framework of [AKLN17], which connected two-choice load balancing with MultiQueue data structures in the sequential case, we can obtain guarantees for this popular data structure pattern in concurrent executions.

2.2 System Model

Asynchronous Shared Memory. We consider a standard asynchronous shared-memory model, e.g. [AW04], in which n threads (or processes) $P_1, \ldots, P_n$, communicate through shared memory, on which they perform atomic operations such as $read$, $write$, $compare-and-swap$ and $fetch - and - increment$. The fetch-and-increment operation takes no arguments, and returns the value of the register before the increment was performed, incrementing its value by 1.

The Oblivious Adversarial Scheduler. Threads follow an algorithm, composed of shared-memory steps and local computation, including random coin flips. The order of process steps is controlled by an adversarial entity we call the *scheduler*. Time t is measured in terms of the number of shared-memory steps scheduled by the adversary. The adversary may choose to crash a set of at most $n-1$ processes by not scheduling them for the rest of the execution. A process that is not crashed at a certain step is *correct*, and if it never crashes then it takes an infinite number of steps in the execution. In the following, we assume a standard *oblivious* adversarial scheduler, which decides on the interleaving of thread steps independently of the coin flips they produce during the execution.

Shared Objects. The algorithms we consider are implementations of shared objects. A shared object O is an abstraction providing a set of *methods*, each given by a sequential specification. In particular, an implementation of a method n for an object O is a set of n algorithms, one for each executing process. When thread P_i invokes method n of object O, it follows the corresponding algorithm until it receives a response from the algorithm. Upon receiving the response, the process is immediately assigned another method invocation. In the following, we do not distinguish between a method n and its implementation. A method invocation is *pending* at some point in the execution if has been initiated but has not yet received a response. A pending method invocation is *active* if it is made by a *correct* process (note that the process may still crash in the future). For example, a concurrent counter could implement $read$ and $increment$ methods, with the same semantics as those of the sequential data structure.

Linearizability. The standard correctness condition for concurrent implementations is *linearizability* [HW90]: roughly, a linearizable implementation ensures that each concurrent operation can be seen as executing at a single instant in time, called its linearization point. The mapping from method calls to linearization points induces a global order on the method calls, which is guaranteed to be consistent to a sequential execution in terms of the method outputs; moreover, each linearization point must occur between the start and end time of the corresponding method.

Recent work, e.g. [HKP+13], considers deterministic relaxed variants of linearizability, in which operations are allowed to deviate from the sequential specification by a *relaxation* factor. Such relaxations appear to be necessary in the case of data structures such as exact counters or priority queues in order to circumvent strong linear lower bounds on their concurrent complexity [AACH+14]. While specifying such data structures in the concurrent case is well-studied [HKP+13, AKY10, HHH+16], less is known about how to specify structured randomized relaxations.

With High Probability. We say that an event occurs *with high probability* in a parameter, e.g. n, if it occurs with probability at least $1 - 1/n^c$, for some constant $c > 0$.

2.3 The MultiCounter Algorithm

Description. The algorithm implements an approximate counter by distributing updates among n distinct counters, each of which supports atomic $read$ and $increment$ operations. Please see Algorithm 1 for pseudocode. To read the counter value, a thread simply picks one of the n counters uniformly at random, and returns its value multiplied by n. To increment the counter value, the thread picks two counter indices i and j uniformly at random, and reads their current values sequentially. It then proceeds to update (increment) the value of the counter which appeared to have a lower value given its two reads. (In case of a tie, or when the two choices are identical, the tie is broken arbitrarily.)

Algorithm 1 Pseudocode for the MultiCounter Algorithm.

```
Shared: $Counters[n]$ // Array of integers representing set of $n$ distinct counters
function READ( )
    $i \leftarrow$ random$(1, n)$
    return $n \cdot Counters[i]$.read()
end function

function INCREMENT( )
    $i \leftarrow$ random$(1, n)$
    $j \leftarrow$ random$(1, n)$
    $x_i \leftarrow Counters[i]$.read()
    $x_j \leftarrow Counters[j]$.read()
    $Counters[\arg\min(x_i, x_j)]$.increment()
end function
```

Relation to Load Balancing. A *sequential* version of the above process, in which the counter is read or incremented *atomically*, is identical to the classic two-choice balanced allocation process [ABKU99], where each counter corresponds to a bin, and each increment corresponds to a new ball being inserted into the less loaded of two randomly chosen bins.

In a concurrent setting, the critical departure from the sequential model is that the values read can be *inconsistent* with respect to a sequential execution: there may be no single point in time when the two counters had the values x_i and x_j observed by the thread; moreover, these values may change between the point where they are read, and the point where the update is performed.

More technically, the sequential variant of the two-choice process has the crucial property that, at each increment step, it is "biased" towards incrementing the counter of lower value. This does not necessarily hold for the concurrent approximate counter: for an operation where a large number of updates occur between the read and the update points, the read information is stale, and therefore the thread's increment choice may be no better than a perfectly random one; in fact, as we shall see in the analysis, it is actually possible for an adversary to engineer cases where the algorithm's choice is biased towards incrementing the counter of *higher value*.

2.4 Distributional Linearizability

We generalize the classic linearizability correctness condition to cover *randomized relaxed* concurrent data structures, such as the MultiCounter. Intuitively, we will say that a concurrent data structure D is distributionally linearizable to a corresponding *relaxed sequential process* R, defined in terms of a sequential specification S, a cost function $cost$ measuring the deviation from the sequential specification, and a distribution $\mathcal{P}$ on the *cost* function values, such that every execution of D can be mapped onto an execution of the relaxed sequential process R, respecting the outputs and the incurred costs, as well as the order of non-overlapping operations. To formalize this definition, we introduce the following machinery, part of which is adopted from [HKP$^+$13].

Data Structures and Labeled Transition Systems. Let Σ be a set of methods including input and output values. A sequential history s is a sequence over Σ, i.e. an element in Σ^*. A (sequential) data structure is a sequential specification S which is a prefix-closed set of sequential histories. For example, the sequential specification of a stack consists of all valid sequences for a stack, i.e. in which every push places elements on top of the stack, and every pop removes elements from the top of the stack.

Given a sequential specification S, two sequential histories $s, t \in S$ are equivalent, written $s \simeq t$, if they correspond to the same "state:" formally, for any sequence $u \in \Sigma^*$, $su \in S$ iff $tu \in S$. Let $[s]_S$ be the equivalence class of $s \in S$.

Subsequently, as shown in [HKP+13], we have that

Lemma 2.4.1 *If $t \in [s]_S$, then for any sequence $u \in \Sigma^*$, if $su \in S$, then $tu \in [su]_S$.*

Next, using the above notations we define a labelled transition sequence of a data structure.

Definition 2.4.1 *Let S be a sequential specification. Its corresponding labeled transition sequence (LTS) is an object $LTS(S) = (Q, \Sigma, \rightarrow, q_0)$, with states $Q = \{[s]_S | s \in S\}$, set of labels Σ, transition relation $\rightarrow \subseteq Q \times \Sigma \times Q$ given by $[s]_S \rightarrow^m [sm]_S$ iff $sm \in S$,* and *initial state $q_0 = [\epsilon]_S$ (ϵ is an empty state).*

Randomized Quantitative Relaxations. Let $S \in \Sigma^*$ be a data structure with $LTS(S) = (Q, \Sigma, \rightarrow, q_0)$. To obtain a randomized quantitative relaxation of S, we apply the following four steps. The first two steps are identical to deterministic quantitative relaxations [HKP+13], whereas the third defines the probability distribution on costs:

1. **Completion**: We start from $LTS(S)$, and construct a completed labeled transition system, with transitions from any state to any other state by any method:
$$LTS_c(S) = (Q, \Sigma, Q \times \Sigma \times Q, q_0).$$
2. **Cost function:** We add a non-negative cost function $cost : Q \times \Sigma \times Q \rightarrow \mathbb{R}_{\geq 0}$ to the LTS. The transition cost will satisfy
$$cost(q, m, q') = 0 \text{ if and only if } q \rightarrow^m q' \text{ in } LTS(S).$$
We call the sequence $\tau = (m_1, k_1), \ldots, (m_n, k_n)$ of transitions and costs the quantitative trace of κ, denoted by $qtr(\kappa)$.
3. **Probability distribution**: Given an arbitrary state $[s]$ in $LTS(S)$, we define a probability space $(\Omega, \mathcal{F}, \mathcal{P})$ on the set of possible transitions and their corresponding costs from this state, where the sample space Ω is the set of all transitions in $Q \times \Sigma \times Q$, the σ-algebra $\mathcal{F}$ is defined in the straightforward way based on the set of elementary events Ω, and $\mathcal{P}$ is a probability measure $\mathcal{P} : \mathcal{F} \rightarrow [0, 1]$. Crucial point is that probabilities depend only on the state $[s]$. Hence, a randomized quantitative relaxetion induces a Markov chain, whose state at each step is given by the state of the corresponding LTS, and whose transitions are LTS transitions with costs and probabilities as above.

Distributional Linearizability. With this in place, we now define distributionally linearizable data structures:

Definition 2.4.2 *Let D be a randomized concurrent data structure, and let R be a randomized quantitative relaxation R of a sequential specification S with respect to a cost function cost, and a probability distribution $\mathcal{P}$ on costs. We say that D is distributionally linearizable to R*

iff for every concurrent schedule σ, there exists a mapping of completed operations in D under σ to transitions in the quantitative path of R, preserving outputs, and respecting the order of non-overlapping operations. This mapping can be used to associate any schedule σ to a distribution of costs *for D under the schedule σ.*

We now make a few important remarks on this definition.

1. The main difficulty when formally defining the "costs" incurred by D in a concurrent execution is in dealing with the execution history, and with the impact of pending operations on these costs. The above definition allows us to define costs, given a schedule, only in terms of the sequential process R, and bounds the incurred costs in terms of the probability distribution defined in R. This definition ensures that the probability distribution on costs incurred at each step only depends on the current state of the sequential process.

2. The second key question is how to use this definition. One subtle aspect of this definition is that the mapping to the sequential randomized quantitative relaxation is done *per schedule*: intuitively, this is because an adversary might change the schedule, and cause the distribution of costs of the data structure to *change*. Thus, it is often difficult to specify a precise cost distribution, which covers all possible schedules. However, for the data structures we analyze, we will be able to provide *tail bounds* on the cost distributions induced by *all possible schedules*.

The natural next question, which we answer in the following section, is whether non-trivial such data structures exist and can be analyzed.

2.5 Analysis of the MultiCounter

We will focus on proving the following result.

Theorem 2.5.1 *Given an oblivious adversary, n distributed counters and n threads, for any fixed schedule, the MultiCounter algorithm is distributionally linearizable to a randomized relaxed sequential counter process, which, at any step t, returns a value that is at most $O(n \log^2 n)$ away from the number of increments applied up to t, both in expectation and with high probability in n.*

We emphasize that the relaxation guarantees are independent of the time t at which the guarantee is examined, and that they would thus hold in infinite executions.

2.5.1 Modeling the Concurrent Process

In the following, we will focus on analyzing executions formed exclusively of increment operations, whose lower-level steps may be interleaved. (Adding read operations at any point during the execution will be immediate.) We model the process as follows. First, we assume a schedule that is fixed by the adversary. For each thread P_j, and non-negative integers j, we consider a sequence of increment operations $(op_i^{(j)})$, each of which is defined by its starting time $s_i^{(j)}$, corresponding to the time when its first read step was scheduled, and completion time $f_i^{(j)}$, corresponding to the time when its update time is scheduled, such that $s_{i+1}^{(j)} > f_i^{(j)}$ for all i, j. (Recall that the scheduler defines a global order on individual steps.) At most n operations may be active at a given time, corresponding to the fact that we only have n parallel threads.

Next, we map the concurrent execution to the sequential one according to the linearization points of the operations. Linearization point of each operation is a point at which it increments a counter. First, we sort the operations based on the time their increments are scheduled by adversary (ties can be broken arbitrarily). Let op_i be the operation which performs i-th increment according to the sorted order and let s_i and f_i be its start and end times correspondingly. For each operation op_i, we record its *contention* ℓ_i as the number of distinct increment operations scheduled between its start and end time. (Alternatively, we could define this quantity as the number of operations which complete in the time interval (s_i, f_i).) Note that at most $n-1$ distinct operations can be concurrent with op_i at any given time, but the contention for a specific operation is potentially unbounded.

We can rephrase the original process as follows. For each operation op_i, the adversary sets the time when it performs the first and its second read of counter values / bin weights, as well as its contention ℓ_i, by scheduling other operations concurrently. The only constraint on the adversary is that not more than n operations can be active at the same time.

Since the adversary is oblivious, we notice that the update process is equivalent to the following: at the time when the update (increment) is scheduled, the thread executing the operation generates two uniform random indices i and j, and is given values x_i and x_j for the two corresponding counters / bin weights, read at previous (possibly different) points in time. We will stick to the bin weight formulation from now on, with the understanding that the two are equivalent.

The thread will then increase the weight of the bin with the smaller value read (among x_i and x_j) by 1. Alternatively, we will say that the ball with weight one is thrown into the bin with the smaller weight. This formulation has the slight advantage that it makes the update process sequential, by moving the random choices to the time when the update is made, using the principle of deferred decisions. Critically, the bin weights on which the update decision is based are potentially stale. We will focus on this simplified variant of the process in the following.

Discussion. The key difference between the above process and the classic power-of-two-choices process is the fact that the choice of bin / counter which the thread updates is based on stale, potentially invalid information. Recall that key to the strong balancing properties of the classic process is the fact that it is biased towards inserting in *less loaded* bins; the process which inserts into randomly chosen bins is called one-choice process and is known to diverge [PTW15]. In particular, notice it is possible that, by the time when the thread performs the update, the order of the bins' load may have changed, i.e. the thread in fact inserts into the *more loaded* bin among its two choices at the time of the update.

Since the oblivious adversary decides its schedule independently of the threads' random choices, it cannot *deterministically* cause a specific update to insert into the more loaded bin. However, it can *significantly bias* an update towards inserting into the more loaded bin:

Assume for example an execution suffix where all n threads read concurrently at some time t_R[2] and then proceed to perform updates, one after another. Pick an operation op for which

[2]Technically, since we count time in terms of shared-memory operations, these reads occur at consecutive

the gap between the two values read x_i and x_j (at the time of the read) is 1, say $x_i = x_j + 1$. So op will increment x_j. At the same time, notice that all the other operations which read concurrently with op are biased towards inserting in x_j rather than x_i, since its rank (in increasing order of weight) is lower than that of bin i. Hence, as the adversary schedules more and more updates between t_R at op's update time, it is increasingly likely to *invert* the relation between i and j by the time of op's update, causing it to insert into the “wrong" bin.

The previous example suggests that the adversary is able to bias some subset of the operations towards picking the wrong bin at the time of the update. Another issue is that operations which experience high contention, for which there are many updates between the read point and the update point, the read values x_i and x_j become meaningless: for example, if the weights of bin i and j become equal at some time t_0 between t_R and op's update, then from this point in time these two bins appear completely symmetrical to the algorithm, and op's choice given the information that $x_i > x_j$ at t_R may be no better than uniform random.

One issue which further complicates this last example is that, at t_0, there may be a non-zero number of other operations which already made their reads (for instance, at t_R), but have not updated yet. Since these operations read at a point where $x_i > x_j$, they are in fact biased towards inserting in x_j. So, looking at the event that op updates the *less loaded* of its two random choices at update time, we notice that its probability in this example is *strictly worse* than uniform random choice.

We summarize this somewhat lengthy discussion with two points, which will be useful in our analysis:

1. As they experience concurrent updates, operations may accrue bias towards inserting into the *more loaded* of their two random choices.
2. Long-running operations may in fact have a higher probability of inserting into the more loaded bin than into the less loaded one, i.e. may become biased towards making the “wrong" choice at the time of the update.

2.5.2 Proof Strategy

We now briefly go over how our approach circumvents the issues described above. Our goal is to show that the expected gap between the loads of the most loaded bin and the least loaded bin is at most $O(\log^2 n)$. Consider the following version of the sequential two-choice process, which we call g-*bounded* two-choice process, for $g \geq 1$. At each step, we select two bins i and j, uniformly at random. Without loss of generality, we assume that $x_i \geq x_j$ are their loads respectively. Then, in the g-*bounded* process, we throw the ball into the bin j, if $x_i - x_j > g$ and we throw it into the bin i otherwise. In the analysis, we implicitly prove that the gap between the loads of the most loaded bin and the least loaded bin is at most $O(g \log{(ng)})$, in expectation. The proof generalizes the analysis provided in [PTW15]. We define the potential Γ and show that it is bounded by $O(ng)$ in expectation (which by definition results in the $O(g \log{(ng)})$ upper bound on the expected gap). This is accomplished by showing that Γ has supermartingale-like behaviour: it decreases in expectation, once it exceeds the threshold with value $O(ng)$. In order to see the intuition behind the upper bound, consider the example when $n = 2$. In this case, the difference between the loads of two bins is a biased random

times after t_R. However, all their read values are identical to the read value at t_R, and hence we choose to simplify notation in this way.

walk on a line, which starts at 0. Let d be the current difference between the loads. Notice that if $0 \leq d \leq g$ then the walk is biased towards moving to $g+1$, and when $d > g$ the walk is biased towards moving to $d-1$. (For $d = 0$ the process is unbiased or bias depends on the tie-breaking rule, and we assume that it is biased towards moving to 1, without loss of generality). We can observe the similar type of bias when d is negative. Thus, by properties of the biased random walks, the expected distance to 0 is in at most $O(g)$. Subsequently, the expected gap between the loads is $O(g)$ as well. With this in place, we are ready to consider the concurrent two-choice process.

As the simple setting, for every operation, assume that the number of operations concurrent with it is upper bounded by Cn, where C is large enough positive integer. We will say that such operations are **good**. Consider operation op. Without loss of generality, let $x_i \geq x_j$ be the loads of the chosen bins, at the time when op performs update. Notice that if $x_i - x_j > Cn$, we are guaranteed that the load of bin i was larger than the load of bin j regardless of when op read their values. Hence, the concurrent two-choice process can be mapped to the Cn-*bounded* sequential two-choice process. Subsequently, assuming that C is constant, the expected gap is at most $O(n \log n)$. Next, we show how the improve the upper on the gap. Fix an operation op. Notice that since the number of operations which are concurrent with op is at at most Cn and they choose bins uniformly at random, using Chernoff's inequality, we can show that for every bin i, the number of balls thrown into i between the start and end of operation op is at most $O(\log n)$ with high probability. Hence, with high probability, op behaves as an operation of the $O(\log n)$-*bounded* two-choice process, which gives us the intuition that the upper bound can be reduced to $O(\log^2 n)$. Indeed, we are able to show that even in this case, the potential Γ retains the supermartingale-like behaviour, allowing us to prove the upper bound of $O(n \log n)$ on the expected value of Γ. Finally, we sketch how to deal with **bad** operations, which are concurrent with more than Cn operations. The first step is to show that out of Cn consecutive operations at most n are **bad**. Then, notice that even though the **bad** operation op might be biased toward making the wrong choice, for each bin i, the probability that op throws the ball into the bin i is at most $\frac{2}{n}$. This allows us to bound the increase in Γ which is caused by **bad** operations. The crucial point is that if we consider the impact of Cn consecutive operations on Γ, we can prove that if C is large enough, then the increase due to (at most) n **bad** operations is mitigated by $(C-1)n$ **good** operations, which results in the $O(\log^2 n)$ upper bound on the expected gap.

2.5.3 Notation and Background

For any bin i and time step t, let $x_i(t)$ be the weight of bin i at step t (after t balls are thrown in total) and let $x(t) = (x_1(t), x_2(t), ..., x_n(t))$ be a vector of weights. Let $\mu(t) = \sum_{i=1}^{m} x_i(t)/n$ be the average weight of the bins at step t. Let $\alpha < 1$ be a parameter to be fixed later. At each step $t+1$, instead of increasing the weight of some bin by one, we allow the increase $w(t)$ to be a positive random variable. Even though initially we concentrate on the case with counters ($w(t) = 1$), it is useful to prove several general Lemmas with random weights in mind, since we will need to use them later. That is, we allow the weight of the thrown ball to be a random variable.

Define
$$\Phi^{seq}(t) = \sum_{i=1}^{n} e^{\alpha(x_i(t)-\mu(t))}, \text{ and } \Psi^{seq}(t) = \sum_{i=1}^{n} e^{-\alpha(x_i(t)-\mu(t))}.$$

Finally, define the potential function
$$\Gamma^{seq}(t) = \Phi^{seq}(t) + \Psi^{seq}(t).$$

We use superscript seq to denote potential functions given by sequential process, which always throws the ball into the bin with the smaller weight. That is, at time step $t+1$, sequential process picks two bins i and j uniformly at random, compares their weights $x_i(t)$ and $x_j(t)$ and increases the weight of the bin with smaller value by the random variable $w(t)$. In contrast, the concurrent process picks two bins i and j uniformly at random, but to make decision, it can only compare the values of $x_i(t_1)$ and $x_j(t_2)$, for $t_1 \le t_2 \le t$, since adversary has control over the schedule of read operations.

In order to bound Γ^{seq}, $w(t)$ should have the following properties :

$$\mathbb{E}[w(t)] = 1 \tag{2.1}$$

and there exist constants $S \ge 1$ and $\lambda > 0$, such that for any $|x| \le \lambda/2$:

$$\mathbb{E}[(e^{xw(t)})''] = \mathbb{E}[M''(x)] < 2S. \tag{2.2}$$

In the case of counters ($w(t) = 1$), we can use $\lambda = 1$ and $S = 1$ since $e^{\frac{1}{2}} \le 2$.

The main technical result of [PTW15] can be phrased as:

Theorem 2.5.2 *Let $\epsilon = \frac{1}{16}$ and let $\alpha \le \min\left(\frac{\epsilon}{6S}, \frac{\lambda}{2}\right)$ be a parameter as given above. Then there exists a constant $C(\epsilon) = \mathrm{poly}(\frac{1}{\epsilon})$ such that, for any time $t \ge 0$, we have $\mathbb{E}[\Gamma^{seq}(t)] \le \frac{4C(\epsilon)n}{\alpha\epsilon}$.*

We would like to point out that the upper bound on α and the value of ϵ are chosen according to the conditions required in [PTW15] and we will assume that they hold throughout this chapter (Later on, in Lemma 2.5.11, we will assume even smaller upper bound on α).

Our goal will be to prove similar theorem in the concurrent case.

Note that this implies that the maximum gap between the most loaded and the least loaded bin at a step is $2\frac{\log n}{\alpha} + O\left(\frac{\log \frac{1}{\alpha}}{\alpha}\right)$ in expectation and with high probability in n (as shown in [PTW15]).

The proof of the above theorem uses the following Lemma, which we also are going to rely on:

Lemma 2.5.3 *Let α and ϵ and $C(\epsilon)$ be the parameters defined in Theorem 2.5.2. Then for any step t:*

$$\mathbb{E}[\Gamma^{seq}(t+1)|x(t)] \le \Big(1 - \frac{\alpha\epsilon}{4n}\Big)\Gamma^{seq}(t) + C(\epsilon).$$

2.5.4 Naive Upper and Lower Bounds

Let $\Gamma^{con}(t), \Phi^{con}(t)$ and $\Psi^{con}(t)$ be the potential functions in the concurrent case.

In this section we derive upper and lower bounds on Γ^{con} per step. These bounds just use the fact that for any bin i and operation op_{t+1} the probability of op_t increasing the weight of i is at most $\frac{2}{n}$, and this is true both for sequential and concurrent processes.

We assume that at step $t+1$, the weight of the ball $w(t)$ satisfies conditions from Section 2.5.3. We start with the upper bound:

Lemma 2.5.4 *For any operation op_{t+1}*

$$\mathbb{E}[\Gamma^{con}(t+1)|x(t)] \le \left(1 + \frac{4\alpha}{n}\right)\Gamma^{con}(t). \tag{2.3}$$

Proof. First we consider what is expected change in Φ^{con}. Let $y_i = x_i(t) - \mu(t)$ and let $\Phi_i^{con}(t) = e^{\alpha y_i}$. Also, let $\Delta\Phi^{con} = \Phi^{con}(t+1) - \Phi^{con}(t)$ and $\Delta\Psi^{con} = \Psi^{con}(t+1) - \Psi^{con}(t)$ We have two cases to consider. If ball is thrown into the bin i, then the change is:

$$\begin{aligned}
\mathbb{E}[\Delta\Phi_i^{con}|x(t)] &= \mathbb{E}[\Phi_i^{con}(t+1)|x(t)] - \Phi_i^{con}(t) \\
&= \mathbb{E}\left[\exp\left(\alpha\Big(x_i(t) - \mu(t) + (w(t) - \frac{w(t)}{n})\Big)\right)\middle| x(t)\right] - e^{\alpha y_i} \\
&= e^{\alpha y_i}\left(\mathbb{E}\left[\exp\left(w(t)\alpha(1-\frac{1}{n})\right)\right] - 1\right) \\
&= e^{\alpha y_i}\left(\mathbb{E}\left[M\Big(\alpha(1-\frac{1}{n})\Big)\right] - 1\right) \\
&\overset{(*)}{=} e^{\alpha y_i}\left(\mathbb{E}\left[M(0) + M'(0)\alpha(1-\frac{1}{n}) + M''(\xi)\alpha^2(1-\frac{1}{n})^2\Big/2\right] - 1\right) \\
&= e^{\alpha y_i}\left(\mathbb{E}\left[1 + w(t)\alpha(1-\frac{1}{n}) + M''(\xi)\alpha^2(1-\frac{1}{n})^2\Big/2\right] - 1\right) \\
&\overset{(2.1),(2.2)}{\leq} e^{\alpha y_i}\left(\alpha(1-\frac{1}{n}) + S\alpha^2(1-\frac{1}{n})^2\right)
\end{aligned}$$

Where in $(*)$ we used the Taylor expansion of $M(x)$ around 0 and in the last step we used that $0 \leq \xi \leq \alpha(1-\frac{1}{n}) \leq \frac{\lambda}{2}$.

Using similar arguments we can prove that when the ball is not thrown into bin i:

$$\Delta\Phi_i^{con} \leq e^{\alpha y_i}\left(-\frac{\alpha}{n} + S\frac{\alpha^2}{n^2}\right) \leq 0.$$

Let $p_i \leq 2/n$ be the probability of the ball being thrown into the bin i. We get that:

$$\begin{aligned}
\mathbb{E}\left[\Delta\Phi_i^{con}|x(t)\right] &\leq p_i e^{\alpha y_i}\left(\alpha(1-\frac{1}{n}) + S\alpha^2(1-\frac{1}{n})^2\right) \\
&\leq p_i(\alpha + S\alpha^2) \leq \frac{4\alpha}{n}e^{\alpha y_i}.
\end{aligned}$$

Hence:

$$\mathbb{E}[\Delta\Phi^{con}|x(t)] = \sum_{i=1}^{n}\mathbb{E}[\Delta\Phi_i^{con}|x(t)] \leq \frac{4\alpha}{n}\Phi^{con}(t). \tag{2.4}$$

In a similar way, we can prove that:

$$\begin{aligned}
\mathbb{E}[\Delta\Psi^{con}|x(t)] &\leq \sum_{i=1}^{n}(1-p_i)(\frac{\alpha}{n} + \frac{S\alpha^2}{n^2})e^{-\alpha y_i} \\
&\leq \sum_{i=1}^{n}(\frac{\alpha}{n} + \frac{S\alpha^2}{n^2})e^{-\alpha y_i} \leq \frac{4\alpha}{n}\Psi^{con}(t)
\end{aligned}$$

Combining this with inequality (2.4), and using the definitions of $\Delta\Phi^{con}$ and $\Delta\Psi^{con}$ gives us proof of the Lemma. □

We proceed by showing the lower bound:

Lemma 2.5.5 *For any operation op_{t+1}*

$$\mathbb{E}[\Gamma^{con}(t+1)|x(t)] \geq \left(1 - \frac{2\alpha}{n}\right)\Gamma^{con}(t). \tag{2.5}$$

Proof. First we consider what is expected change in Φ^{con}. Let $y_i = x_i(t) - \mu(t)$ and let $\Phi_i^{con}(t) = e^{\alpha y_i}$. We have two cases here. If the ball is thrown into bin i, then as in the previous lemma the change is:

$$\begin{aligned}
&\mathbb{E}[\Delta\Phi_i^{con}|x(t)] = \mathbb{E}[\Phi_i^{con}(t+1)|x(t)] - \Phi_i^{con}(t) \\
&= e^{\alpha y_i}\left(\mathbb{E}\left[1 + w(t)\alpha(1-\frac{1}{n}) + M''(\xi)\alpha^2(1-\frac{1}{n})^2\Big/2\right] - 1\right) \\
&\geq e^{\alpha y_i}\alpha(1-\frac{1}{n}) \geq \frac{\alpha}{2}e^{\alpha y_i} \geq 0.
\end{aligned}$$

Where in the last step we used that $n \geq 2$ and the fact that exponential function is non-negative.

Using similar arguments we can prove that, when the ball is not throw into bin i:

$$\Delta\Phi_i^{con} \geq -\frac{\alpha}{n}e^{\alpha y_i}.$$

Let $p_i \leq 2/n$ be the probability that the ball is thrown into the bin i. We get that:

$$\mathbb{E}\left[\Delta\Phi_i^{con}|x(t)\right] \geq -(1-p_i)\frac{\alpha}{n}e^{\alpha y_i} \geq -\frac{\alpha}{n}e^{\alpha y_i}.$$

Hence:

$$\mathbb{E}[\Delta\Phi^{con}|x(t)] = \sum_{i=1}^{n}\mathbb{E}[\Delta\Phi_i^{con}|x(t)] \geq -\frac{\alpha}{n}\Phi^{con}(t). \tag{2.6}$$

In a similar way, we can prove that:

$$\mathbb{E}[\Delta\Psi^{con}|x(t)] \geq -\sum_{i=1}^{n}p_i\alpha(1-\frac{1}{n}) \geq -\frac{2\alpha}{n}\Psi^{con}(t)$$

Combining this with inequality (2.6), and using the definitions of $\Delta\Phi_i^{con}$ and $\Delta\Psi_i^{con}$ gives us proof of the Lemma. □

2.5.5 Main Argument

Let $C \geq 1$ be a constant which we will fix later. Recall that ℓ_t is the number of operations which run concurrently with operation op_t. We call operation op_t **good** if $\ell_t \leq Cn$, otherwise we call it **bad**. We start by considering Cn consecutive operations and proving that at most n of them can be bad:

Lemma 2.5.6 *For any t, we have that $|t' : t \leq t' \leq t + Cn - 1, \ell_{t'} > Cn| < n$.*

Proof. We argue by contradiction. Let us assume that the number of bad operations is at least n. By the pigeonhole principle, there exist bad operations op_i and op_j, $t \leq i < j \leq t+Cn-1$, which are performed by the same thread. This means that since these operations are not concurrent, we have that $s_j > f_i$. Thus, we get a contradiction:

$$Cn \leq \ell_j = |t' : s_j \leq f_{t'} < f_j| \leq |t' : f_i \leq f_{t'} < f_j| \leq j - i < Cn$$

. □

Upper Bound on a Potential for Counters. In the following we concentrate on the case when $w(t) = 1$, for any t (the case with counters). We start by defining random variables which help us to upper bound Γ^{con}.

Definition 2.5.1 *For each bin i, and step $t \geq Cn$, let $H_i(t)$ be the number of times i was chosen by operations op_{t-Cn+1}, op_{t-Cn+1}, ..., op_t. In this case, if some operation chooses bins i and j, we count both as chosen and we also say that i was chosen twice if $i = j$. Observe that if op_{t+1} is good, then $H_i(t)$ is the upper bound on the number of increments bin i receives during the entire run of operation op_{t+1} (excluding the increment which might be performed by op_{t+1}). Also, let $H_{max}(t) = max\{H_1(t), H_2(t), ..., H_n(t)\}$.*

With this in place we are ready to bound the potential.

Lemma 2.5.7 *For any **good** operation op_{t+1}, such that $t \geq Cn$:*

$$\mathbb{E}[\Gamma^{con}(t+1)|x(t), H_{max}(t)] \leq \Big(1 - \frac{\alpha\epsilon}{4n}\Big)\Gamma^{con}(t) + C(\epsilon) + \frac{4\alpha\Gamma^{con}(t)}{n}(e^{\alpha H_{max}(t)} - 1).$$

Proof. Since we condition on $x(t)$, we can assume that potentials at step t are the same both for sequential and concurrent processes (This is not true for the next step since processes can increment weights of different bins). We have that:

$$\begin{aligned}
&\mathbb{E}[\Gamma^{con}(t+1)|x(t), H_{max}(t)] \\
&\quad = \mathbb{E}[\Gamma^{seq}(t+1)|x(t), H_{max}(t)] + \mathbb{E}[\Gamma^{con}(t+1) - \Gamma^{seq}(t+1)|x(t), H_{max}(t)] \\
&\quad \overset{\text{Lemma 2.5.3}}{\leq} \Big(1 - \frac{\alpha\epsilon}{4n}\Big)\Gamma^{seq}(t) + C(\epsilon) \\
&\qquad\qquad + \mathbb{E}[\Gamma^{seq}(t+1) - \Gamma^{con}(t+1)|x(t), H_{max}(t)].
\end{aligned}$$

Hence our goal is to upper bound $\mathbb{E}[\Gamma^{seq}(t+1) - \Gamma^{con}(t+1)|x(t), H_{max}(t)]$. w.l.o.g we assume that $x_1(t) \leq x_2(t) ... \leq x_n(t)$. We couple sequential and concurrent processes so that the bin choices i and j are the same in both cases. Let $i \leq j$, then sequential process always increments the weight of bin i, but for the concurrent process it depends on when its reads occurred, for example it can be that during reads the weight of bin j was smaller than the weight of bin i but then the increments done by concurrent processes reversed the order. The crucial thing is that in this case $x_j(t) - x_i(t) \leq H_{max}(t)$. Hence, assuming the worst case (concurrent process increments the weight of bin j) we have that

$$\begin{aligned}
\Phi^{con}(t+1) - \Phi^{seq}(t+1) &= e^{\alpha(x_j(t) - \mu(t) + 1 - \frac{1}{n})} + e^{\alpha(x_i(t) - \mu(t) - \frac{1}{n})} \\
&\quad - e^{\alpha(x_i(t) - \mu(t) + 1 - \frac{1}{n})} - e^{\alpha(x_j(t) - \mu(t) - \frac{1}{n})} \\
&= e^{\alpha(x_i(t) - \mu(t))} e^{-\frac{\alpha}{n}} (e^{\alpha} - 1)(e^{\alpha(x_j(t) - x_i(t))} - 1) \\
&\leq 2\alpha e^{\alpha(x_i(t) - \mu(t))}(e^{\alpha H_{max}(t)} - 1).
\end{aligned}$$

Where in the last step we used that $e^{\alpha} \leq 1 + 2\alpha$, since $\alpha \leq \frac{1}{2}$. Also,

$$\begin{aligned}
\Psi^{con}(t+1) - \Psi^{seq}(t+1) &= e^{-\alpha(x_j(t) - \mu(t) + 1 - \frac{1}{n})} + e^{-\alpha(x_i(t) - \mu(t) - \frac{1}{n})} \\
&\quad - e^{-\alpha(x_i(t) - \mu(t) + 1 - \frac{1}{n})} - e^{-\alpha(x_j(t) - \mu(t) - \frac{1}{n})} \\
&= e^{-\alpha(x_i(t) - \mu(t))} e^{\frac{\alpha}{n}} (e^{-\alpha} - 1)(e^{-\alpha(x_j(t) - x_i(t))} - 1) \\
&= e^{-\alpha(x_i(t) - \mu(t))} e^{\frac{\alpha}{n}} (1 - e^{-\alpha})(1 - e^{-\alpha(x_j(t) - x_i(t))}) \\
&\leq 2\alpha e^{-\alpha(x_i(t) - \mu(t))}(1 - e^{-\alpha H_{max}(t)}).
\end{aligned}$$

Where in the last step we used that $e^{\frac{\alpha}{n}} \leq 2$, since $\alpha \leq \frac{1}{2}$, and $e^{-\alpha} \geq 1 - \alpha$. The above bounds no longer depend on j and for any bin i the probability of being one out of two random choices of op_{t+1} is at most $\frac{2}{n}$, hence:

$$\begin{aligned}
&\mathbb{E}[\Gamma^{seq}(t+1) - \Gamma^{con}(t+1)|x(t)] \\
&\leq \sum_{i=1}^{n} \frac{4\alpha}{n} e^{\alpha(x_i(t)-\mu(t))}(e^{\alpha H_{max}(t)} - 1) + \sum_{i=1}^{n} \frac{4\alpha}{n} e^{-\alpha(x_i(t)-\mu(t))}(1 - e^{-\alpha H_{max}(t)}) \\
&= \frac{4\alpha \Phi^{con}(t)}{n}(e^{\alpha H_{max}(t)} - 1) + \frac{4\alpha \Psi^{con}(t)}{n}(1 - e^{-\alpha H_{max}(t)}) \\
&\leq \frac{4\alpha \Gamma^{con}(t)}{n}(e^{\alpha H_{max}(t)} - 1).
\end{aligned}$$

Where in the last step we used that $e^{\alpha H_{max}(t)} + e^{-\alpha H_{max}(t)} \geq 2$. □

Let $N = \lfloor \frac{2Cn}{2e^3 C \log n} \rfloor$ and for $0 \leq K \leq N$, let $A_K(t)$ be the event that $2e^3 CK \log n \leq H_{max}(t) < 2e^3 C(K+1) \log n$. We proceed by proving the following lemma:

Lemma 2.5.8 *For any **good** operation op_{t+1}, such that $t \geq Cn$:*

$$\begin{aligned}
\mathbb{E}[\Gamma^{con}(t+1)] &\leq \Big(1 - \frac{\alpha\epsilon}{4n}\Big) \mathbb{E}[\Gamma^{con}(t)] + C(\epsilon) \\
&+ \sum_{K=0}^{N} \frac{4\alpha\, \mathbb{E}[\Gamma^{con}(t)|A_K(t)] Pr[A_K(t)]}{n}(e^{2\alpha e^3 C(K+1)\log n} - 1).
\end{aligned}$$

Proof. First, we remove conditioning from the inequality proved in Lemma 2.5.7).

$$\begin{aligned}
&\mathbb{E}[\Gamma^{con}(t+1)] = \mathbb{E}_{x(t),H_{max}(t)}[\mathbb{E}[\Gamma^{con}(t+1)|x(t), H_{max}(t)]] \\
&\leq \mathbb{E}_{x(t),H_{max}(t)} \Bigg[\Big(1 - \frac{\alpha\epsilon}{4n}\Big) \mathbb{E}[\Gamma^{con}(t)|x(t), H_{max}(t)] + C(\epsilon) \\
&\qquad + \frac{4\alpha\, \mathbb{E}[\Gamma^{con}(t)(e^{\alpha H_{max}(t)} - 1)|x(t), H_{max}(t)]}{n} \Bigg] \\
&\leq \Big(1 - \frac{\alpha\epsilon}{4n}\Big) \mathbb{E}[\Gamma^{con}(t)] + C(\epsilon) + \frac{4\alpha\, \mathbb{E}[\Gamma^{con}(t)(e^{\alpha H_{max}(t)} - 1)]}{n}.
\end{aligned}$$

Next, to finish the proof of the Lemma, we use the upper bound on $H_{max}(t)$, which is implied by conditioning on the event $A_K(t)$:

$$\begin{aligned}
&\mathbb{E}[\Gamma^{con}(t)(e^{\alpha H_{max}(t)} - 1)] = \sum_{K=0}^{N-1} \mathbb{E}[\Gamma^{con}(t)(e^{\alpha H_{max}(t)} - 1)|A_K(t)] Pr[A_K(t)] \\
&\leq \sum_{K=0}^{N-1} \mathbb{E}[\Gamma^{con}(t)(e^{2\alpha e^3 C(K+1)\log n} - 1)|A_K(t)] Pr[A_K(t)].
\end{aligned}$$

□

Our next goal is to upper bound $Pr[A_K(t)]$ and $\mathbb{E}[\Gamma^{con}(t)|A_K(t)]$. We start with deriving the concentration bounds for $H_{max}(t)$.

Lemma 2.5.9 *For any $t > Cn$ and constant $K \geq 1$:*

$$Pr[A_K(t)] \leq Pr[H_{max}(t) \geq 2Ke^3 C \log n] \leq \frac{1}{(eK \log n)^{2KCe^3 \log n}}.$$

Proof. Note that $H_{max}(t)$ is a maximum number of balls some bin receives if we throw $2Cn$ balls into n initially empty bins (Recall that all the random choices which operations make are independent). For a fixed bin i, let $H_i(t)$ be the number of balls it receives. We know that $\mathbb{E}[H_i(t)] = 2C$. Hence, using Chernoff's inequality we get that

$$\begin{aligned} Pr[H_i(t) \geq 2Ke^3C\log n] &\leq \left(\frac{e^{Ke^3\log n - 1}}{(Ke^3\log n)^{Ke^3\log n}} \right)^{2C} \\ &\leq \frac{1}{n}\frac{1}{(eK\log n)^{2KCe^3\log n}}. \end{aligned}$$

we get the proof of the Lemma by union bounding over n bins. □

We proceed by upper bounding $\mathbb{E}[\Gamma^{con}(t)|A_K(t)]$.

Lemma 2.5.10 *For any $t \geq Cn$:*

$$\mathbb{E}[\Gamma^{con}(t)|A_K(t)] \leq \mathbb{E}[\Gamma^{con}(t)]e^{3\alpha e^3C(K+1)\log n}.$$

Proof.

$$\begin{aligned} &\mathbb{E}[\Phi^{con}(t)|A_K(t), x(t-Cn)] - \Phi^{con}(t-Cn) \\ &= \sum_{i=1}^{n} e^{\alpha(x_i(t-Cn)-\mu(t-Cn))}\left(e^{\alpha\left(x_i(t)-\mu(t)-x_i(t-Cn)+\mu(t-Cn)\right)} - 1 \right). \end{aligned}$$

Since we condition on $A_k(t)$, for every i we have that

$$x_i(t) - x_i(t-Cn) \leq H_{max}(t) \leq 2e^3C(K+1)\log n.$$

Also, $\mu(t) > \mu(t-Cn)$. Thus

$$\begin{aligned} &\mathbb{E}[\Phi^{con}(t)|A_K(t), x(t-Cn)] - \Phi^{con}(t-Cn) \\ &\leq \sum_{i=1}^{n} e^{\alpha(x_i(t-Cn)-\mu(t-Cn))}\left(e^{2\alpha e^3C(K+1)\log n} - 1 \right) \\ &= \Phi^{con}(t-Cn)\left(e^{2\alpha e^3C(K+1)\log n} - 1 \right). \end{aligned}$$

Similarly

$$\begin{aligned} &\mathbb{E}[\Psi^{con}(t)|A_K(t), x(t-Cn)] - \Psi^{con}(t-Cn) \\ &= \sum_{i=1}^{n} e^{-\alpha(x_i(t-Cn)-\mu(t-Cn))}\left(e^{-\alpha\left(x_i(t)-\mu(t)-x_i(t-Cn)+\mu(t-Cn)\right)} - 1 \right). \end{aligned}$$

We have that $x_i(t) \geq x_i(t-Cn)$ and

$$\mu(t) - \mu(t-Cn) \leq H_{max}(t) \leq 2e^3C(K+1)\log n$$

Thus

$$\begin{aligned} &\mathbb{E}[\Phi^{con}(t)|A_K(t), x(t-Cn)] - \Phi^{con}(t-Cn) \\ &\leq \sum_{i=1}^{n} e^{-\alpha(x_i(t-Cn)-\mu(t-Cn))}\left(e^{2\alpha e^3C(K+1)\log n} - 1 \right) \\ &= \Psi^{con}(t-Cn)\left(e^{2\alpha e^3C(K+1)\log n} - 1 \right). \end{aligned}$$

Hence

$$\begin{aligned}&\mathbb{E}[\Gamma^{con}(t)|A_K(t), x(t-Cn)] - \Gamma^{con}(t-Cn)]\\&\le \Gamma^{con}(t-Cn)\left(e^{2\alpha e^3 C(K+1)\log n} - 1\right)\end{aligned}$$

Notice that $A_K(t)$ is independent of $x(t-Cn)$, since in the definition of $H_{max}(t)$ we just consider random choices made by $op_{t-Cn+1}, ..., op_t$. This allows us to remove conditioning on $x(t-Cn)$ and after regrouping the terms in the above inequality we get

$$\mathbb{E}[\Gamma^{con}(t)|A_K(t)] \le \mathbb{E}[\Gamma^{con}(t-Cn)]e^{2\alpha e^3 C(K+1)\log n} \tag{2.7}$$

By applying Lemma 2.5.5 Cn times we get that

$$\mathbb{E}[\Gamma^{con}(t)|x(t-Cn)] \ge \Gamma^{con}(t-Cn)\left(1-\frac{2\alpha}{n}\right)^{Cn} \ge \Gamma^{con}(t-Cn)e^{-4C\alpha}.$$

After removing conditioning we get that

$$\mathbb{E}[\Gamma^{con}(t)] \ge \mathbb{E}[\Gamma^{con}(t-Cn)]e^{-4C\alpha}.$$

By combining the above inequality with (2.7) we get that:

$$\begin{aligned}\mathbb{E}[\Gamma^{con}(t)|A_K(t)] &\le \mathbb{E}[\Gamma^{con}(t)]e^{2\alpha e^3 C(K+1)\log n}e^{4\alpha C}\\&\le \mathbb{E}[\Gamma^{con}(t)]e^{3\alpha e^3 C(K+1)\log n}.\end{aligned}$$

□

Finally

Lemma 2.5.11 *For any good operation op_{t+1}, such that $t \ge Cn$, we have that if $C \ge 2$ and $\alpha \le \frac{1}{4096Ce^3\log n}$ then*

$$\mathbb{E}[\Gamma^{con}(t+1)] \le \left(1-\frac{\alpha\epsilon}{8n}\right)\mathbb{E}[\Gamma^{con}(t)] + C(\epsilon).$$

Proof. Since $\alpha \le \frac{1}{4096Ce^3\log n}$ and $C \ge 2$:

$$\begin{aligned}&\sum_{K=1}^{N}\mathbb{E}[\Gamma^{con}(t)|A_K(t)]Pr[A_K(t)](e^{2\alpha e^3 C(K+1)\log n}-1)\\&\overset{\text{Lemmas 2.5.9 and 2.5.10}}{\le}\sum_{K=1}^{N}\frac{\mathbb{E}[\Gamma^{con}(t)]e^{5\alpha e^3 C(K+1)\log n}}{(eK\log n)^{2KCe^3\log n}}\\&\le \sum_{K=1}^{N}\frac{\mathbb{E}[\Gamma^{con}(t)]e^{e^3CK\log n}}{e^{2KCe^3\log n}} \le \sum_{K=1}^{N}\frac{\mathbb{E}[\Gamma^{con}(t)]}{e^{2Ke^3\log n}} \le \sum_{K=1}^{\infty}\frac{\mathbb{E}[\Gamma^{con}(t)]}{n^{16K}}\\&\le \frac{2\,\mathbb{E}[\Gamma^{con}(t)]}{n^{16}} \le \frac{\mathbb{E}[\Gamma^{con}(t)]}{2048}.\end{aligned} \tag{2.8}$$

Also, for $K = 0$

$$\begin{aligned}&\mathbb{E}[\Gamma^{con}(t)|A_0(t)]Pr[A_0(t)](e^{2\alpha e^3 C\log n}-1)\\&\overset{\text{Lemma 2.5.10}}{\le}\mathbb{E}[\Gamma^{con}(t)]e^{3\alpha e^3 C\log n}(e^{2\alpha e^3 C\log n}-1)\\&\le \mathbb{E}[\Gamma^{con}(t)]e^{\frac{3}{4096}}(e^{\frac{1}{2048}}-1)\\&\le 2\,\mathbb{E}[\Gamma^{con}(t)]\frac{1}{1024} = \frac{\mathbb{E}[\Gamma^{con}(t)]}{512}.\end{aligned}$$

Hence, we get that

$$\sum_{K=0}^{N} \mathbb{E}[\Gamma^{con}(t)|A_K(t)]Pr[A_K(t)](e^{2\alpha e^3 C(K+1)\log n} - 1)$$
$$\leq \frac{\mathbb{E}[\Gamma^{con}(t)]}{2048} + \frac{\mathbb{E}[\Gamma^{con}(t)]}{512} = \frac{5\,\mathbb{E}[\Gamma^{con}(t)]}{2048}.$$

By plugging the above inequality in Lemma 2.5.8 we get that

$$\mathbb{E}[\Gamma^{con}(t+1)] \leq \left(1 - \frac{\alpha\epsilon}{4n}\right)\mathbb{E}[\Gamma^{con}(t)] + C(\epsilon) + \frac{20\alpha\,\mathbb{E}[\Gamma^{con}(t)]}{2048n}.$$

Recall that $\epsilon = \frac{1}{12}$, thus $\frac{20}{2048} \leq \frac{\epsilon}{8}$ and this finishes the proof of the lemma. □

Endgame. With all this machinery in place, we proceed by proving the following Lemma.

Lemma 2.5.12 *If $\alpha \leq \frac{1}{4096Ce^3\log n}$ and $C \geq 433$, then at any time step t*

$$\mathbb{E}[\Gamma^{con}(t)] \leq \frac{146C(\epsilon)n}{\alpha\epsilon}.$$

Proof. The proof is by induction on t. We will first prove that, if $\mathbb{E}[\Gamma^{con}(t)] \leq \frac{146C(\epsilon)n}{\alpha\epsilon}$ for $t \geq Cn$, then $\mathbb{E}[\Gamma^{con}(t+Cn)] \leq \frac{146C(\epsilon)n}{\alpha\epsilon}$.

We have two cases. The first is if there exists a time $\tau \in [t, t+Cn]$ such that $\mathbb{E}[\Gamma^{con}(\tau)] \leq \frac{72C(\epsilon)n}{\alpha\epsilon}$. Let us now focus on bounding the maximum expected value of $\Gamma^{con}(t+Cn)$ in this case. First, notice that the maximum expected increase of Γ^{con} because of a good step is an additive $C(\epsilon)$ factor. By Lemma 2.5.4, the expected value of Γ^{con} after a bad operation is upper bounded a multiplicative $(1+\frac{4\alpha}{n})$ factor. Hence, by Lemma 2.5.6 and expected maximum value of Γ^{con} at $t+Cn$ is at most

$$\left(\frac{72C(\epsilon)n}{\alpha\epsilon} + C(\epsilon)(C-1)n\right)\left(1+\frac{4\alpha}{n}\right)^n \leq \left(\frac{72C(\epsilon)n}{\alpha\epsilon} + \frac{C(\epsilon)n}{4096\alpha e^3\log n}\right)e^{4\alpha}$$
$$\leq \frac{146C(\epsilon)}{\alpha\epsilon}.$$

The second case is if there exists no such time in $[t, t+Cn]$, meaning that $\mathbb{E}[\Gamma^{con}(\tau)] > \frac{72C(\epsilon)n}{\alpha\epsilon}, \forall\tau \in [t, t+Cn]$. Then, by Lemma 2.5.11, we have that, at each good step,

$$\mathbb{E}[\Gamma^{con}(t+1)] \leq \mathbb{E}[\Gamma^{con}(t)]\left(1 - \frac{\alpha\epsilon}{9n}\right). \tag{2.9}$$

Hence, we can expand the recursion to upper bound the change in Γ^{con} between t and $t+Cn$ as

$$\mathbb{E}[\Gamma^{con}(t+Cn)] \leq \mathbb{E}[\Gamma^{con}(t)]\left(1 - \frac{\alpha\epsilon}{9n}\right)^{(C-1)n}\left(1+\frac{4\alpha}{n}\right)^n$$
$$\leq \mathbb{E}[\Gamma^{con}(t)]e^{-\frac{\alpha\epsilon(C-1)}{9}+4\alpha} \leq \mathbb{E}[\Gamma^{con}(t)].$$

Where in the last step we used that $C \geq 1 + 36/\epsilon = 433$.

To establish the base of induction, note that by Lemma 2.5.4, for each $0 \leq t \leq 2Cn$:

$$\mathbb{E}[\Gamma^{con}(t)] \leq \Gamma^{con}(0)(1+\frac{4\alpha}{n})^{2Cn} = 2n(1+\frac{4\alpha}{n})^{2Cn}$$
$$\leq 2ne^{8\alpha C} \leq 4n \leq \frac{146C(\epsilon)n}{\alpha\epsilon}.$$

This concludes the proof of the Lemma. □

The following Lemma completes the proof of Theorem 2.5.1.

Lemma 2.5.13 *Given an oblivious adversary, n distributed counters and n threads, for any time t in the execution of the approximate counter algorithm the counter returns a value that is at most $O(n\log^2 n)$ away from the number of increment operations which completed up to time t, in expectation. Moreover, for any t and all R sufficiently large, we have*

$$\Pr\left[\exists i : |n \cdot x_i(t) - n \cdot \mu_i(t)| > Rn\log^2 n\right] \leq n^{-\Omega(R)} .$$

Proof. The proof is similar to [PTW15] (the main difficulty was to upper bound the potential). We aim to bound $Gap(t)$, the maximum gap between the weight of two bins at a step.

By choosing $C = 433$ and $\alpha = \frac{1}{4096Ce^3\log n} = \Theta(\frac{1}{\log n})$ and applying Lemma 2.5.12 we get that $\mathbb{E}[\Phi^{con}(t)] = O(n\log n)$ and $\mathbb{E}[\Psi^{con}(t)] = O(n\log n)$ for all t. Let $x_{max}(t)$ denote the maximum weight of any bin at time t, and let $x_{min}(t)$ be the minimum weight of any bin. Then, we have

$$\begin{aligned}\alpha\,\mathbb{E}[x_{max}(t) - \mu(t)] &= \log(\exp\left(\mathbb{E}[\alpha(x_{max}(t) - \mu(t))]\right)) \\ &\overset{(a)}{\leq} \log\mathbb{E}[\exp(\alpha(x_{max}(t) - \mu(t)))] \\ &\overset{(b)}{\leq} \log\mathbb{E}[\Phi^{con}(t)] \leq O(\log n + \log\log n) = O(\log n) ,\end{aligned}$$

where (a) follows from Jensen's inequality, and (b) follows from the definition of Φ^{con}. Similarly, we have $\mathbb{E}[\mu(t) - x_{min}(t)] \leq O(\log^2 n)$. Since the true value of the counter at time t is $n \cdot \mu(t)$, these two statements imply that for all i, we have $\mathbb{E}[|n \cdot x_i(t) - n \cdot \mu(t)|] \leq O(n\log^2 n)$, as desired.

We now prove the high probability bound. Observe that if $x_{max}(t) - \mu(t) > R\log^2 n$, then we have $\Gamma^{con}(t) \geq \Phi^{con}(t) \geq e^{\alpha R\log^2 n}$. Hence, for large enough R:

$$\begin{aligned}\Pr[x_{max}(t) - \mu(t) > R\log^2 n] &\leq \Pr[\Phi^{con}(t) \geq e^{\alpha R\log^2 n}] \\ &\overset{Markov}{\leq} \frac{O(n\log n)}{e^{\alpha R\log^2 n}} \\ &\leq n^{-\Omega(R)} .\end{aligned}$$

Similarly, $\Pr[\mu(t) - x_{min}(t) > R\log^2 n] \leq n^{-\Omega(R)}$.

Combining these two guarantees with a union bound immediately yields the desired guarantee. □

2.6 Distributional Linearizability for Concurrent Relaxed Queues

We now extend the analysis in the previous section to imply distributional linearizability guarantees in concurrent executions for a variant of the MultiQueue process analyzed by [AKLN17]. This process is presented in Algorithm 2. We note that this process applies specifically to implement general concurrent *queues*, and will also apply to *priority queues* assuming that a sufficiently large buffer of elements always exists in the queues such that no insertion is ever performed on an element of *higher* priority than an element which has already been removed.

2.6.1 Application to Concurrent Relaxed Queues

Description. We wish to implement a concurrent data structure with queue like semantics, so that we have guarantees on the rank of dequeued elements. We assume we are given a set of n linearizable priority queues such that each supports $Add(e, p)$, $DeleteMin$, $ReadMin$, where p is the priority of the element, and $ReadMin$ returns the element with smallest priority in the priority queue, but does not remove it. We also assume that each processor i has access to a clock $Clock_i$ which gives an absolute time, and which are consistent amongst all the processors, that is, if processor i reads $Clock_i$ in the linearization before processor j reads $Clock_j$, then processor i's value is smaller. Such an assumption is realistic; recent Intel processors support the RDTSC hardware operation, which provides this functionality for cores on the same socket.

The procedure, given formally in Algorithm 2, is similar to our approximate counter. To enqueue, a thread reads the wall clock, chooses a random priority queue, and adds the element to that priority queue with priority given by the time. To dequeue, we choose two random priority queues, find the one having a higher priority element on top, and delete from that priority queue. In case two processes enqueue to the same priority queue concurrently, their clock values will ensure a consistent ordering, handled by the internal implementation of the priority queues.

Algorithm 2 Pseudocode for Relaxed Queue Algorithm.

```
Shared: $PQs[n]$ // Set of $n$ distinct priority queues
individual: $Clock_i$ // A wall clock for processor $i$, for each $i$
function Enqueue($e$)
    $p \leftarrow Clock_i$.Read()
    $i \leftarrow$ random$(1, n)$
    $PQs[i]$.Add$(e, p)$
end function
function Dequeue( )
    $i \leftarrow$ random$(1, n)$
    $j \leftarrow$ random$(1, n)$
    $(e_i, p_i) \leftarrow PQs[i]$.ReadMin()
    $(e_j, p_j) \leftarrow PQs[j]$.ReadMin()
    if ( $p_i > p_j$ ): $i = j$
    return $PQs[i].DeleteMin()$
end function
```

Analysis. The Analysis mostly follows the steps in [AKLN17]. We define the rank of element with timestamp p as the number of elements which are currently in the system and have timestamp with value at most p (Including itself, and assuming that no two operations have the same timestamp).

First we assume that Dequeues operations never see an empty queue. Given this assumption we can also assume:

- Enqueue operations happen sequentially, sorted by linearization order.
- Dequeue operations are invoked after all Enqueue operations are finished.

Since the timestamps are increasing in linearization order, the two assumptions above do not change the outcome (the rank of returned element) of Dequeue operations and are needed solely for the purpose of analysis.

We proceed by defining the auxiliary exponential label process. We are given n, initially empty queues in which we insert infinitely many labels as follows: for each queue i, if the last inserted label in i is v_i (0 if the queue is empty), then we insert label $v_i + Exp(\frac{1}{n})$ in it. We define the rank of label v as the number of labels which are currently in the queues and have value at most v (Since exponential distribution is continuous we assume that no labels have the same value). We will call these queues label queues to distinguish them from queues we use in Algorithm 2.

Theorem 2 in [AKLN17] says that for any rank r and label queue i, probability of label with rank r being in queue i is $\frac{1}{n}$, and this is independent of the location of all the other labels. The proof uses the memorylessness of exponential distribution. Since for each queue the probability of element e with initial (before $Dequeue()$ operations occur) rank r being enqueued in it is $\frac{1}{n}$ (regardless of where the other elements are located), via coupling we can assume that e is enqueued in the queue i, if the label queue i contains the label with rank r. Then, we remove all the extra labels from label queues, that is, if the element with rank r does not exist in the queues, then we remove the label with rank r from the label queues as well. Next, for each $Dequeue()$ operation which chooses queues i and j uniformly at random and proceeds to dequeue from the queue which has the element with the smaller rank (timestamp) on top, we also check the labels on top of label queues i and j and remove the smaller one. Notice that this way, at any point in time, if the element with current rank r is in queue i, then the the label with current rank r is in label queue i as well, and vice versa. This can be formally proved by induction on $Dequeue()$ operations. Here, we switch gears and concentrate on proving rank bounds on the process with labels. The process can be formulated as follows. Let $v_1(t), v_2(t), ..., v_n(t)$ be the labels on top of the label queues after t dequeues have occurred. Initially, we have that $v_i(t) = 0$, for each $1 \le i \le n$. Then at each step $t+1$, we pick two label queues i and j uniformly at random and if w.l.o.g queue i has the smaller label on top, then $v_i(t+1) = v_i(t) + Exp(1/n)$ (for every $k \neq i$, we have that $v_k(t+1) = v_k(t)$). Notice the similarity between this process and Algorithm 1. Let $x_i(t) = \frac{v_i(t)}{n}$ be the weight of bin i at step t. Our initial aim is to upper bound $\Gamma^{con}(t)$ in this case.

Lemma 2.6.1 *Given that $w(t) = \frac{Exp(\frac{1}{n})}{n}$ at every step t, if $\alpha \le \frac{1}{4096Ce^3 \log n}$ and $C \ge 433$, then at any time step t*

$$\mathbb{E}[\Gamma^{con}(t)] \le \frac{146C(\epsilon)n}{\alpha\epsilon}.$$

Proof. Our goal is to apply Lemma 2.5.12 when $w(t) = \frac{Exp(\frac{1}{n})}{n}$ at every step t. For this we will just need to show that Lemma 2.5.11 still holds. The key steps towards accomplishing this are generalizing Lemma 2.5.9 for random weights of mean 1, as opposed to weights of value 1, since we are no longer able to apply Chernoff's inequality and making sure that (2.8) still holds.

First we establish the bounds in (2.1) and (2.2), in order to be able to apply lemmas 2.5.5 and 2.5.4. We know that for each t, $w(t) = \frac{Exp(1/n)}{n}$. Clearly $\mathbb{E}[w(t)] = \frac{\mathbb{E}[Exp(1/n)]}{n} = 1$. In this case the moment generating function is $M(x) = \mathbb{E}[e^{xw(t)}] = \mathbb{E}[e^{\frac{x}{n}Exp(\frac{1}{n})}] = \frac{\frac{1}{n}}{\frac{1}{n} - \frac{x}{n}} = \frac{1}{1-x}$, for $x < 1$. This gives us that $M''(x) = \frac{2}{(1-x)^3}$. Hence, if $\lambda = 1$ and $S = 8$, we have that for every $x < \lambda/2$, $M''(x) < 2S$. This means that to apply Lemma 2.5.3 we will need $\alpha \le \frac{1}{576}$ (which is feasible, since in order to prove Lemma 2.5.12 we need an upper bound on α to be even smaller).

Recall that in Definition 2.5.1 we had that $H_{max}(t) = \max\{H_1(t), H_2(t), ..., H_i(t)\}$, where $H_i(t)$ was the number of times bin i was a random choice made by operations $op_{t-Cn+1}, op_{t-Cn+2}, ..., op_t$ and then we knew that if op_{t+1} was a good operation, the total number of balls thrown into the bin i by the operations which were concurrent with op_{t+1} was at most $H_i(t)$. In order to have the same property in this case, we redefine $H_i(t)$ as follows.

For $0 \le u \le Cn$, let $z(u) = (z_1(u), z_2(u), ..., z_n(u))$ be the n dimensional vector. We assume that $z_i(0) = 0$ for each $1 \le i \le n$. For $0 \le u < n$, consider operation $op_{t-Cn+u+1}$. Let i and j be the random bins it chooses, we know that it increases the weight of the bin, which has the smaller value at the time of performed reads, by $w(t-Cn+u+1)$. We set $z_i(u+1) = z_i(u) + w(t-Cn+u+1)$, if $i \ne j$, we set $z_j(u+1) = z_j(u) + w(t-Cn+u+1)$ and for $k \ne i, j$ we set $z_k(u+1) = z_k(u)$.

Notice that $z_i(Cn)$ is the upper bound on the total weight of the balls thrown into the bin i by operations $op_{t-Cn+1}, op_{t-Cn+2}, ..., op_t$. And thus we can set $H_i(t) = z_i(Cn)$. Hence, our goal is to upper bound $\max\{z_1(Cn), z_2(Cn), ..., z_n(Cn)\}$

We use argument similar to Lemma 2.5.4. Let $\Upsilon_i(u) = e^{\frac{z_i(u)}{8}}$ and $\Upsilon(u) = \sum_{i=1}^n \Upsilon_i(u)$ (Thus, $\Upsilon(0) = n$). If i and j are chosen by operation $op_{t-Cn+u+1}$, then

$$\begin{aligned}
&\mathbb{E}[\Upsilon(u+1)|z(t)] - \Upsilon(u) \\
&\quad \le \mathbb{E}[\Upsilon_i(u+1)|z(t)] - \Upsilon_i(u)] + \mathbb{E}[\Upsilon_j(u+1)|z(t)] - \Upsilon_j(u)] \\
&\quad = \Big(e^{\frac{z_i(u)}{8}} + e^{\frac{z_j(u)}{8}}\Big)\Big(\mathbb{E}\Big[M\Big(\frac{1}{8}\Big)\Big] - 1\Big) \\
&\quad = \Big(e^{\frac{z_i(u)}{8}} + e^{\frac{z_j(u)}{8}}\Big)\Big(\mathbb{E}\Big[M(0) + \frac{M'(0)}{8} + \frac{M''(\xi)}{2\cdot 8^2}\Big] - 1\Big) \\
&\quad = \Big(e^{\frac{z_i(u)}{8}} + e^{\frac{z_j(u)}{8}}\Big)\Big(\mathbb{E}\Big[1 + \frac{w(t-Cn+u+1)}{8} + \frac{M''(\xi)}{2\cdot 8^2}\Big] - 1\Big) \\
&\quad \le \frac{1}{4}\Big(e^{\frac{z_i}{8}} + e^{\frac{z_j}{8}}\Big)
\end{aligned}$$

Where in the the last step we used $0 \le \xi \le \frac{1}{8}$, and as we established above $M''(x) \le 2S = 16$, for each $x \le \frac{1}{2}$. Also, recall that the weight is one in expectation at every step. Hence,

$$\begin{aligned}
\mathbb{E}[\Upsilon(u+1)|z(t)] - \Upsilon(u) &\le \frac{1}{n^2}\sum_{1\le i\le n, 1\le j\le n} \frac{1}{4}\Big(e^{\frac{z_i}{8}} + e^{\frac{z_j}{8}}\Big) \\
&= \frac{2}{n}\sum_{i=1}^n \frac{1}{4} e^{\frac{z_i}{8}} = \frac{\Upsilon(u)}{2n}.
\end{aligned}$$

After removing conditioning we get that

$$\mathbb{E}[\Upsilon(u+1)] \le (1 + \frac{1}{2n})\,\mathbb{E}[\Upsilon(u+1)].$$

After applying the above inequality Cn times we also get that

$$\mathbb{E}[\Upsilon(Cn)] \le n(1 + \frac{1}{2n})^{Cn} \le ne^{C/2}.$$

Finally we proceed as in the proof of Lemma 2.5.13

$$\begin{aligned} Pr[H_{max}(t) > 2KCe^3 \log n] &\leq Pr[\exists i : z_i(Cn) > 2KCe^3 \log n] \\ &\leq Pr[\Upsilon(Cn) > e^{\frac{KCe^3 \log n}{4}}] \\ &\leq \frac{ne^{C/2}}{e^{\frac{KCe^3 \log n}{4}}}. \end{aligned}$$

The last step is to verify that (2.8) is still true. Notice that even though the (2.8) uses $\alpha \leq \frac{1}{4096Ce^3 \log n}$ and $C \geq 2$, the Lemma 2.5.12 requires that $C \geq 433$, and we will take advantage of this upper bound:

$$\begin{aligned} &\sum_{K=1}^{\infty} \mathbb{E}[\Gamma^{con}(t)|A_K(t)]Pr[A_K(t)](e^{2\alpha e^3 C(K+1)\log n} - 1) \\ &\overset{\text{Lemma 2.5.10}}{\leq} \sum_{K=1}^{\infty} \frac{\mathbb{E}[\Gamma^{con}(t)]e^{5\alpha e^3 C(K+1)\log n + \log n + \frac{C}{2}}}{e^{\frac{KCe^3 \log n}{4}}} \\ &\leq \sum_{K=1}^{\infty} \frac{\mathbb{E}[\Gamma^{con}(t)]e^{K + \log n + \frac{C}{2}}}{e^{2KC \log n}}. \end{aligned}$$

We have that $K \geq 1$, $\log n \geq \frac{1}{2}$ (assuming $n \geq 2$), and $C \geq 433$, thus we have that $\frac{C}{2} \leq KC \log n$, $K \leq \frac{KC \log n}{4}$ and $\log n \leq \frac{KC \log n}{4}$. Hence

$$\mathbb{E}[\Gamma^{con}(t)|A_K(t)]Pr[A_K(t)](e^{2\alpha e^3 C(K+1)\log n} - 1) \leq \sum_{K=1}^{\infty} \frac{\mathbb{E}[\Gamma^{con}(t)]}{e^{\frac{KC \log n}{2}}}$$

$$\leq \sum_{K=1}^{\infty} \frac{\mathbb{E}[\Gamma^{con}(t)]}{e^{Kn^{16}}} \leq \frac{\mathbb{E}[\Gamma^{con}(t)]}{2048}.$$

With this in place, we know that 2.5.11 holds if even if $w(t) = \frac{Exp(\frac{1}{n})}{n}$ at every step t and then we can just use Lemma 2.5.12 to finish the proof. □

Now we are ready to upper bound the ranks of dequeued elements.

Theorem 2.6.2 *Assuming an oblivious adversary, the MultiQueue algorithm with parameter n (Algorithm 2) is distributionally linearizable to a sequential randomized relaxed queue Q_R, which ensures that at each step t, the maximum expected rank of dequeued element is $O(n \log^2 n)$, and average expected rank is $O(n \log n \log \log n)$.*

Proof. First, we bound the expected maximum rank of the elements on top. Recall that $x_{max}(t)$ and $x_{min}(t)$ are the largest and smallest weights of bins after t steps and let $v_{max}(t) = nx_{max}(t)$ and $v_{max}(t) = nx_{max}(t)$ be the largest and smallest labels on top of label queues after t dequeue operations. We start by showing that for any $1 \leq i \leq n$,

$$\begin{aligned} \mathbb{E}[rank(v_i(t))] &\leq \sum_{1 \leq j \leq n, j \neq i} (1 + \frac{\mathbb{E}\left[|v_i(t) - v_j(t)|\right]}{n} \\ &= \sum_{1 \leq j \leq n, j \neq i} (1 + \mathbb{E}\left[|x_i(t) - x_j(t)|\right]). \end{aligned} \tag{2.10}$$

The proof is similar to the proof of Lemma 11 in [AKLN17]. We fix (condition on) $v_1(t), v_2(t), ..., v_n(t)$. For each $v_j(t) > v_i(t)$ we know that the label queue j does not contain labels which are smaller than $v_i(t)$, hence the labels in j do not influence the rank of $v_i(t)$. In the case when $v_j(t) < v_i(t)$, we know that the number of labels in j which are smaller than $v_i(t)$ is one plus the number of labels in j which belong to the interval $(v_j(t), v_i(t))$.

We know that the difference between consecutive labels in each label queue is $Exp(\frac{1}{n})$ hence the expected number of labels in j which belong to interval $(v_j(t), v_i(t))$ is upper bounded by $\mathbb{E}[Poi(\frac{v_i(t)-v_j(t)}{n})] = \frac{v_i(t)-v_j(t)}{n}$ (this simply follows from the properties of Exponential and Poisson distributions). Thus, obviously $\mathbb{E}[rank(v_i(t))] \leq \sum_{1\leq j\leq n, j\neq i}(1+\frac{|v_i(t)-v_j(t)|}{n})$ and (2.10) follows after removing conditioning on $v_1(t), v_2(t), ..., v_n(t)$. With this in place, we have that for any $1 \leq i \leq n$

$$\begin{aligned}\mathbb{E}[rank(v_i(t))] &\leq \sum_{1\leq j\leq n, j\neq i}(1+\mathbb{E}\Big[|x_i(t)-x_j(t)|\Big]) \\ &\leq (n-1)+(n-1)\,\mathbb{E}[x_{max}(t)-x_{min}(t)]) \\ &= O(n\log^2 n).\end{aligned}$$

Where the last step follows from the proof of Lemma 2.5.13, which says that both $\mathbb{E}[x_{max}(t)-\mu(t)]$ and $\mathbb{E}[\mu(t)-x_{min}(t)]$ are $O(\log^2 n)$.

Next we aim to upper bound $\sum_{i=1}^n \frac{\mathbb{E}[rank(v_i(t))]}{n}$. This is exactly average expected rank of removed label since during removal we choose both label queues uniformly at random. We have that

$$\begin{aligned}\sum_{i=1}^n \frac{\mathbb{E}[rank(v_i(t))]}{n} &\leq \frac{1}{n}\sum_{i=1}^n \sum_{1\leq j\leq n, j\neq i}(1+\mathbb{E}\Big[|x_i(t)-x_j(t)|\Big]) \\ &\leq n+\frac{1}{n}\sum_{i=1}^n\sum_{j=1}^n \mathbb{E}\Big[|x_i(t)-x_j(t)|\Big] \\ &\leq n+\frac{1}{n}\sum_{i=1}^n\sum_{j=1}^n \mathbb{E}\Big[|x_i(t)-\mu(t)|+|x_j(t)-\mu(t)|\Big] \\ &= n+2\sum_{i=1}^n \mathbb{E}\Big[|x_i(t)-\mu(t)|\Big]. \qquad (2.11)\end{aligned}$$

Using Jensen's inequality we get that

$$\begin{aligned}\frac{\sum_{i=1}^n \alpha|x_i(t)-\mu(t)|}{n} &= \log\left(e^{\frac{\sum_{i=1}^n \alpha|x_i(t)-\mu(t)|}{n}}\right) \\ &\leq \log\left(\frac{\sum_{i=1}^n e^{\alpha|x_i(t)-\mu(t)|}}{n}\right) \leq \log\left(\frac{\Gamma_{con}(t)}{n}\right).\end{aligned}$$

Hence

$$\begin{aligned}\mathbb{E}\left[\frac{\sum_{i=1}^n \alpha|x_i(t)-\mu(t)|}{n}\right] \leq \mathbb{E}\left[\log\left(\frac{\Gamma_{con}(t)}{n}\right)\right] &\overset{Jensen}{\leq} \log\left(\frac{\mathbb{E}[\Gamma_{con}(t)]}{n}\right) \\ &\overset{\text{Lemma 2.6.1}}{\leq} \log\left(\frac{146C(\epsilon)}{\alpha\epsilon}\right) = O(\log\log n).\end{aligned}$$

Where in the last step we used $\alpha = \Theta(\frac{1}{\log n})$ and for the same reason we get that $\sum_{i=1}^n |x_i(t)-\mu(t)| = O(n\log n\log\log n)$. Finally, (2.11) gives us that

$$\sum_{i=1}^n \frac{\mathbb{E}[rank(v_i(t))]}{n} \leq n+2O(n\log n\log\log n) = O(n\log n\log\log n).$$

□

We can also also show the high probability bound on the maximum rank:

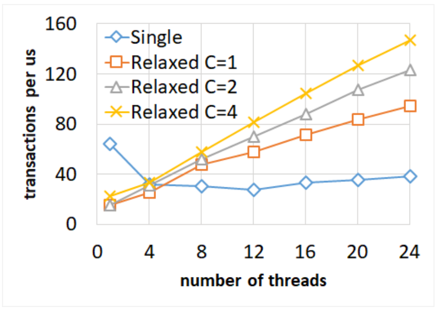
Corollary 2.6.3 *At any step t and large enough R, probability of rank of returned element being larger than $2Rn\log^2 n$ is at most $2n^{-\Omega(R)}$.*

Proof. Note that Lemma 2.5.13 shows that for large enough R, $Pr[x_{max}(t) - x_{min}(t) \geq R\log^2 n] \leq n^{\Omega(-R)}$. This holds in the case of queues as well. Also, notice that if $x_{max}(t) - x_{min}(t) \leq R\log^2 n$, then the maximum rank is at most $\sum_{i=1}^{n} Poi(\frac{Rnlog^2 n}{n}) = Poi(Rn\log^2 n)$ (Recall that the sum of n independent Poisson random variables is also Poisson). Hence by the tail bounds of Poisson distribution probability of maximum rank being larger than $2Rn\log^2 n$ is also at most $n^{\Omega(-R)}$. Thus, by the law of total probability the probability of maximum rank being larger than $Rn\log^2 n$ is at most $2Rn^{\Omega(-R)}$. □

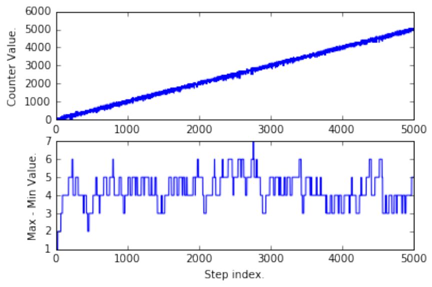
2.7 Experimental Results

Setup. Our experiments were run on an Intel E7-4830 v3 with 12 cores per socket and 2 hyperthreads (HTs) per core, for a total of 24 threads, and 128GB of RAM. In all of our experiments, we pinned threads to avoid unnecessary context switches. Hyperthreading is only used with more than 12 threads. The machine runs Ubuntu 14.04 LTS. All code was compiled with the GNU C++ compiler (G++) 6.3.0 with compilation options `-std=c++11 -mcx16 -O3`.

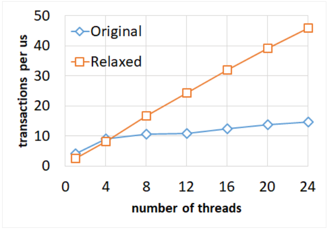
Synthetic Benchmarks. We implemented and benchmarked the MultiCounter algorithm on a multicore machine. To test the behavior under contention, threads continually increment the counter value using the two-choice process. We use no synchronization other than the atomic fetch and increment instruction for the update. Figure 2.1a shows the scalability results, while Figure 2.1b shows the "quality" guarantees of the implementation in terms of values returned by the counter over time, as well as maximum gap between bins over time. Quality is measured in a single-threaded execution, for 64 counters. (Recording quality accurately in a concurrent execution appears complicated, as it is not clear how to order the concurrent read steps.)



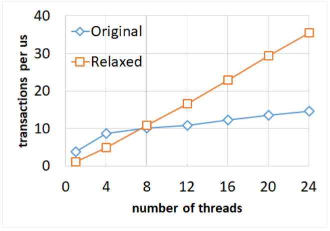
(a) Scalability of the concurrent counter for different values of the ratio C between counters and # threads.



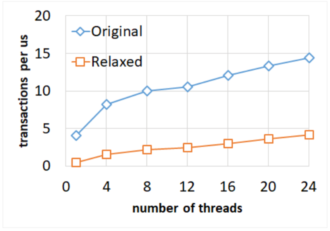
(b) Quality results for the concurrent counter in a single-threaded execution. The x axis is # increments.



(c) TL2 benchmark, 1M objects.



(d) TL2 benchmark, 100K objects.



(e) QTL2 benchmark, 10K objects.

Figure 2.1: Experimental Results for the Concurrent Counter.

TL2 Benchmark. Transactional Locking II (TL2) is a software implementation of transactional memory introduced by [DSS06]. TL2 guarantees opacity by using fine-grained locking and a global clock G. TL2 associates a *version lock* with each memory location. A version lock behaves like a traditional lock, except it additionally stores a version number that represents the value of G when the memory location protected by the lock was last modified. At a high level, a transaction starts by reading G, and uses the clock value it reads to determine whether it ever observes the effects of an uncommitted transaction. If so, the transaction will abort. Otherwise, after performing all of its reads, it locks the addresses in its write set (validating these locations to ensure that they have not been written recently), rereads G to obtain a new version v', performs its writes, then releases its locks, updating their versions to v'.

TL2 with Relaxed Global Clocks. In the standard implementation of TL2, G is incremented using fetch-and-add (FAA). This quickly becomes a concurrency bottleneck as the number of threads increases, so the the authors developed several improved implementations of G. However, they too experience scaling problems at large thread counts. We replace this global clock counter G with a MultiCounter implementation, and compare against a highly-optimized baseline implementation.

Due to the fact that the counter is relaxed, reasoning about the correctness of the resulting algorithm is no longer straightforward. In particular, a key property we need to enforce is that the timestamp which a thread writes to a set of objects as part of its transaction (generated when the thread is holding locks to commit and written to all objects in its write set) cannot be held by any other threads at the same time, since such threads might read those concurrent updates concurrently, and believe that they occurred in the past. For this reason, we modify the TL2 algorithm so that threads write “in the future," by adding a quantity Δ, which exceeds the maximum clock *skew* we expect to encounter in the MultiCounter over an execution, to the maximum timestamp $t_{\max}$ they have encountered during their execution so far. Thus, each new write always increments an object's timestamp by $\geq \Delta$. We stress that that the (approximate) global clock is implemented by the MultiCounter algorithm, and that it is disjoint from the object timestamps.

This protocol induces the following trade-offs. First, the resulting transactional algorithm only ensures safety with high probability, since the Δ bound might be broken at some point during the execution, and lead to a non-serializable transaction, with extremely low probability. Second, we note that, once an object is written with a timestamp that occurs in the future, transactions which immediately read this object may abort, since they see a timestamp that is larger than theirs. Hence, once an object is written, at least Δ operations should occur *without accessing this object*, so that the system clock is incremented past the read point without causing readers to abort. Intuitively, this upper bounds the frequency at which objects should be written to for this approximate timestamping mechanism to be efficient. On the positive side, this mechanism allows us to break the scalability bottleneck caused by the global clock.

We verify this intuition through implementation. See Figures 2.1c—2.1e. We are given an array of n transactional objects, with n between 10K and 1M. Transactions pick 2 array locations uniformly at random, then start a transaction, increment both locations, and then commit the transaction. We record the average throughput out of ten one-second experiments. We verify correctness by checking that the array contents are consistent with the number of executed operations at the end of the run; none of these experiments have resulted in erroneous outputs. We record the rate at which transactions commit, as a function of the number of threads. We note that, for 1M and 100K objects, the average frequency at which each location is written is below the heavy abort threshold, and we obtain almost linear scaling with MultiCounters. At 10K objects we surpass this threshold, and see a considerable drop in performance, because of a large number of aborts.

2.8 Related Work

Randomized Load Balancing. The classic two-choice balanced allocation process was introduced in [ABKU99], where the authors show that, under two-choice insertion, the most loaded among n bins is at most $O(\log \log n)$ above the average, both in expectation and with high probability. The literature studying analyses and extensions of this process is extremely vast, hence we direct the reader to [RMS01, Mit96] for in-depth surveys of these techniques. Considerable effort has been dedicated to understanding guarantees in the “heavily-loaded" case, where the number of insertion steps is unbounded [BCSV00, PTW15], and in the “weighted" case, in which ball weights come from a probability distribution [TW07, BFHM08]. A tour-de-force by Peres, Talwar, and Wieder [PTW15] gave a potential argument characterizing a general form of the heavily-loaded, weighted process on *graphs*. Our analysis starts from their framework, and modifies it to analyze a concurrent, adversarial process. One significant change from their analysis is that, due to the adversary, changes in the potential are only partly stochastic: most steps might be slightly biased away from the better of the two choices, while a subset of choices might be almost deterministically biased towards the *wrong* choice. Further, the adversary can decide the *order* in which these different steps, with different biases, occur. Recently, [LS21] studied heavily-loaded, sequential two-choice process with incomplete information. In their model, at each step, two bins are chosen uniformly at random, but it is not possible to directly compare their loads. Instead, the choice has to be made by estimating the loads of the bins using $\Theta(\log \log n)$ binary queries, of type “Is the load at least the load of the second most loaded bin?” or “Is the load at least 128?” The authors show that, even in this case, the gap between the most loaded bin and the average is at most $O(\log \log n)$, with high probability. Even though their analysis deals with the *wrong* choices made due to incomplete information, the approach is different from ours, since we consider an asynchronous setting.

Lenzen and Wattenhofer [LW16] analyzed *parallel* balls-into-bins processes, in which n balls need to be distributed among n bins, under a communication model between the balls and the bins, showing that almost-perfect allocation can be achieved in $O(\log^* n)$ rounds of communication. This setting is quite different from the one we consider here. Similar delayed information models, where outdated information is given to the insertion process were considered by Mitzenmacher [Mit00] and by Berenbrink, Czumaj, Englert, Fridetzky, and Nagel [BCE+12]. The former reference proposes a bulletin board model with periodic updates, in which information about the load of the model is updated only periodically (every T seconds), and various allocation mechanisms. The author provides an analysis of this process in the asymptotic case (as $n \to \infty$), supported by simulations. The latter reference [BCE+12] considers a similar model where balls arrive in *batches*, and must perform allocations collectively based solely on the information available at the beginning of the batch, without additional communication. The authors prove that the greedy multiple-choice process preserves its strong load balancing properties in this setting: in particular, the gap between min and max remains $O(\log n)$. The key difference between these models and the one we consider is that our model is completely asynchronous, and in fact the interleavings are chosen adversarially. The technique we employ is fundamentally different from those of [Mit00, BCE+12]. In particular, our techniques can be applied to the setting considered in [BCE+12], albeit the resulting upper bound on the gap bound will be $O(\log^2 n)$.

In [AKLN17], authors analyzed the following producer-consumer process: a set of balls labelled $1, 2, \ldots, b$ are inserted sequentially at random into n bins; in parallel, balls are removed from the bins by always picking the lower labelled (higher priority) of two uniform random choices.[3] This process sequentially models a series of popular implementations of concurrent priority queue data structures, e.g. [RSD15, HLH+13]. This process provides the following guarantees: in each step t, the expected *rank* of the label removed among labels still present in the system is $O(n)$, and $O(n \log n)$ with high probability in n. That is, this sequential process provides a structured probabilistic relaxation of a standard priority queue.

Relaxed Data Structures. The process considered in [AKLN17] is sequential, whereas the data structures implemented are concurrent. Thus, there was a significant gap between the theoretical guarantees and the practical implementation. Our current work extends to concurrent data structures, closing this gap. Under the oblivious adversary assumption and given our parametrization, we show for the first time that practical data structures such as [RSD15, HLH+13, AKLN17] provide guarantees in real executions.

Designing efficient concurrent/parallel data structures with relaxed semantics was initiated by Karp and Zhang [KZ93], with other significant early work by Deo and Prasad [DP92] and Sanders [San98]. It has recently become an extremely active research area, see e.g. [SL00, BFK+11, WGTT15, AKLS15, HLH+13, NLP13, RSD15, AKLN17, RAT] for recent examples. To the best of our knowledge, ours is the first analysis of randomized relaxed concurrent data structures which works under arbitrary oblivious schedulers: previous analyses such as [AKLS15, RSD15, AKLN17] required strong assumptions on the set of allowable interleavings. Dice et al. [DLM13] considered randomized data structures for scalable exact and approximate counting. They consider the efficient parallelization of sequential approximate counting methods, and therefore have a significantly different focus than our work.

[3]Balls in each bin are sorted in increasing order of label, i.e. each bin corresponds to a sequential priority queue.

CHAPTER 3

Applications of Relaxed Scheduler

3.1 Introduction

Several classic problems in graph processing and computational geometry can be solved *incrementally*: algorithms are structured as a series of *tasks*, each of which examines a subset of the algorithm state, performs some computation, and then updates the state. For instance, in Dijkstra's classic graph single-source shortest paths (SSSP) algorithm [Dij59], the state consists of the current distance estimates for each node in the graph, and each task corresponds to a node "relaxation," which may update the distance estimates of the node's neighbors. In the case of the classic sequential variant, the order in which tasks get executed is dictated by the sequence of node distances. At the same time, many other incremental algorithms, such as Delaunay mesh triangulation, assume arbitrary (or random) orders on the tasks to be executed, and can even provide efficiency guarantees under such orderings [BGSS16].

A significant amount of attention has been given to *parallelizing* such incremental iterative algorithms, e.g. [GLG$^+$12, NLP13, BGSS16, DBS17, DBS21]. One approach has been to study the dependence structure of such algorithms, proving that, in many cases, the dependency chains are *shallow*. This can be intuitively interpreted as proving that such algorithms should have significant levels of parallelism. One way to exploit this fine-grained parallelism, e.g. [BFS12, SBFG13] has been to carefully split the execution into task prefixes of limited length, and to parallelize the execution of each prefix efficiently. While this approach can be efficient, it does require an understanding of the workload and task structure, and may not be immediately applicable to algorithms where the task ordering is dependent on the input.

An alternative approach has been to employ scalable data structures which only ensure *relaxed priority order* to schedule task-based programs. The idea can be traced back to Karp and Zhang [KZ93], who studied parallel backtracking in the PRAM model, and noticed that, in some cases, the scheduler can relax the strict order induced by the sequential algorithm, allowing tasks to be processed speculatively ahead of their dependencies, without loss of correctness. For instance, when parallelizing SSSP, e.g. [AKLS15, SW16, NLP13], the scheduler may retrieve vertices in arbitrary order without breaking correctness, as the distance at each vertex is guaranteed to eventually converge to the minimum. However, there is intuitively a trade-off between the performance gains arising from using scalable relaxed schedulers, and the loss of determinism and the possible wasted work due to having to re-execute speculative tasks.

This approach is quite popular in practice, as several efficient relaxed schedulers have been proposed [SL00, BFK$^+$11, WGTT15, AKLS15, HLH$^+$13, NLP13, RSD15, AKLN17, SW17], which can attain state-of-the-art results for graph processing and machine learning [NLP13, GLG$^+$12], and even have hardware implementations [JSY$^+$16]. At the same time, despite showing good empirical performance, this approach does not come with analytical bounds: in particular, for most known algorithms, it is not clear how the relaxation factor in the scheduler affects the total work performed by the parallel algorithm.

We address this question in this chapter. Roughly, we show under analytic assumptions that, for a set of fundamental algorithms including parallel Dijkstra's and Delaunay mesh triangulation, the extra work engendered by scheduler relaxation can be negligible with respect to the total number of tasks executed by the sequential algorithm. On the negative side, we show that relaxation does not come for free: we can construct worst-case instances where the cost of relaxation is asymptotically non-negligible, even for relatively benign relaxed schedulers.

We model the relaxed execution of incremental algorithms as follows. The *algorithm* is specified as an ordered sequence of tasks, which may or may not have precedence constraints. The algorithm's execution is modeled as an interaction between a *processor*, which can execute tasks, modify state, and possibly create new tasks, and a *scheduler*, which stores the tasks in a priority order specified by the algorithm. At each step, the processor requests a new task from the scheduler, examines whether the task can be processed (i.e., that all precedence constraints are satisfied), and then executes the task, possibly modifying the state and inserting new tasks as a consequence.

An exact scheduler would return tasks following priority order. Since ensuring such strict order semantics is known to lead to contention and poor performance [AACH$^+$14], practical scalable schedulers often *relax* the priority order in which tasks are returned, up to some constraints. For generality, in this chapter, we assume when proving performance upper bounds that the scheduler may in fact be *adversarial*—actively trying to get the algorithm to waste steps, up to some natural *rank inversion* and *fairness* constraints. Specifically, the two natural constraints we enforce on the scheduler are on 1) the *maximum rank inversion* between the highest priority task present and the rank of the task returned, and on 2) *fairness*, in terms of the maximum number of schedule steps for which the task of highest priority may remain unscheduled. For convenience, we upper bound both these quantities by a parameter k, which we call the *relaxation factor* of the scheduler. Simply put, a k-relaxed scheduler must 1) return one of the k highest-priority elements in every step; and 2) return a task at the latest k steps after it has become the highest-priority task available to the scheduler. We note that real schedulers enforce such constraints either deterministically [WGTT15] or with high probability [AKLN17, ABK$^+$18, ABKN18].

A significant limitation of the above model is that it is *sequential*, as it assumes a single processor which may execute tasks, but we believe that it provides good intuition about why relaxed schedulers are effective.

It is natural to ask whether incremental algorithms can still provide any guarantess on total work performed under $k-$relaxed schedulers. Additional work may arise due to relaxation for two reasons. A first cause for wasted work is if a task may need to be re-executed once the state is updated, this is the case when running parallel SSSP: due to relaxation, a node may be speculatively relaxed at a distance that is higher than its optimal distance from the source, leading it to be relaxed several times. The second is if the parallel execution enforces ordering constraints between data-dependent tasks: for instance, when executing a graph algorithm, the task corresponding to a node u may need to be processed *before* the task corresponding to any neighbor which has higher priority in the initial node ordering.

We note that neither phenomenon may occur when the priority order is strict—since the top priority task cannot have preceding constraints nor need to be re-executed—but are inherent in parallel executions.

A trivial upper bound on wasted work for an algorithm with total work W under a k-relaxed scheduler would be $O(kW)$—intuitively, in the worst case the scheduler may return k tasks before the top priority one, which can always be executed without violating constraints. The key observation we make in this work is that, in the case of SSSP the extra work depends on the graph structure. Additionally, because of their local dependency structure, some popular incremental algorithms will incur *significantly less* overhead due to out-of-order execution.

For SSSP, which does not have a dependency structure but may have to re-execute tasks, we use an approach, based on Δ-stepping, [MS03]. We bound the total overhead of relaxation to $O(\mathrm{poly}(k)\, d_{\max}/w_{\min})$, where $d_{\max}$ is the maximum shortest distance from the source to some other node, and $w_{\min}$ is the minimum edge weight. While this overhead may in theory be arbitrarily large, depending on the input, we note that for many graph models, this overhead is small. (For example, for Erdös-Renyi graphs with constant weights, the overhead is $O(\mathrm{poly}\, k \log n)$.)

For incremental algorithms, such as Delaunay mesh triangulation and sorting by insertion, we show that the expected overhead of execution via a k-relaxed scheduler is $O(\mathrm{poly}\, k \log n)$, where n is the number of tasks the sequential variant of the algorithm executes. We exploit the following properties of incremental algorithms, shown in [BGSS16]: The probability that the task at position j is dependent on the task at position $i < j$ depends only on the tasks at positions $1, 2, ..., i$ and j, and assuming a random order of tasks, this probability is upper bounded by $O(1/i)$. While the technical details are not immediate, the argument boils down to bounding, for each top-priority task, the number of *dependent* tasks which may be returned by the scheduler while the task is still in the scheduler queue.

It is interesting to interpret these overheads in the context of practical concurrent schedulers such as MultiQueues [RSD15, GLG+12], where the relaxation factor k is proportional to the number of concurrent processors p, up to logarithmic factors. Since in most instances the size of the number of tasks n is significantly larger than the number of available processors p, the overhead of relaxation can be seen to be comparatively small. This observation has been already made empirically for specific instances, e.g. [LNP15]; our analysis formalizes this observation in our model.

On the negative side, we also show that the overhead of relaxation is non-negligible in some instances. Specifically, we exhibit instances of incremental algorithms where the overhead of relaxed execution is $\Omega(\log n)$. Interestingly, this lower bound does not require the scheduler to be adversarial: we show that it holds even in the case of the relatively benign MultiQueue scheduler [RSD15, AKLN17].

3.2 Relaxed Schedulers: The Sequential Model

We begin by formally introducing our sequential model of relaxed priority schedulers. We represent a priority scheduler as a *relaxed ordered set* data structure Q_k, where the integer parameter k is the *relaxation factor*. A relaxed priority scheduler contains $< task, priority >$ pairs and supports the following operations:

1. $Q_k.Empty()$, returns `true` if Q_k is empty, and `false` otherwise.

2. $ApproxGetMin()$, returns a $< task, priority >$ pair if one is available, without deleting it.

3. $DeleteTask(task)$, removes specified task from the scheduler. This is used to remove a task returned by $ApproxGetMin()$, if applicable.

4. $Insert(< task, priority >)$, inserts a new task-priority pair in Q_k.

We denote the rank (in Q_k) of the task returned by the t-th $ApproxGetMin()$ operation by $rank(t)$, and call it the rank of a task returned on step t. For a task u, let $inv(u)$ be the number of inversions experienced by task u between the step when u becomes the highest priority task in Q_k and the step when task u is returned by the scheduler. That is, $inv(u) + 1$ is the number of $ApproxGetMin()$ operations needed for the highest priority task u to be scheduled.

Rank and Fairness Properties. The relaxed priority schedulers Q_k we consider will enforce the following properties:

1. $RankBound$: at any time step t, $rank(t) \leq k$.

2. $Fairness$: for any task u, $inv(u) \leq k - 1$.

Priority schedulers such as k-LSM [WGTT15] enforce these properties deterministically, where k is a tunable parameter. In the case of MultiQueues [RSD15] with q queues, Corollary 2.6.3 shows that for large enough constant R, rank bound holds for $k = 2Rq\log q$ with probability at least $1 - 2q^{-\Omega(R)}$. On the other hand, MultiQueue returns the task with the highest priority with probability at least $\frac{1}{q}$, hence with probability at least $1 - q^{-2R}$, $inv(u) \leq 2Rq\log q$, for any fixed task u.

The above probabilities are calculated for a fixed task and fixed step. Let s be the total number of times $DeleteTask()$ operation removes the task with the highest priority during the entire time the scheduler is operating. Note that for the algorithms we consider in this chapter, $s \leq n$ (n is the total number of tasks used by the algorithm, and hence available to the scheduler), since each task becomes the task with the highest priority at most once during the entire run of the algorithm. Using union bound we can show that with probability at least $1 - nq^{-2R}$, the $Fairness$ holds during the entire operation of the scheduler. Conditioned on this event, we have that the total number of times $DeleteTask()$ task operation is invoked is at most $2Rnq\log q$, hence by again using union bound we get that with probability at least $1 - 4Rnq^{-\Omega(R)}q\log q$, the $RankBound$ holds during the entire operation of the scheduler as well. Finally, by using the law of total probability and setting $R = q = \Theta(\log n)$ we get that with probability at least $1 - 4Rnq^{-2R+1}\log q - nq^{-2R} = 1 - O(n^{\Omega(-R)})$, the $RankBound$ and $FairNess$ hold for $k = 2Rq\log q = O(R\log n \log\log n) = O(\text{poly}(\log n))$ during the entire time the scheduler is operating.

We proceed by showing how the usage of relaxed scheduler affects the SSSP algorithm.

3.3 Analyzing SSSP under Relaxed Scheduling

Preliminaries. Since the algorithm is different from the ones we considered thus far, we re-introduce some notation. We assume we are given a directed graph $G = (V, E)$ with positive edge weights $w(e)$ for each edge $e \in E$, and a source vertex s. For each vertex $v \in V$, let $d(v)$ be the weight of a shortest path from s to v. Additionally, let $d_{max} = \max\{d(v) : v \in V\}$ and $w_{min} = \min\{w(e) : e \in E\}$.

We consider the sequential pseudocode from Algorithm3, which uses a relaxed priority queue Q_k to find shortest paths from s via a procedure similar to the Δ-stepping algorithm [MS03].

In this algorithm $Q_k.push(v, dist)$ inserts a vertex v with distance $dist$ in Q_k, $Q_k.pop()$ removes and returns a vertex, distance pair $(v, dist)$, such that v is among the k smallest distance vertices in Q_k. We also assume that Q_k supports a $Q_k.DecreaseKey(v, dist)$ operation, which atomically decreases the distance of vertex v in Q_k to $dist$.

Algorithm 3 SSSP algorithm based on a relaxed priority queue.

```
Data: Graph $G = (V, E)$, source vertex $s$.
      Initially empty relaxed priority queue $Q_k$.
      Array $dist[n]$ for tentative distances.
for each vertex $v \in V$ do
    $dist[v] \leftarrow +\infty$
end for
$dist[s] \leftarrow 0$
$Q_k.push(s, 0)$
while !$Q_k.empty()$ do
    $(v, curDist) \leftarrow Q_k.pop()$
    for $u : (v, u) \in E$ do
        $e \leftarrow (v, u)$
        if $dist[u] > curDist + w(e)$ then
            $dist[u] \leftarrow curDist + w(e)$
            % We assume that we can check whether $u$ is in $Q_k$.
            % The check can be implemented via
            % maintaining the corresponding flag for each vertex.
            if $u \in Q_k$ then
                $Q_k.DecreaseKey(u, dist[u])$
            else
                $Q_k.Add(u, dist[u])$
            end if
        end if
    end for
end while
```

Analysis. We will prove the following statement, which upper bounds the extra work incurred by the relaxed scheduler:

Theorem 3.3.1 *The number of $Q_k.pop()$ operations performed by Algorithm 3 is*

$$O(k^2 d_{\max}/w_{\min}) + n. \tag{3.1}$$

Proof. Our analysis will follow the general pattern of Δ-stepping analysis. We will partition the vertex set V into buckets, based on distance: vertex v belongs to bucket B_i iff $d(v) \in [iw_{min}, (i+1)w_{min})$. Let $t = d_{\max}/w_{\min}$ be the total number of buckets we need (for simplicity we assume that $d_{\max}/w_{\min}$ is an integer).

Observe that because of the way we defined buckets, we have the following property, which we will call *the bucket property* : for any vertex $v \in V$, no shortest path from s to v contains vertices which belong to the same bucket.

We say that Algorithm 3 processes vertex v *at the correct distance* if $Q_k.pop()$ returns $(v, d(v))$, this means that $dist[v] = d(v)$ at this point and we relax outgoing edges of v. (See Algorithm 3 for clarification.)

We fix $i < t$ and look at what happens when Algorithm 3 processes all vertices in the buckets $B_0, B_1, ..., B_i$ at the correct distance. Because of the bucket property, we get that $d(u) = dist[u]$ for every $u \in B_{i+1}$, and the vertices from bucket B_{i+1} are either ready to be processed at the correct distance, or are already processed at the correct distance. To avoid the second case, we also assume that if $Q_k.pop()$ returns $(u, d(u))$, where $u \in B_{i+1}$ and not all vertices in the buckets $B_0, B_1, ..., B_i$ are processed at the correct distance, then this $Q_k.pop()$ operation still counts towards the total number of $Q_k.pop()$ operations, but it does not actually remove the task and does not perform edge relaxations, even though u is ready to be processed at the correct distance. This assumption only increases the total number of $Q_k.pop()$ operations, so to prove the claim it suffices to derive an upper bound for this pessimistic case.

Once the algorithm processes the vertices in buckets $B_0, B_1, ..., B_i$ at the correct distances, we know that the only vertices with tentative distance less than $(i+2)w_{min}$ are the vertices in the bucket B_{i+1}. (Note that this statement would not hold if we didn't have the $DecreaseKey$ operation: if we insert multiple copies of vertices in Q_k with different distances, as in some versions of Dijkstra, there might exist outdated copies of vertex $u \in B_j, j < i$, even though u was already processed at the correct distance.) This means that, at this point, the top $|B_{i+1}|$ vertices (vertices with the smallest distance estimates) belong to B_{i+1}.

Next, we bound how many $Q_k.pop()$ operations are needed to process the vertices in B_{i+1}, after all vertices in the buckets $B_0, B_1, ..., B_i$ are processed. If $|B_{i+1}| > k$, using the rank property, we have that the first $(|B_{i+1}| - k)$ $Q.pop()$ operations process vertices in B_{i+1}. If $|B_{i+1}| \leq k$, we know that it will take at most k^2 $Q_k.pop()$ operations to process all vertices in B_{i+1}, since, because by the fairness bound, the number of $Q_k.pop()$ operations to return the top vertex (the one with the smallest tentative distance) is at most k, and we showed that the top vertex belongs to B_{i+1} until all vertices in B_{i+1} are processed. By combining these two cases, we get that the number of $Q_k.pop()$ operations to process vertices in B_{i+1} at the correct distance is at most $|B_{i+1}| + k^2$.

Thus the number of $Q_k.pop()$ operations performed by Algorithm 3 in total is at most:

$$\sum_{i=0}^{t}(k^2 + |B_i|) = n + O(k^2 d_{\max}/w_{\min}), \tag{3.2}$$

as claimed.

□

Lower Bound. Note that for some graphs, the above result is tight up to a k factor, in the worst case. For example, consider a clique graph with a vertex set $\{1, 2..., n\}$. Let the weight

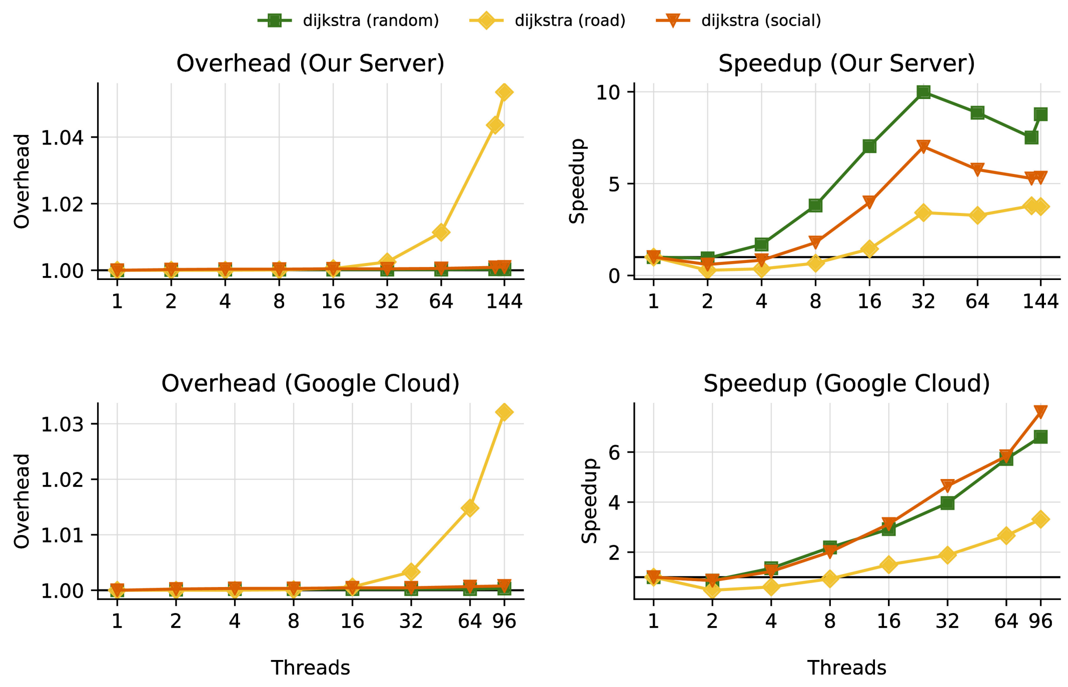


Figure 3.1: Overheads (left) and speedups (right) for parallel SSSP Dijkstra's algorithm executed via a MultiQueue relaxed scheduler on random, road network, and social network graphs. The overhead is measured as the ratio between the number of tasks executed via a relaxed scheduler versus an exact one.

of the edge from vertex i to vertex j be 1, if $j - i = 1$ and n otherwise. Let vertex 1 be the source vertex. We have that $w_{min} = 1$ and $d_{max} = n - 1$, and it is easy to see that in the worst case the number of $Q_k.pop()$ performed by Algorithm 3 is $\Omega(nk) = \Omega(kd_{max}/w_{min})$.

Discussion. A clear limitation is that the bound depends on the maximum distance $d_{\max}$, and on the minimum weight $w_{\min}$. Hence, this bound would be relevant only for low-diameter graphs with bounded edge weights. We note however this case is relatively common: for instance, [DBS17] considers weighted graph models of low diameter, where weights are chosen in the interval $[1, \log n)$. These assumptions appear to hold in for many publicly available weighted graphs [LK14]. Further, our argument assumes a relaxed scheduler supporting $DecreaseKey$ operations. This operation can be supported by schedulers such as the SprayList [AKLS15] or MultiQueues [RSD15, AKLN17] where elements are hashed consistently into the priority queues.

3.4 Experiments

We implemented the parallel SSSP Dijkstra's algorithm described in Section 3.3 using an instance of the MultiQueue relaxed priority scheduler [RSD15, AKLN17]. In the classic sequential algorithm nodes are processed sequentially, while in this parallel version a node can be processed several times due to out-of-order execution. In our experiments, we are interested in the total number of tasks processed by the concurrent variant, in order to examine the overhead of relaxation in concurrent executions. In addition, we also measure execution times for increasing number of threads. *Overhead* is measured as the average number of tasks

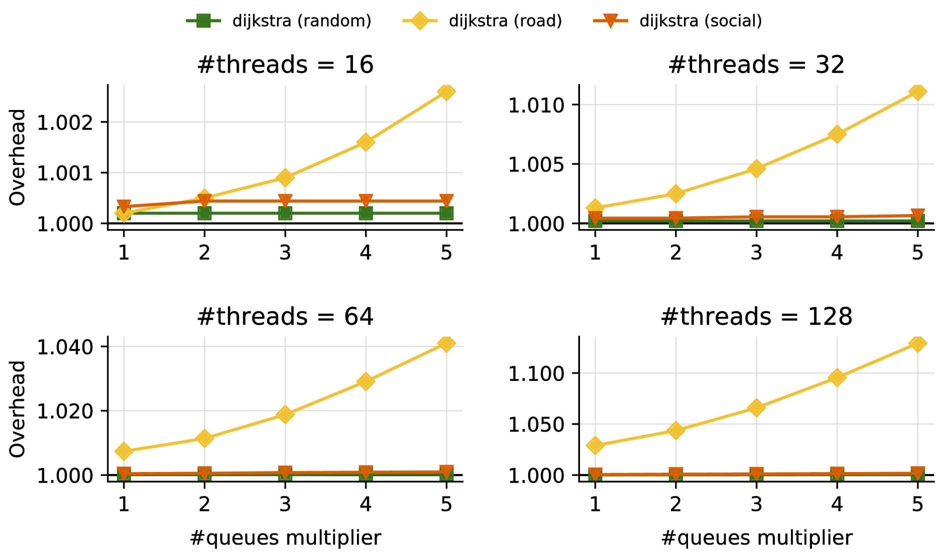


Figure 3.2: Relaxation overheads versus relaxation factor/queue multiplier for parallel SSSP Dijkstra's algorithm. The number of queues is the multiplier (x axis) times the number of threads, and is proportional to the average relaxation factor of the queue [AKLN17].

executed in a concurrent execution divided by the number of tasks executed in a sequential execution using an exact scheduler.

Sample graphs. We use the following list of graphs in our experiments:

- Random undirected graph with 1 million nodes and 10 million edges, with uniform random weights between 0 and 100 (**random**);
- USA road network graph with physical distances as edge lengths; ~ 24 million nodes and ~ 58 million edges (**road**) [DGJ09];
- LiveJournal social network friendship graph; ~ 5 million nodes and ~ 69 million edges, with uniform random weights between 0 and 100 (**social**) [LK14].

Platforms. We evaluated the experiment on a server with 4 Intel Xeon Gold 6150 (Skylake) sockets. Each socket has 18 2.70 GHz cores, each of which multiplexes 2 hardware threads, for a total of 144 hardware threads. In addition, we ran the experiment on a Google Cloud Platform VM supporting to 96 hardware threads.

Experimental results. The experimental results are summarized in Figure 3.1. On the left column, notice that, on both machines, the overheads of relaxation are almost negligible: for the random graph and the social network, the overheads are almost 1% at all thread counts, what practically means the absence of extra work. (Recall that the number of queues is always $2\times$ the number of threads, so the relaxation factor increases with the thread count.)

The road network incurs higher overheads (5% at 144 threads / 288 queues). This can be explained by the higher diameter of the graph (6261, versus 16 for the LiveJournal and 6 for the random graphs), and by the higher variance in edge costs for the road network. In terms of speedup (right), our implementation scales well for 1-2 sockets on our local server, after which NUMA effects become prevalent. NUMA effects are less prevalent on the Google Cloud machine, but the maximum speedup is also more limited ($< 7\times$ instead of $10\times$).

In Figure 3.2, we examine the relaxation overhead (in terms of the amount of extra tasks executed) versus the relaxation factor. While we cannot control the relaxation factor exactly, we know that the average value of this factor is proportional to the number of queues allocated, which is the number of threads (fixed for each sub-plot) times the multiplier for the number of queues (the x axis) [AKLN17]. We notice that these overheads are only non-negligible for the road network graph. On the one hand, this suggests that our worst-case analysis is not tight, but can also be interpreted as showing that the overheads of relaxation do become apparent on dense, high-diameter graphs such as road networks.

Next, we describe how incremental algorithms can be implemented using the relaxed scheduler.

3.5 Incremental Algorithms

3.5.1 General Definitions

We assume a model of incremental algorithms which execute a set of tasks iteratively, one by one, and where each task incrementally updates the algorithm's state. For example, in incremental graph algorithms, the shared state corresponds to a data structure storing the graph nodes, edges, and meta-data corresponding to nodes. Tasks usually correspond to vertex operations, and are usually inserted and executed in some order, given by the input. If this task order is random, we say that the incremental algorithm is randomized. We will consider both randomized incremental algorithms, where each task has a priority based on the random order, and deterministic ones, where the order is fixed. Using an exact scheduler corresponds to executing tasks in the same order as the sequential algorithm, while using a relaxed scheduler allows out-of-order execution of tasks.

Definition. More formally, randomized incremental algorithms such as Delaunay triangulation and comparison sorting with via BST insertion can be modelled as follows:

We are given n tasks, which must be executed iteratively in some (possibly random) order. Initially, each task u is assigned a unique label $\ell(u)$. For instance, this label can be based on a random permutation of n given tasks, π. That is, for task u, $\ell(u) = i$, iff $\pi(i) = u$. A lower label can be equated with higher priority. Each task performs some computation and updates the algorithm state. In the case of Delaunay triangulation, tasks update the triangle mesh, while in the case of Comparison Sorting tasks modify the BST accordingly. Generic sequential pseudocode is given in Algorithm 4. We note that a similar generic algorithm was presented in [ABKN18] for parallelizing greedy iterative algorithms.

When using a relaxed priority Q_k instead of an exact priority queue Q, one issue is the presence of inter-task dependencies. These dependencies are specified by the algorithm, and are affected by the permutation of the tasks.

If task v depends on task u and $\ell(u) < \ell(v)$, then task v can not be processed before task u. We call task u an *ancestor* of task v in this case. We assume that the task returned by the relaxed scheduler can be processed only if all of its ancestors are already processed.

Pseudocode is given in Algorithm 5.

Observe that the For loop runs for exactly n steps in the exact case, but it may require extra steps in the relaxed case. We are interested in upper bounding the number of extra steps,

Algorithm 4 General Framework for incremental algorithms.

```
Data: Sequence of tasks $V = (v_1, v_2, ..., v_n)$, in decreasing priority order.
% $Q$ is an exact priority queue.
$Q \leftarrow$ tasks of $V$ with priorities
for each step $t$ do
    % remove the task with highest priority
    $v_t \leftarrow Q.DeleteMin()$
    $Process(v_t)$
    % stop if $Q$ is empty.
    if $Q.empty()$ then
        break
    end if
end for
```

Algorithm 5 General Framework for executing incremental algorithms using relaxed priority schedulers.

```
Data: Sequence of tasks $V = (v_1, v_2, ..., v_n)$, in decreasing priority order.
% $Q_k$ is a relaxed priority queue.
$Q_k \leftarrow$ tasks of $V$ with priorities
for each step $t$ do
    % get the task with highest priority from $Q_k$.
    $v_t \leftarrow Q_k.GetMin()$
    % check if $v_t$ has no dependencies.
    if $CheckDependencies(v_t)$ then
        $Q_k.Delete(v_t)$
        $Process(v_t)$
    end if
    % stop if $Q_k$ is empty.
    if $Q_k.empty()$ then
        break
    end if
end for
```

since this is a measure of the additional work incurred when executing via the relaxed priority queue. In order to do this, we need to specify some properties for the dependencies of the incremental algorithms we consider.

Denote by p_{ij} be probability that task with label j depends on task with label i. We require the incremental algorithms to have the following property:

- for each pair of task indices $i < j$, $p_{ij} \leq C/i$ (probability is calculated over the randomness of permutation), where C is large enough constant which depends on the incremental algorithm.

The property of incremental algorithms such as Delaunay triangulation and Comparison sorting is established in [BGSS16], but we will briefly go over them to give some intuition.

Comparison Sorting. We consider at comparison sorting of n elements (tasks), which is implemented by inserting them in the Binary Search Tree in increasing label order. Recall that labels are assigned according to the random permutation, this makes sure that the binary search tree is well balanced. Each element (task) depends on all its ancestors in the binary search tree, since our goal is to preserve binary search tree build by execution with the exact scheduler. Alternative way to define dependencies is: for element with label j to depend on element with label $i < j$, element with label j must be a child of element with label i in the binary search tree which given by insertion of elements with labels $1, 2, .., i-1, i, j$ (precisely in this order). Also, note that element with label j is a leaf node in the binary search tree (with $i+1$ elements), since it is the last element to be inserted. Hence, by the properties of binary search tree element with label j can depend on element with label i, only if they are placed consecutively when elements with labels $1, 2, ..., i, j$ are sorted according to their values. If we fix of elements with labels $1, 2, .., i-1, i, j$, but not the order of elements with labels $1, 2, ..., i$, we will get that there are only two elements among the elements with labels $1, 2, .., i$ on which the element with label j depends on. Hence, since the permutation which assigns labels is random, the probability of the element with label i being one of these two elements is at most $\frac{2}{i}$.

Delaunay Triangulation. We are given n point in 2D plane, our aim is to find triangulation such that no point encroaches on any triangle. Point encroaches on a triangle if it is inside its circumcircle. We assume non-degenerate case where no four points lie on the same circle and no there are on the same line. We again assign labels to points (tasks) according to the random permutation and incrementally add them to the triangulation. Every time point is added, we detect the area which consists of all triangles it encroaches on. Then, we remove all the edges inside the area and retriangulate by connecting the newly added point to the points of area. The purpose of dependencies in this case is to make sure that algorithm performs the same steps when adding the point. More precisely, the area it encroaches on should be the same both in exact and relaxed executions, also, the triangles which share edges with the area should also be the same since they also get tested for encroachment. Hence, the point with label j depends on the point with label $i < j$ if in the Delaunay triangulation of points with labels $1, 2, .., j-1, j$, none of the triangles i belongs to, shares an edge with a triangle j belongs to. If we fix the points with labels $1, 2, ..., j-1, j$, but not their order, the expected number of neighbours the point with label j has in the triangulation is at most 6 (We use Euler's theorem and the fact that labels are assigned according to the random permutation). Then once the point with label j is fixed and it has d_j neighbours, we know that in order for the point with label j to depend on the point with label i, either j must be neighbour of i or there must exist vertices u and v, such that i, u, v and j, u, v form a triangle. Hence, since there are $2d_j$ such points and the permutation is random, probability that the point with label j depends on the point with label i is at most $\frac{2d_j}{j-1} \leq \frac{2d_j}{i}$. Hence, after taking the randomness of d_j into the account, the probability that the point with label j depends on the point with label i is at most

$$\sum_{k=1}^{i} Pr[d_j = k]\frac{2k}{i} = \frac{2\,\mathbb{E}[d_j]}{i} \leq \frac{12}{i}.$$

3.5.2 Analysis

In this section, we prove an upper bound on the number of extra steps required by our generic relaxed framework for executing incremental algorithms. As a first step, we will derive some additional properties of the relaxed scheduler.

Let A_{ij} be the event that task with label at least j is returned by the scheduler before task with label i is processed by the incremental algorithm. Observe that if the scheduler returns the highest priority task, then this task can always be processed by the incremental algorithm, since this task is guaranteed to have no ancestors.

Lemma 3.5.1 *If $j - i \geq 2k^2$, $Pr[A_{ij}] = 0$.*

Proof. For labels $j - i \geq 2k^2$, let u be the task with label i and let v be some task with label at least j. Also, let t be the earliest step at which rank of u is at most k. This means that at time step $t - 1$, $rank(u) > k$ and by rank property no tasks with labels larger than i were scheduled at time steps $1, 2, ..., t - 1$. Thus, we have that at time step t, $rank(v) > j - i \geq 2k^2$. Because of the fairness property it takes k steps to remove the task with highest priority(lowest label), so task u will be returned by the scheduler and subsequently will be processed by the algorithm no later than at time step $t + k^2$. Rank of v can decrease by at most 1 after each step, thus at time step $t + k^2$, $rank(v) > 2k^2 - k^2 \geq k$. Hence, v can be returned by the scheduler only after time step $t + k^2$ and this gives us that $Pr[A_{ij}] = 0$. □

For any label i, let R_i be the number of times scheduler returns task with label greater than i (some task can be counted multiple times), before task with label i is processed by the algorithm. The following holds:

Lemma 3.5.2 *For any i, $R_i \leq k^2$.*

Proof. Let u be the task with label i. Also, let t be the earliest step at which rank of u is at most k.

This means that at time step $t - 1$, $rank(u) > k$ and by rank property no task with label at least i can be returned by the scheduler at time steps $1, 2, ..., t - 1$. Because of the fairness property it takes k steps to remove the task with highest priority(lowest label), so task u will be returned by the scheduler and subsequently will be processed by the algorithm no later than at time step $t + k^2$. Trivially, the total number of times some task with label at least i(or in fact any label) can be returned by the scheduler before the time step $t + k^2$ is k^2. □

With the above lemmas in place we can proceed to prove an upper bound for the extra number of steps.

Theorem 3.5.3 *The expected number of extra steps is upper bounded by $O(poly(k) \log n)$.*

Proof. Let D_{ij} be the event that task with label j depends on task with label $i < j$. From the properties of incremental algorithms we consider, we get that $Pr[D_{ij}] = p_{ij} \leq C/i$.

Recall that for $i < j$, A_{ij} is the event that task with label at least j is returned by relaxed scheduler before task with label i is processed by the algorithm.

Note that pair of labels (i, j) can be charged only if D_{ij} and A_{ij}. Let L_{ij} be the event that (i, j) is charged at least once. That is, L_{ij} will happen if and only if D_{ij} and A_{ij} happen. Thus, if $j - i \geq 2k^2$, using Lemma 3.5.1 we have that $Pr[L_{ij}] \leq Pr[A_{ij}] = 0$. When $j - i < 2k^2$, the property of incremental algorithms allows us show that, $Pr[L_{ij}] \leq Pr[D_{ij}] \leq \frac{C}{i}$. Additionally, it is easy to see that the total number of times (i, j) can be charged is upper bounded by

R_i(Recall 3.5.2). By combining the arguments above, we get that:

$$\begin{aligned}
E[\#extrasteps] &\leq \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr[L_{ij}]R_i \\
&\overset{\text{Lemma 3.5.2}}{\leq} \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr[L_{ij}]k^2 \\
&\overset{\text{Lemma 3.5.1}}{\leq} \sum_{i=1}^{n-1}\sum_{j=i+1}^{i+2k^2} Pr[L_{ij}]k^2 \\
&\leq \sum_{i=1}^{n-1}\sum_{j=i+1}^{i+2k^2} Pr[D_{ij}]k^2 \\
&\leq \sum_{i=1}^{n-1}\sum_{j=i+1}^{i+2k^2} \frac{C}{i}k^2 \leq \sum_{i=1}^{n-1} \frac{C}{i}2k^4 \leq O(k^4 \log n). \qquad (3.3)
\end{aligned}$$

□

3.6 Lower Bound on Wasted Work

In this section, we prove the lower bound on the cost of relaxation in terms of additional work. We emphasize the fact that this argument does not require the scheduler to be adversarial: in fact, we will prove that a fairly benign relaxed priority scheduler, the MultiQueue [RSD15], can cause incremental algorithms to incur $\Omega(\log n)$ wasted work.

More precisely, let $inv_{i,i+1}$ be event that the relaxed scheduler returns the task with label $i+1$ before task with label i. First, we will prove the following claim for $MultiQueue$ being used as a relaxed scheduler:

Claim 1 *For every $i > 1$, $Pr[inv_{i,i+1}] \geq 1/8$.*

Proof. First we describe how incremental algorithms work using $MultiQueues$. The $MultiQueue$ maintains k sequential priority queues, where k can be assumed to be a fixed parameter. As before, each task is assigned a label according to the random permutation of input tasks (lower label means higher priority). Initially, all tasks are inserted in $MultiQueue$ as follows: for each task, we select one priority queue uniformly at random out, and insert the task into it. To retrieve a task, the processor selects two priority queues uniformly at random and returns the task with highest priority (lowest label), among the tasks on the top of selected priority queues.

Let $\ell(u) = i$ and $\ell(v) = i + 1$. Additionally, let q_u and q_v be the queues where u and v are inserted in initially. Also, let $T_{u,v}$ be event that u and v are the top tasks of queues at some point during the run of our algorithm. We have that:

$$\begin{aligned}
Pr[inv_{i,i+1}] &= Pr[q_u \neq q_v]\Bigg(Pr[T_{u,v}]Pr[inv_{i,i+1}|T_{u,v}, q_u \neq q_v] \\
&\quad + Pr[\neg T_{u,v}]Pr[inv_{i,i+1}|\neg T_{u,v}, q_u \neq q_v]\Bigg) \\
&\geq (1-\frac{1}{k})Min\Bigg(Pr[inv_{i,i+1}|T_{u,v}, q_u \neq q_v], \\
&\qquad Pr[inv_{i,i+1}|\neg T_{u,v}, q_u \neq q_v]\Bigg).
\end{aligned}$$

Observe that if $\neg T_{u,v}$ and $q_u \neq q_v$, this means that tasks u and v are never compared against each other. Consider two runs of our algorithm until it returns either u or v, first with initially chosen q_u and q_v and second with q_u and q_v swapped (these cases have equal probability of occurring). Since vertices u and v have consecutive labels and are never compared by the MultiQueue, this means that all the comparison results are the same in both cases, hence the scheduler has equal probability of returning u or v. (It is worth mentioning here is that $T_{u,v}$ only depends on values q_u and q_v and does not depend on their ordering.)

This means that :

$$Pr[inv_{i,i+1}|\neg T_{u,v}, q_u \neq q_v] \geq 1/2.$$

Now we look at the case where u and v are top tasks of queues at some step t. Let X_u be event that u is returned by MultiQueue and similarly, let X_v be event that v is returned. We need to lower bound the probability that X_v happens before X_u . We can safely ignore all the other tasks returned by scheduler and processed by algorithm since it is independent of whether u or v is returned first. Let r be the number of top tasks in queues which have labels larger than $i+1$. At step t, $Pr[X_u] = 3/k^2 + 2r/k^2$ and $Pr[X_v] = 1/k^2 + 2r/k^2$, So we have that

$$Pr[X_u] \leq 3Pr[X_v].$$

Observe that during the run of algorithm r will start to increase but we will always have invariant that $Pr[X_u] \leq 3Pr[X_v]$. This means that probability that X_v happens before X_u is at least:

$$\frac{Pr[X_v]}{Pr[X_u] + Pr[X_v]} \geq 1/4.$$

This gives us that:

$$Pr[inv_{i,i+1}|T_{u,v}, q_u \neq q_v] \geq 1/4.$$

and consequently, since $1 - 1/k \geq 1/2$ we get that:

$$Pr[inv_{i,i+1}] \geq (1-1/k)/4 \geq 1/8.$$

□

Theorem 3.6.1 *For Delaunay triangulation and comparison sorting, the expected number of extra steps is lower bounded by* $\Omega(\log n)$.

Proof. To establish the lower bound, we can assume that if the scheduler returns vertex v, which depends on some other unprocessed vertex, we check if vertex u with label $\ell(v) - 1$ is not processed and we charge edge $e = (u, v)$, if v depends on u. This way, we get that $p_{i,i+1}$ and $Pr[inv_{i,i+1}]$ are not correlated, since if we run algorithm to the point where vertex with label i or $i+1$ is returned, it will never check the dependency between them.

We will employ the following property of Delaunay triangulation and BST-based comparison sorting: for any $i > 0$, $p_{i,i+1} \geq 1/i$. This property is easy to verify: in Delaunay triangulation there is at least $1/i$ probability that vertices with labels i and $i+1$ are neighbours in the Delaunay triangulation of vertices with labels $1, 2, ..., i, i+1$, in BST based comparison sorting there is at least $1/i$ probability that tasks with labels i and $i+1$ have consecutive keys among keys of tasks with labels $1, 2, ..., i, i+1$ and in both cases the task with label $i+1$ will depend on the task with label i (see [BGSS16]).

This, in combination with Claim 1 will give us the lower bound on the number of extra steps, since if task with label $i+1$ depends on the task with label i and it is returned first by scheduler, this will trigger at least one extra step, caused by not being able to process task:

$$E[\#extrasteps] \geq \sum_{i=1}^{n-1} p_{i,i+1} Pr[inv_{i,i+1}] \geq 1/8 \log n.$$

□

3.7 Related Work

Parallel scheduling of iterative algorithms is a vast area, so a complete survey is beyond our scope. We begin by noting that our results are not relevant to standard work-stealing schedulers [BL99, BJK$^+$96] since such schedulers do not provide any guarantees in terms of the *rank of elements removed*. We are aware of only one previous attempt to add priorities to work-stealing schedulers [IS15], using a multi-level global queue of tasks, partitioned by priority. This technique is different, and provides no work guarantees.

An early instance a relaxed scheduler is in the work of Karp and Zhang [KZ93], for parallelizing backtracking in the PRAM model. This area has recently become very active, and several relaxed scheduler designs have been proposed, trading off relaxation and scalability, e.g. [SL00, BFK$^+$11, WGTT15, AKLS15, HLH$^+$13, NLP13, RSD15, AKLN17, SW17]. In particular, software packages for graph processing [NLP13] and machine learning [GLG$^+$12] implement such relaxed schedulers.

Our work is related to the line of research by Blelloch et al. [Ble17, BFS12, BFGS12, SGB$^+$14, BGSS16], as well as [CRT87, CF90, FN18], which examines the dependency structure of a broad set of iterative/incremental algorithms and exploit their inherent parallelism for fast implementations. We benefit significantly from the analytic techniques introduced in this work.

We note however some important differences between these results and our work. The first difference concerns the scheduling model: references such as [BFS12, SGB$^+$14, BGSS16] assume a synchronous PRAM execution model, and focus on analyzing the maximum dependency length of algorithms under random task ordering, validating the results via concurrent implementations. By contrast, we employ a relaxed scheduling model, that models data processing systems based on relaxed priority schedulers, such as [NLP13], and provide work bounds for such executions. Although superficially similar, our analysis techniques are different from those of e.g. [BFS12, BGSS16] since our focus is not on the maximum dependency depth of the algorithms, but rather on the number of local dependencies which may be exploited by the adversarial scheduler to cause wasted work. We emphasize that the fact that the algorithms we consider may have low dependency depth does not necessarily help, since a sequential algorithm could have low dependency depth and be inefficiently executable by a relaxed scheduler: a standard example is when the dependency depth is low (logarithmic), but each “level" in a breadth-first traversal of the dependency graph has high fanout. This has low depth, but would lead to high speculative overheads. (In practice, greedy graph coloring on a dense graph would lead to such an example.)

A second difference concerns the interaction between the scheduler and the algorithm. The scheduling mechanisms proposed in e.g. [BFS12] assume knowledge about the algorithm structure, and in particular adapt the length of the prefix of tasks which can be scheduled at any given time to the structure of the algorithm. In contrast, we assume a basic scheduling model, which may even be adversarial (up to constraints), and show that such schedulers, which relax priority order for increased scalability, inherently provide bounded overheads in terms of wasted work due to relaxation.

Finally, we note that references such as [BFS12, BGSS16] focus on algorithms which are efficient under random orderings of the tasks. In the case of SSSP, we show that relaxed schedulers can efficiently execute algorithms which have a fixed optimal ordering.

Another related reference is [ABKN18], in which we introduced the scheduling model used in this paper, related it to MultiQueue schedulers [RSD15], and analyzed the work complexity of some simple iterative greedy algorithms such as maximal independent set or greedy graph coloring. We note the technique introduced in this paper only covered a relatively limited set of iterative algorithms, where the set of tasks are defined and fixed in advance, and focused on the complexity of greedy maximal independent set (MIS) under relaxed scheduling. In contrast, here we consider more complex *incremental* algorithms, in which tasks can be added and modified dynamically. Moreover, as stated, here we also cover algorithms such as SSSP, in which computation should follow a fixed sequential task ordering, as opposed to a random ordering which was the case for greedy MIS.

CHAPTER 4

Elastic Consistency: A Practical Semantics Model for Distributed Stochastic Gradient Descent

4.1 Introduction

Machine learning models can match or surpass humans on specialized tasks such as image classification [KSH12, HZRS16], speech recognition [SFJY14], or complex games [SHM+16]. One key tool behind this progress is the *stochastic gradient descent (SGD)* family of methods [RM51], which are by and large the method of choice for training large-scale machine learning models. Essentially, SGD can serve to minimize a d-dimensional function $f : \mathbb{R}^d \to \mathbb{R}$, assumed to be differentiable. We commonly assume that we are given access to (possibly noisy) gradients of this function, denoted by $\widetilde{G}$. Sequential SGD will start at a randomly chosen point $\vec{x}_0$, say 0^d, and converge towards a minimum of the function by iterating the following procedure:

$$\vec{x}_{t+1} = \vec{x}_t - \eta \widetilde{G}(\vec{x}_t) \tag{4.1}$$

where $\vec{x}_t$ is the current estimate of the optimum, also called the *parameter* and η is the *learning rate*. If the function is convex, this procedure is known to converge to the minimum of the function [B+15], whereas in the non-convex case, it will converge towards a point of zero gradient [GL13]. In *supervised learning*, f is usually the total error of a given parameter $\vec{x}$ on a given dataset $\mathcal{D}$. For each sample s in $\mathcal{D}$, the classification error is encoded via the loss $\ell(s, \vec{x})$. Training minimizes the function $f(\vec{x}) = \frac{1}{m} \sum_{s \in \mathcal{D}} \ell(s, \vec{x})$, where m is the size of the dataset, and the gradient at a randomly chosen datapoint $\widetilde{G}$ is an unbiased estimator of ∇f.

Due to the size of datasets, it is common to *distribute* the optimization process across multiple processors. A standard way of parallelizing SGD is to process a *batch* of samples in parallel, dividing the computation of gradient updates among processors. Assume for simplicity that each processor is allotted one sample, whose corresponding gradient it computes with respect to the current parameter $\vec{x}_t$. Processors then *sum* their stochastic gradients, and update their local parameters by the resulting sum, leading to the following global iteration:

$$\vec{x}_{t+1} = \vec{x}_t - \frac{\eta}{n} \sum_{i=1}^{n} \widetilde{G}^i(\vec{x}_t), \tag{4.2}$$

where $\widetilde{G}^i$ is the stochastic gradient obtained at the processor i at the given step, and n is the batch size, equal to the number of processors. Since this sum is the same at every processor, this procedure yields the same, perfectly consistent, parameter at each processor at the end of every parallel iteration. The average $(1/n)\sum_{i=1}^{n}\widetilde{G}^i(\vec{x}_t)$ is still a stochastic gradient, but with *lower variance* than gradients at single samples, which can lead to better convergence [B+15]. Since samples are now processed in parallel, the number of samples processed per second should in theory be multiplied by n.

However, in practice, maintaining perfect consistency of the parameter $\vec{x}_t$ can negate the benefits of parallelization. Keeping the parameters perfectly consistent has a *communication cost*: since the size of gradient updates is *linear* in the size of the parameter, the resulting communication may easily become a system bottleneck for models with millions of parameters [KSH12, AGL+17]. Consistency also induces a *synchronization cost*, since processors need to synchronize in a barrier-like fashion upon each iteration update, which can occur every few milliseconds. For this reason, there have been several proposals for relaxing the consistency requirements of SGD-like iterations, under various system constraints. These proposals can be broadly categorized as follows:

- **Asynchronous Methods:** Such implementations [RRWN11, LHLL15] allow processors to forgo the barrier-like synchronization step performed at each iteration, or even across parameter components, and move forward with computation without waiting for potentially slow straggler processors.
- **Communication Compression:** These methods aim to reduce the bandwidth cost of exchanging the gradients. This usually entails performing (possibly lossy) compression of the gradients before transmission, followed by efficient encoding and reduction/summation, and decoding on the receiver end. This can be either via bit-width reduction (quantization), e.g. [SFJY14, AGL+17], or via structured sparsification of the updates [LHM+18, AH17, AHJ+18, KRSJ19].

Additional approaches exist, for instance to reduce the *frequency* of communication via large-batch methods or local steps, e.g. [GDG+17, LSPJ18, Sti19]. Another axis controls parameter maintenance: *centralized* methods such as the *parameter server* [Li14] maintain the parameter at a single entity, whereas *decentralized* methods [LZZ+17, LDS21] have each processor maintain their own version of the model.

The question of providing convergence bounds for distributed optimization goes back to the foundational work of Bertsekas and Tsitsiklis [BT89], and has recently risen to prominence [DCM+12, HCC+13, CSAK14, RRWN11, HCC+13, SZOR15, LHLL15, CDR15, LPLJ16]. However, many of these proofs are often specialized to the algorithm and models, and do not generalize to different settings. It is therefore natural to ask: are there generic conditions covering all natural consistency relaxations for SGD, under which one can prove convergence?

Contribution. In this chapter, we introduce a convergence criterion for SGD-based optimization called *elastic consistency*, which is independent of the system model, but can be specialized to cover various model consistency relaxations.

In a nutshell, elastic consistency says that, for SGD to converge, it is sufficient that the distance between the *view* of the parameter perceived by a processor, with respect to which the gradient is taken, and the "true" view of the system, corresponding to all the updates to the parameter generated up to that point by all processors, be uniformly bounded across iterations, and decreasing proportionally to the learning rate. Intuitively, in this case, the perturbed iterates do not stray "too far" from eachother, and can still globally converge.

To our knowledge, elastic consistency is satisfied in most settings where asynchronous or communication-reduced methods have been analyzed so far, although proving this property for some systems is not always immediate.

Elastic consistency provides a unified analysis framework for all the method types discussed above. Consequently, we are able to re-prove or improve convergence bounds for several methods, and to tackle new models and consistency relaxations. Our contributions are as follows:

1. Under standard smoothness assumptions on the loss, elastic consistency is sufficient to guarantee convergence rates for inconsistent SGD iterations for both *convex* and *non-convex* objectives. This condition is also *necessary* for SGD convergence: we provide simple worst-case instances where SGD convergence is linear in the elastic consistency parameter, showing that the iterations will diverge if elastic consistency is regularly broken.

2. We show that elastic consistency is satisfied by both asynchronous message-passing and shared-memory models, centralized or decentralized, with or without faults, and by communication-reduced methods. This implies new convergence bounds for SGD in the classic asynchronous and semi-synchronous message-passing models [AW04] and extends previous analyses for the shared-memory model [SZOR15, ADSK18].

4.2 Elastic Consistency

Distributed Model and Adversarial Scheduling. We consider distributed systems consisting of n processors: $\{1, 2, \ldots, n\}$, some of which may be faulty, where communication happens either by message-passing, or via shared-memory. For simplicity, we will specify the system and fault models in the corresponding sections. We assume that the scheduling of steps (e.g. reads/writes in shared-memory, or message delivery in message-passing) is controlled by an *oblivious* adversarial entity. This means that scheduling decisions may be adversarial, but are *independent* of the randomness in the algorithm, and in particular of the data sampling. Practically, this implies that the conditioning on any random event involving previous SGD iterations $s < t$ does not impact choices made at iteration t.

Distributed Optimization. We assume that each of the n processors is given access to random samples coming from an unknown d-dimensional data distribution $\mathcal{D}$, and collaborates to jointly minimize $f : \mathcal{X} \to \mathbb{R}$ over the distribution $\mathcal{D}$, where $\mathcal{R}^d$ is a compact subset of $\mathbb{R}^d$. In practice, nodes optimize over a finite set of samples $S = \{S_1, S_2, \ldots, S_m\}$, and the function f is defined as

$$f(\vec{x}) = \frac{1}{m} \sum_{i=1}^{m} \ell(S_i, \vec{x}) \tag{4.3}$$

where ℓ is the loss function at a sample s. The goal is to find $\vec{x}^* \in \mathcal{R}^d$, which minimizes the expected loss over samples, defined as: $\vec{x}^* = \text{argmin}_{\vec{x}} f(\vec{x}) = \text{argmin}_{\vec{x}} \mathbb{E}_{s \sim \mathcal{D}}[\ell(s, \vec{x})]$.

Properties of Stochastic Gradients. Let $\tilde{G}$ be a stochastic gradient. We make the following standard assumptions [SZOR15]:

1. **Unbiasedness.** The i.i.d. stochastic gradients are unbiased estimators of the true gradient of the function f:
$$\forall \vec{x} \in \mathbb{R}^d,\ \mathbb{E}\left[\tilde{G}(\vec{x})\right] = \nabla f(\vec{x}). \tag{4.4}$$

2. **Bounded Variance.** The stochastic gradients have bounded variance:
$$\forall \vec{x} \in \mathbb{R}^d,\ \mathbb{E}\left[\|\tilde{G}\left(\vec{x}\right) - \nabla f(\vec{x})\|^2\right] \leq \sigma^2. \tag{4.5}$$

3. **Bounded Second Moment.** It is sometimes assumed that the second moment of the stochastic gradients over the sample space is bounded:
$$\forall \vec{x} \in \mathbb{R}^d,\ \mathbb{E}\left[\|\tilde{G}\left(\vec{x}\right)\|^2\right] \leq M^2. \tag{4.6}$$
Elastic consistency *does not* require the second moment bound to ensure convergence—the variance bound is sufficient. However, in e.g. asynchronous shared-memory [SZOR15], this stronger assumption is common, and we will use it to bound the elastic consistency constant.

Properties of the Objective Function. We will make use of the following standard definitions regarding the objective function f: $\forall\ \vec{x}, \vec{y} \in \mathbb{R}^d$,

1. **Smoothness.** The function $f : \mathbb{R}^d \to \mathbb{R}$ is smooth iff:
$$\|\nabla f\left(\vec{x}\right) - \nabla f\left(\vec{y}\right)\| \leq L\|\vec{x} - \vec{y}\| \text{ for } L > 0. \tag{4.7}$$

2. **Strong convexity.** Problems such as linear regression have a strongly convex objective:
$$(\vec{x} - \vec{y})^T(\nabla f(\vec{x}) - \nabla f(\vec{y})) \geq c\left\|x - y\right\|^2 \text{ for } c > 0. \tag{4.8}$$
For such functions, the bound over second moment of stochastic gradients does not hold $\forall\ \vec{x} \in \mathbb{R}^d$ [NNvD+18]. Therefore, we restrict $f : \mathcal{X} \to \mathbb{R}$ for a convex set $\mathcal{X} \subset \mathbb{R}^d$, such that $\forall x \in \mathcal{X}$, (4.6) is satisfied. For simplicity, we omit the projection step onto $\mathcal{X}$, in the case when $\vec{x}_t$ does not belong to $\mathcal{X}$ for some iteration t.

3. **Lower bound for non-strongly-convex functions.** In many settings, such as training of neural networks, the objective function is not necessarily strongly-convex. In such "non-strongly-convex" settings, it is necessary to assume that f is bounded from below:
$$\exists f^* \text{ finite s.t. } \forall\ \vec{x} \in \mathbb{R}^d,\ f(\vec{x}) \geq f^*. \tag{4.9}$$

4.2.1 Elastic Consistency Definition

An Abstract Consistency Model. We assume that we have n processors, which share a *parameter oracle* $\mathcal{O}$. In each iteration $t+1$ (alternatively, at step $t+1$), each processor i invokes this oracle, and receives a *local view* at of the parameter at step t (after t iterations), which we denote $\vec{v}_t^i$. The processor then uses this view of the parameter to generate a new update (stochastic gradient) $\tilde{G}(\vec{v}_t^i)$, which it applies to the shared model.

The key question is how to express the consistency of the local views across processors. For this, we introduce an auxiliary variable $\vec{x}_t$, which we call the *global parameter*. Initially we have that $\vec{x}_0 = \vec{v}_0^1 = ... = \vec{v}_0^n$.

We consider two cases, depending on how data-parallel SGD is implemented. The first is the **single steps** case, where the gradient generated locally by each processor is *directly applied* to the model. This is done in asynchronous shared-memory [SZOR15] or some message-passing implementations [LHLL15], and is modelled as:
$$\vec{x}_{t+1} = \vec{x}_t - \eta\tilde{G}(\vec{v}_t^i). \tag{4.10}$$

The second is the **parallel steps** case, where processors' gradients are aggregated before they are applied to the shared model, for instance via averaging. This is common in synchronous message-passing settings, e.g. [Li14]. Formally, we are given a set $I_t \subseteq \{1, 2..., n\}$, of gradients to be averaged, such that $\frac{n}{2} \leq |I_t| \leq n$. Each processor $i \in I_t$ calculates a stochastic gradient based on its local view at step t, and the global model is updated by aggregating all the gradients in I_t:

$$\vec{x}_{t+1} = \vec{x}_t - \frac{\eta}{n} \sum_{i \in I_t} \tilde{G}(\vec{v}_t^i). \tag{4.11}$$

Then, elastic consistency says the following:

Definition 4.2.1 (Elastic Consistency) *A distributed system provides* elastic consistency *for the SGD iteration defined in (4.10) or (4.11), if there exists a constant $B > 0$, which, for every step t, bounds the expected norm difference between the true parameter $\vec{x}_t^i$, and the view $\vec{v}_t^i$ returned by the oracle at processor i at step t. Importantly, B should be independent of the iteration count t, but possibly dependent on the system definition. Formally,*

$$\mathbb{E}\left[\|\vec{x}_t - \vec{v}_t^i\|^2\right] \leq \eta^2 B^2, \tag{4.12}$$

where $B > 0$, η is the learning rate, and the expectation is taken over the randomness in the algorithm. We call B the elastic consistency constant.

Discussion. First, please note that, in the above, time t counts each time step at which a stochastic gradient is generated at a node, in sequential order. Intuitively, in the **single steps** case, the auxiliary parameter $\vec{x}_t$ is defined as the sum of generated gradients up to and excluding step t, multiplied by the corresponding learning rate. In the analysis, we need to define $\vec{x}_t$ precisely for each application; please see Table 4.1 for example bounds, which we will prove in Section 4.4.

Generally, the process for deriving the elastic consistency bound B for a given system is as follows. We assume a *fixed, constant* LR sequence, which ensures convergence w.r.t. the sequential iteration (1). We then examine the distributed system specification to bound $\|\vec{x}_t - \vec{v}_t\|^2$. (Please see Section 4.6 for step-by-step examples.) Crucially, for all the systems and consistency relaxations we analyze, this bound separates cleanly into the *LR part* (η^2), and a *constant part* (B^2) which is *independent of time or LR*. This holds for all t. Finally, the learning rate sequence used by the algorithm may need to be adjusted (e.g. normalized) to satisfy certain convergence constraints.

In Section 4.4, we show that virtually all known models and consistency relaxations satisfy this condition (see Table 4.1). Elastic consistency gives wide latitude to the parameter oracle about which exact values to return to the processor: the returned view can contain random or even adversarial noise, or updates may be delayed or missing, as long as their relative weight is bounded and independent of time.

4.3 Elastic Consistency and SGD Convergence

We now show that this notion of consistency implies non-trivial convergence guarantees for SGD for different types of objective functions, and that this notion is in some sense necessary for convergence.

4.3.1 Elastic Consistency is *Sufficient* for Convergence

The Non-Convex Case. We begin with the more general case where the objective function is not necessarily convex. In this case, since convergence to a global minimum is not guaranteed for SGD, we will only require convergence to a point of vanishing gradients, as is standard, e.g. [WWLZ18]. Specifically, assuming elastic consistency, we prove the following theorems:

Theorem 4.3.1 *Consider SGD iterations defined in (4.10) and satisfying elastic consistency bound (4.12). For a smooth non-convex objective function f, whose minimum x^* we are trying to find and the constant learning rate $\eta = \frac{1}{\sqrt{T}}$, where $T \geq 64L^2$ is the number of iterations:*

$$\min_{t \in [T-1]} \mathbb{E}\|\nabla f(\vec{x}_t)\|^2 \leq \frac{8(f(\vec{x}_0) - f(x^*))}{\sqrt{T}} + \frac{4B^2L^2}{T} + \frac{8L\sigma^2}{\sqrt{T}} + \frac{16L^3B^2}{T\sqrt{T}}.$$

This result can be specialized to parallel iterations (4.11), yielding the following theorem, which ensures $\sqrt{n}$ parallel speedup in the number of nodes n:

Theorem 4.3.2 *Consider SGD iterations defined in (4.11) and satisfying elastic consistency bound (4.12). For a smooth non-convex objective function f, whose minimum x^* we are trying to find and the constant learning rate $\eta = \frac{\sqrt{n}}{\sqrt{T}}$, where $T \geq 64L^2n$ is the number of iterations:*

$$\min_{t \in [T-1]} \mathbb{E}\|\nabla f(\vec{x}_t)\|^2 \leq \frac{8(f(\vec{x}_0) - f(x^*))}{\sqrt{Tn}} + \frac{4B^2L^2n}{T} + \frac{8L\sigma^2}{\sqrt{Tn}} + \frac{16L^3B^2n\sqrt{n}}{T\sqrt{T}}.$$

The Strongly Convex Case. We can provide improved guarantees under strong convexity:

Theorem 4.3.3 *Consider SGD iterations defined in (4.10) and satisfying elastic consistency bound (4.12). For a smooth, strongly convex function f, whose minimum x^* we are trying to find and the constant learning rate $\eta = \frac{2\log T}{cT}$, where $T \geq \frac{256L^2}{c^2}$ is a number of iterations:*

$$\mathbb{E}\|\vec{x}_T - x^*\|^2 \leq \frac{\|\vec{x}_0 - x^*\|^2}{T} + \frac{16\log^2 T L^2B^2}{c^4T^2} + \frac{12\sigma^2\log T}{T} + \frac{48\log^3 T B^2L^2}{c^4T^3}.$$

This convex result also directly generalizes to the parallel case (4.11), where we obtain linear speed-up in n:

Theorem 4.3.4 *Consider SGD iterations defined in (4.11) and satisfying elastic consistency bound (4.12). For a smooth, strongly convex function f, whose minimum x^* we are trying to find and the constant learning rate $\eta = \frac{2(\log T + \log n)}{cT}$, where $T \geq \frac{256L^2p}{c^2}$ is a number of iterations:*

$$\mathbb{E}\|\vec{x}_T - x^*\|^2 \leq \frac{\|\vec{x}_0 - x^*\|^2}{Tn} + \frac{16(\log T + \log n)^2L^2B^2}{c^4T^2} + \frac{12\sigma^2(\log T + \log n)}{Tn} + \frac{48(\log T + \log n)^3B^2L^2}{c^4T^3}.$$

Discussion. The parameter B abstracts the distributed-system specific parameters to provide a clean derivation of the convergence theory. In turn, depending on the system setting, B might depend on the second-moment bound M^2 or variance bound σ^2, but also on system parameters such as the maximum delay τ, on the number of failures f, or on the characteristics of the compression scheme. Specifically, assuming that the parameters are constant, the convergence of the non-convex objectives is at a rate of $O(1/\sqrt{T})$ for SGD iterations defined in (4.10) and $O(1/\sqrt{Tn})$ for SGD iterations defined in (4.11). For a strongly-convex objective, we achieve rates of $\tilde{O}(1/T)$ and $\tilde{O}(1/(Tn)))$ for SGD iterations defined in (4.10) and (4.11) correspondingly (Here, $\tilde{O}$ notation hides $\log T$ and $\log n$ factors).

4.3.2 Elastic Consistency is *Necessary* for Convergence

We can also show that elastic consistency can be directly linked to convergence; in particular, in the worst case, an adversarial parameter oracle can slow down convergence *linearly* in B^2. The intuition is that once the algorithm is close to the minimum, an adversarial oracle can force it to evaluate gradient at point which is B away and this will cause the algorithm to overshoot the minimum. The argument is similar to that of [ADSK18].

Lemma 4.3.5 (Convergence Lower Bound) *There exists a convex (quadratic) function f and an adversarial oracle $\mathcal{O}$ s.t. the algorithm with elastic consistency bound converges B times slower than the exact algorithm (B=0).*

4.4 Distributed System Models and their Elastic Consistency Bounds

4.4.1 Fault-Tolerant Message-Passing Systems

We consider a message-passing (MP) system of n nodes $\mathcal{P} = \{1, 2, \dots, n\}$ executing SGD iterations, which are connected to each other by point-to-point links. To simplify the description, we will focus on the case where the system is *decentralized*: in this case, each node i acts both as a worker (generating gradients) and as a parameter server (PS) (maintaining a local parameter copy). (The only difference is that, in the *centralized* PS case, a single designated node would maintain a global parameter copy.) Without loss of generality, all nodes start with the same parameter $\vec{v}_0^i = \vec{0}$. The system proceeds in global iterations, indexed by t. In each iteration, workers generate a stochastic gradient based on its current model copy $\vec{v}_t^i$, and broadcasts it to all other nodes.

Consistency Relaxations: Consider node i, and recall that it acts as a parameter server, in addition to being a worker itself. Let $\mathcal{L}_t^i \subseteq \{1, \dots, n\}$ denote the set of nodes from which i receives stochastic gradients at iteration t. By convention, $i \in \mathcal{L}_t^i$. In the *synchronous failure-free* case, all nodes would receive exactly the same set of messages, and the execution would be equivalent to a sequential batch-SGD execution. In a real system, not all nodes may have the same view of each round, due to failures or asynchrony. Specifically, we consider the following distinct consistency relaxations:

(a) **Crash faults.** A node $i \in \mathcal{P}$ may *crash* during computation or while sending messages. In this case, the node will remain inactive for the rest of the execution. Importantly, node i's crash during broadcasting may cause other nodes to have different views at the iteration, as some of them may receive i's message while others may not, resulting in inconsistent updates of parameter $\vec{x}$ across the nodes. We will assume that $f \leq n/2$ nodes may crash in the message-passing system.

(b) **Message-omission failures.** In practical systems, each node could implement iteration t by waiting until an interval Υ^t_{max} in terms of *clock time* has elapsed: if the message from peer j is not received in time, node i moves on, updating its local parameter $\vec{x}^i_t$ only w.r.t. received messages. However, node i will include j's message into its view if received in a later iteration, although some messages may be permanently delayed. We assume a parameter f which upper bounds the number of messages omitted at any point during the execution. We note that this model is stronger than the Crash-fault model considered above, as we can simulate a node's failure by discarding all its messages after the crash.

(c) **Asynchrony.** The two above models assume that nodes proceed synchronously, in *lock-step*, although they may have inconsistent views due to node or message failures. An alternative relaxation [LHLL15] is if nodes proceed *asynchronously*, i.e. may be in different iterations at the same point in time. Specifically, in this case, we assume that there exists a maximum delay τ_{max} such that each message/gradient can be delayed by at most τ_{max} iterations from the iteration when it was generated.

(d) **Communication-Compression.** Another way of reducing the distribution cost of SGD has been to compress the stochastic gradients communicated at each round. In this context, sparsification with memory [Str15, SFJY14, AH17, AHJ+18, KRSJ19] has proven to be particularly effective. This process can be modelled as follows. We assume that each node maintains a local version of the parameter $\vec{v}^i_t$, and an *error/memory* vector $\vec{\epsilon}^i_t$, initially $\vec{0}$. In each iteration, each node computes a new gradient $\widetilde{G}\left(\vec{v}^i_t\right)$ based on its local parameter. It then adds the current error vector $\vec{\epsilon}^i_t$ to the gradient, to obtain its full proposed update $\vec{\nabla}^i_t$. However, before transmitting this update, it compresses it by using a (lossy) compression function Q. The compressed update $Q(\vec{\nabla}^i_t)$ is then transmitted, and the error vector is updated to $\vec{\epsilon}^i_{t+1} \leftarrow \vec{\nabla}^i_t - Q(\vec{\nabla}^i_t)$. Our analysis will only require that Q satisfies $\|Q(\vec{\nabla}) - \vec{\nabla}\|^2 \leq \gamma \|\vec{\nabla}\|^2,\ \forall \vec{\nabla} \in \mathbb{R}^d$, for some $1 > \gamma \geq 0$. All memory-based techniques satisfy this, for various definitions of Q and γ. (We provide examples in Section 4.6.6.)

We note that the above discussion considered these methods independently. However, we do note that our method does allow for these relaxations to be combined—for instance, one can analyze an asynchronous, fault-tolerant method with communication compression.

4.4.2 Asynchronous Shared-Memory Systems

We consider a system with n processors (or threads) $\mathcal{P} = \{1, 2, \ldots, n\}$, which can communicate through shared memory. Specifically, we assume that the parameter vector $\vec{w} \in \mathbb{R}^d$ is shared by the processors, and is split into d components, one per dimension. Processors can atomically read a component via a `read` operation, and update it via the atomic `fetch&add` (`faa`) operation, which reads the current value of the component and updates it in place, in a single atomic step. In each iteration t, each processor first obtains a local view $\vec{v}^i_t$ of the parameter by scanning through the shared parameter $\vec{w}$ component-wise. It then generates a stochastic gradient $\widetilde{G}\left(\vec{v}_t^i\right)$ based on this view, and proceeds to update $\vec{x}$ via `faa` on each component, in order. (We refer the reader to Section 4.6.5 for a full description, including pseudocode.)

Consistency Relaxation. Ideally, threads would proceed in lock step, first obtaining perfect, identical snapshots of $\vec{w}$, calculating gradients in terms of this identical parameter, and then summing the gradients before proceeding to the next iteration. However, in practice, threads are *asynchronous*, and proceed at arbitrary speeds. This causes their snapshots to be inconsistent, as they might contain some partial concurrent updates, but not others. The challenge is to prove SGD convergence in this case. It is common [RRWN11, SZOR15] to assume a bound τ_{max} on the maximum delay between the time (iteration) when an individual update was generated, and the iteration when it has been applied, and becomes visible to all processors. In this case, the auxiliary variable $\vec{x}_t$ used by elastic consistency will correspond to the sum of first t stochastic gradients, ordered by the time when the atomic `faa` over the first index of $\vec{w}$ was performed.

4.4.3 Elastic Consistency Bounds for Specific Systems

Given these definitions, we can now state the elastic consistency bounds for the different types of distributed systems and consistency relaxations. Please see Table 4.1, and the Appendix for detailed derivations.

System	**Consistency Relaxation**	Bound B	**Novelty**
Shared-Memory	τ_{max}-Bounded Asynchrony	$\sqrt{d}\tau_{max}M$	Extends [SZOR15, ADSK18]
Message-Passing	τ_{max}-Bounded Asynchrony	$\frac{(n-1)\tau_{max}M}{n}$	Reproves [LHLL15]
Message-Passing	τ_{max}-Bounded Asynchrony	$O(\frac{(n-1)\tau_{max}\sigma}{n})$	New
Message-Passing	*Distributed* Communication-Compression with Error Feedback	$\sqrt{\frac{(2-\gamma)\gamma}{(1-\gamma)^3}}M$	Improves [AHJ$^+$18, SCJ18, KRSJ19]
Message-Passing	Synchronous, f Crash or Message-drop Faults	Mf/n	New
Message-Passing	Synchronous, f Crash or Message-drop Faults	$O(\sigma f/n)$	New

Table 4.1: Summary of elastic consistency bounds.

Implications. Plugging the asynchronous shared-memory bound into Theorems 4.3.1 and 4.3.3 implies convergence bounds in the smooth non-convex case, extending [SZOR15, ADSK18], which focus on the convex case, whereas the asynchronous message-passing bound implies similar bounds to the best known in the non-convex case for this model [LHLL15]. For synchronous message-passing with communication-compression, our framework implies the first general bounds for the *parallel, multi-node* case: references [SCJ18, KRSJ19] derive tight rates for such methods, but in the *sequential* case, where there is a single node which applies the compressed gradient onto its model, whereas [AHJ$^+$18] considers the multi-node case, but requires an additional analytic assumption. Please see Section 4.7 for additional discussion on related work. In the crash-prone case, elastic consistency implies new convergence bounds for crash or message-omission faults. Note that, although the elastic bound is the same for both f crash and message-omissions (Mf/n), the derivations in the Appendix are slightly different. The framework also allows us to combine consistency relaxations, i.e. consider communication-compression with crashes.

One relative weakness of the above results is that the bounds depend on the gradient second-moment bound. This is not due to elastic consistency itself, but due to the fact that we needed a bound on M to bound the elastic consistency constant for these systems, which is consistent with previous work, e.g. [SZOR15, ADSK18, LHLL15]. Next, we show that this limitation, which is common in the literature, can be removed by slightly altering the algorithms.

4.5 Detailed Convergence Analysis

4.5.1 Complete Proof of Convergence in the Non-Convex Case

In order to unify analysis for iterations (4.10) and (4.11) and we consider the following iterations:

$$\vec{x}_{t+1} = \vec{x}_t - \frac{\eta}{k}\sum_{i \in I_t} \tilde{G}(\vec{v}_t^i). \tag{4.13}$$

where we require that $\frac{\|I_t\|}{k} \geq \frac{1}{2}$. (4.10) is (4.13) with $k=1$ and $I_t = \{i\}$, and (4.11) is (4.13) with $k=n$. Note that since in (4.11) $|I_t| \geq \frac{n}{2}$, the requirement for (4.13) is satisfied.

Lemma 4.5.1 *Consider SGD iterations defined in (4.13) and satisfying elastic consistency bound (4.12). For a smooth non-convex objective function f and the constant learning rate $\eta \leq \frac{1}{8L}$. We have that:*

$$\mathbb{E}[f(\vec{x}_{t+1})] \leq \mathbb{E}[f(\vec{x}_t)] - \frac{\eta}{8}\mathbb{E}\|\nabla f(\vec{x}_t)\|^2 + \frac{\eta^3 B^2 L^2}{2} + \frac{L\eta^2\sigma^2}{k} + 2L^3\eta^4 B^2. \tag{4.14}$$

Proof. We condition on $\vec{x}_t$ and $\{\vec{v}_t^i | i \in I_t\}$ and calculate expectation with respect to the randomness of stochastic gradient(We call this $\mathbb{E}_s$). By descent lemma we get that :

$$\begin{aligned}
\mathbb{E}_s[f(\vec{x}_{t+1})] &\leq f(\vec{x}_t) - \sum_{i \in I_t}\frac{\eta}{k}\mathbb{E}_s\langle \tilde{G}(\vec{v}_t^i), \nabla f(\vec{x}_t)\rangle + \frac{L^2\eta^2}{2k^2}\mathbb{E}_s\|\sum_{i\in I_t}\tilde{G}(\vec{v}_t^i)\|^2 \\
&\overset{(4.4)}{=} f(\vec{x}_t) - \frac{\eta|I_t|}{k}\|\nabla f(\vec{x}_t)\|^2 + \frac{\eta}{k}\sum_{i\in I_t}\langle \nabla f(\vec{x}_t) - \nabla f(\vec{v}_t^i), \nabla f(\vec{x}_t)\rangle \\
&\quad + \frac{L\eta^2}{2k^2}\mathbb{E}_s\left\|\sum_{i\in I_t}\Big(\tilde{G}(v_t^i) - \nabla f(\vec{v}_t^i) + \nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t) + \nabla f(\vec{x}_t)\Big)\right\|^2 \\
&\overset{Cauchy-Schwarz}{\leq} f(\vec{x}_t) - \frac{\eta|I_t|}{k}\|\nabla f(\vec{x}_t)\|^2 + \frac{\eta}{k}\sum_{i\in I_t}\langle \nabla f(\vec{x}_t) - \nabla f(\vec{v}_t^i), \nabla f(\vec{x}_t)\rangle \\
&\quad + \frac{L\eta^2}{k^2}\mathbb{E}_s\left\|\sum_{i\in I_t}\Big(\tilde{G}(v_t^i) - \nabla f(\vec{v}_t^i)\Big)\right\|^2 \\
&\quad + \frac{L\eta^2}{k^2}\mathbb{E}_s\left\|\sum_{i\in I_t}\Big(\nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t) + \nabla f(\vec{x}_t)\Big)\right\|^2
\end{aligned}$$

Observe that $\mathbb{E}_s\left\|\sum_{i\in I_t}\Big(\tilde{G}(v_t^i) - \nabla f(\vec{v}_t^i)\Big)\right\|^2$ is the same as a variance of random variable $\sum_{i\in I_t}\tilde{G}(v_t^i)$. Recall that we are conditioning on $\vec{x}_t$ and $\{\vec{v}_t^i | i \in I_t\}$, hence $\tilde{G}(v_t^i)$ are independent random variables (because agents in I_t compute stochastic gradients independently). This means that because of the variance bound (4.5):

$$\mathbb{E}_s \left\| \sum_{i \in I_t} \left(\tilde{G}(v_t^i) - \nabla f(\vec{v}_t^i) \right) \right\|^2 \leq |I_t| \sigma^2.$$

By combining the two inequalities above we get:

$$\begin{aligned}
\mathbb{E}_s[f(\vec{x}_{t+1})] &\overset{Cauchy-Schwarz}{\leq} f(\vec{x}_t) - \frac{\eta |I_t|}{k} \|\nabla f(\vec{x}_t)\|^2 + \frac{\eta}{k} \sum_{i \in I_t} \langle \nabla f(\vec{x}_t) - \nabla f(\vec{v}_t^i), \nabla f(\vec{x}_t) \rangle \\
&\quad + \frac{2L\eta^2 |I_t|}{k^2} \sum_{i \in I_t} \left(\|\nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t)\|^2 + \|\nabla f(\vec{x}_t)\|^2 \right) + \frac{L\eta^2 |I_t| \sigma^2}{k^2} \\
&\overset{Young}{\leq} f(\vec{x}_t) - \frac{\eta |I_t|}{k} \|\nabla f(\vec{x}_t)\|^2 + \sum_{i \in I_t} \frac{\eta}{2k} \|\nabla f(\vec{x}_t) - \nabla f(\vec{v}_t^i)\|^2 + \frac{\eta |I_t|}{2k} \|\nabla f(\vec{x}_t)\|^2 \\
&\quad + \frac{2L\eta^2 |I_t|}{k^2} \sum_{i \in I_t} \left(\|\nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t)\|^2 + \|\nabla f(\vec{x}_t)\|^2 \right) + \frac{L\eta^2 |I_t| \sigma^2}{k^2} \\
&\overset{(4.7)}{\leq} f(\vec{x}_t) - \frac{\eta |I_t|}{2k} \|\nabla f(\vec{x}_t)\|^2 + \frac{\eta L^2}{2k} \sum_{i \in I_t} \|\vec{x}_t - \vec{v}_t^i\|^2 \\
&\quad + \frac{2L\eta^2 |I_t|}{k^2} \sum_{i \in I_t} \left(L \|\vec{v}_t^i - \vec{x}_t)\|^2 + \|\nabla f(\vec{x}_t)\|^2 \right) + \frac{L\eta^2 |I_t| \sigma^2}{k^2}.
\end{aligned}$$

Next, we use elastic consistency bound in the above inequality:

$$\begin{aligned}
\mathbb{E}[f(\vec{x}_{t+1})] &= \mathbb{E}[\mathbb{E}[f(\vec{x}_{t+1}) | \vec{x}_t, \{\vec{v}_t^i | i \in I_t\}]] \\
&\leq \mathbb{E}[f(\vec{x}_t)] - \frac{\eta |I_t|}{2k} \mathbb{E} \|\nabla f(\vec{x}_t)\|^2 + \frac{\eta L^2}{2k} \sum_{i \in I_t} \mathbb{E} \|\vec{x}_t - \vec{v}_t^i\|^2 \\
&\quad + \frac{2L\eta^2 |I_t|}{k^2} \sum_{i \in I_t} \left(\|\nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t)\|^2 + \|\nabla f(\vec{x}_t)\|^2 \right) + \frac{L\eta^2 |I_t| \sigma^2}{k^2} \\
&\leq \mathbb{E}[f(\vec{x}_t)] - \frac{\eta |I_t|}{2k} \mathbb{E} \|\nabla f(\vec{x}_t)\|^2 + \frac{\eta^3 B^2 L^2 |I_t|}{2k} + \frac{L\eta^2 \sigma^2 |I_t|}{k^2} + \frac{2L^3 \eta^4 B^2 |I_t|^2}{k^2} \\
&\quad + \frac{2L\eta^2 \mathbb{E} \|\nabla f(\vec{x}_t)\|^2 |I_t|^2}{k^2}.
\end{aligned}$$

To finish the proof recall that $k/2 \leq |I_t| \leq k$ and $\eta \leq \frac{1}{8L}$, which if used in the inequality above gives us:

$$\begin{aligned}
\mathbb{E}[f(\vec{x}_{t+1})] &\leq \mathbb{E}[f(\vec{x}_t)] - \frac{\eta |I_t|}{2k} \mathbb{E} \|\nabla f(\vec{x}_t)\|^2 + \frac{\eta^3 B^2 L^2}{2} + \frac{L\eta^2 \sigma^2}{k} + 2L^3 \eta^4 B^2 \\
&\quad + \frac{\eta \mathbb{E} \|\nabla f(\vec{x}_t)\|^2 |I_t|}{4k} \\
&\leq \mathbb{E}[f(\vec{x}_t)] - \frac{\eta |I_t|}{4k} \mathbb{E} \|\nabla f(\vec{x}_t)\|^2 + \frac{\eta^3 B^2 L^2}{2} + \frac{L\eta^2 \sigma^2}{k} + 2L^3 \eta^4 B^2 \\
&\leq \mathbb{E}[f(\vec{x}_t)] - \frac{\eta}{8} \mathbb{E} \|\nabla f(\vec{x}_t)\|^2 + \frac{\eta^3 B^2 L^2}{2} + \frac{L\eta^2 \sigma^2}{k} + 2L^3 \eta^4 B^2.
\end{aligned}$$

□

Theorem 4.5.2 *Consider SGD iterations defined in (4.13) and satisfying elastic consistency bound (4.12). For a smooth non-convex objective function f, whose minimum x^* we are trying to find and the constant learning rate $\eta = \frac{\sqrt{k}}{\sqrt{T}}$, where $T \geq 64L^2 k$ is the number of iterations:*

$$\min_{t\in[T-1]} \mathbb{E}\|\nabla f(\vec{x}_t)\|^2 \le \frac{8(f(\vec{x}_0)-f(x^*))}{\sqrt{Tk}} + \frac{4B^2L^2k}{T} + \frac{8L\sigma^2}{\sqrt{Tk}} + \frac{16L^3B^2k\sqrt{k}}{T\sqrt{T}}.$$

Proof. Observe that since $\eta = \sqrt{k}/\sqrt{T} \le 1/8L$, by Lemma 4.5.1 we have that :

$$\mathbb{E}[f(\vec{x}_{t+1})] \le \mathbb{E}[f(\vec{x}_t)] - \frac{\eta}{8}\mathbb{E}\,\|\nabla f(\vec{x}_t)\|^2 + \frac{\eta^3B^2L^2}{2} + \frac{L\eta^2\sigma^2}{k} + 2L^3\eta^4B^2.$$

By summing the above inequality for $t = 0, 1, .., T-1$ we get that:

$$\sum_{t=0}^{T-1}\mathbb{E}[f(\vec{x}_{t+1})] \le \sum_{t=0}^{T-1}\Big(\mathbb{E}[f(\vec{x}_t)] - \frac{\eta}{8}\mathbb{E}\,\|\nabla f(\vec{x}_t)\|^2 + \frac{\eta^3B^2L^2}{2} + \frac{L\eta^2\sigma^2}{k} + 2L^3\eta^4B^2\Big).$$

Which can be rewritten as:

$$\sum_{t=0}^{T-1}\frac{\eta}{8}\mathbb{E}\,\|\nabla f(\vec{x}_t)\|^2 \le f(\vec{x}_0) - \mathbb{E}[f(\vec{x}_T)] + \frac{\eta^3B^2L^2T}{2} + \frac{L\eta^2\sigma^2T}{k} + 2L^3\eta^4B^2T.$$

Next, we divide by $\frac{\eta T}{8}$ and use the fact that $\mathbb{E}[f(\vec{x}_T)] \ge f(x^*)$:

$$\sum_{t=0}^{T-1}\frac{1}{T}\mathbb{E}\,\|\nabla f(\vec{x}_t)\|^2 \le \frac{8(f(\vec{x}_0)-f(x^*))}{\eta T} + 4\eta^2B^2L^2 + \frac{8L\eta\sigma^2}{k} + 16L^3\eta^3B^2.$$

By plugging in the value of η we get:

$$\begin{aligned}\min_{t\in[T-1]} \mathbb{E}\|\nabla f(\vec{x}_t)\|^2 &\le \sum_{t=0}^{T-1}\frac{1}{T}\mathbb{E}\,\|\nabla f(\vec{x}_t)\|^2 \le \frac{8(f(\vec{x}_0)-f(x^*))}{\eta T} + 4\eta^2B^2L^2 + \frac{8L\eta\sigma^2}{k} + 16L^3\eta^3B^2\\ &\le \frac{8(f(\vec{x}_0)-f(x^*))}{\sqrt{Tk}} + \frac{4B^2L^2k}{T} + \frac{8L\sigma^2}{\sqrt{Tk}} + \frac{16L^3B^2k\sqrt{k}}{T\sqrt{T}}.\end{aligned}$$

□

Theorem 4.3.1 follows by setting $k = 1$ and Theorem 4.3.2 follows by setting $k = n$.

4.5.2 Complete Proof of Convergence in the Strongly Convex Case

First, we show the proof of the Lemma which is known to be useful for the analysis of convergence of SGD in the strongly convex case.

Lemma 4.5.3 *Let f be a L-smooth, c-strongly convex function, whose minimum x^* we are trying to find. For any vector $\vec{x}$, we have:*

$$\langle \nabla f(\vec{x}), \vec{x} - x^*\rangle \ge \frac{1}{2L}\|\nabla f(\vec{x})\|^2 + \frac{c}{2}\|\vec{x} - x^*\|^2. \tag{4.15}$$

Proof.

$$\begin{aligned}0 &\le f(\vec{x} - \frac{1}{L}\nabla f(\vec{x})) - f(x^*) = f(\vec{x} - \frac{1}{L}\nabla f(\vec{x})) - f(\vec{x}) + f(\vec{x}) - f(x^*)\\ &\overset{(4.7)}{\le} \langle \nabla f(\vec{x}), -\frac{1}{L}\nabla f(\vec{x})\rangle + \frac{1}{2L}\|\nabla f(\vec{x})\|^2 + f(\vec{x}) - f(x^*)\\ &\overset{(4.8)}{\le} -\frac{1}{2L}\|\nabla f(\vec{x})\|^2 + \langle \nabla f(\vec{x}), \vec{x} - x^*\rangle - \frac{c}{2}\|\vec{x} - x^*\|^2.\end{aligned}$$

The proof of the lemma follows after rearranging the terms in the above inequality. □

Lemma 4.5.4 *Consider SGD iterations defined in (4.13) and satisfying elastic consistency bound (4.12). For a smooth, strongly convex objective function f and the constant learning rate $\eta \leq \frac{1}{4L}$. We have that:*

$$\mathbb{E}\,\|\vec{x}_{t+1} - x^*\|^2 \leq (1 - \frac{\eta c}{4})\,\mathbb{E}\,\|\vec{x}_t - x^*\|^2 + \frac{2\eta^3 L^2 B^2}{c} + \frac{2\eta^2\sigma^2}{k} + 4L^2\eta^4 B^2. \tag{4.16}$$

Proof. We condition on $\vec{x}_t$ and $\{\vec{v}_t^i | i \in I_t\}$ and calculate expectation with respect to the randomness of stochastic gradient(We call this $\mathbb{E}_s$). By descent lemma we get that :

$$\begin{aligned}
\mathbb{E}_s\,\|\vec{x}_{t+1} - x^*\|^2 &= \mathbb{E}_s\,\|\vec{x}_t - \frac{\eta}{k}\sum_{i\in I_t}\tilde{G}(\vec{v}_t^i) - x^*\|^2 \\
&= \|\vec{x}_t - x^*\|^2 - \frac{2\eta}{k}\sum_{i\in I_t}\mathbb{E}_s\langle\tilde{G}(\vec{v}_t^i), \vec{x}_t - x^*\rangle + \frac{\eta^2}{k^2}\mathbb{E}_s\,\|\sum_{i\in I_t}\tilde{G}(\vec{v}_t^i)\|^2 \\
&\overset{(4.4)}{=} \|\vec{x}_t - x^*\|^2 - \frac{2\eta|I_t|}{k}\langle\nabla f(\vec{x}_t), \vec{x}_t - x^*\rangle + \frac{2\eta}{k}\sum_{i\in I_t}\langle\nabla f(\vec{x}_t) - \nabla f(\vec{v}_t^i), \vec{x}_t - x^*\rangle \\
&\quad + \frac{\eta^2}{k^2}\mathbb{E}_s\,\|\sum_{i\in I_t}\Big(\tilde{G}(v_t^i) - \nabla f(\vec{v}_t^i) + \nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t) + \nabla f(\vec{x}_t)\Big)\|^2 \\
&\overset{Cauchy-Schwarz}{\leq} \|\vec{x}_t - x^*\|^2 - \frac{2\eta|I_t|}{k}\langle\nabla f(\vec{x}_t), \vec{x}_t - x^*\rangle \\
&\quad + \frac{2\eta^2}{k^2}\mathbb{E}_s\,\|\sum_{i\in I_t}\Big(\tilde{G}(v_t^i) - \nabla f(\vec{v}_t^i)\Big)\|^2 \\
&\quad + \frac{2\eta^2}{k^2}\mathbb{E}_s\,\|\sum_{i\in I_t}\Big(\nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t) + \nabla f(\vec{x}_t)\Big)\|^2.
\end{aligned}$$

Observe that $\mathbb{E}_s\left\|\sum_{i\in I_t}\Big(\tilde{G}(v_t^i) - \nabla f(\vec{v}_t^i)\Big)\right\|^2$ is the same as a variance of random variable $\sum_{i\in I_t}\tilde{G}(v_t^i)$. Recall that we are conditioning on $\vec{x}_t$ and $\{\vec{v}_t^i | i \in I_t\}$, hence $\tilde{G}(v_t^i)$ are independent random variables (because agents in I_t compute stochastic gradients independently). This means that because of the variance bound (4.5):

$$\mathbb{E}_s\left\|\sum_{i\in I_t}\Big(\tilde{G}(v_t^i) - \nabla f(\vec{v}_t^i)\Big)\right\|^2 \leq |I_t|\sigma^2.$$

Also by Cauchy-Schwarz inequality we get that

$$\begin{aligned}
\mathbb{E}_s\,\|\sum_{i\in I_t}\Big(\nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t) + \nabla f(\vec{x}_t)\Big)\|^2 &= \|\sum_{i\in I_t}\Big(\nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t) + \nabla f(\vec{x}_t)\Big)\|^2 \\
&\leq 2|I_t|\sum_{i\in I_t}\Big(\|\nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t)\|^2 + \|\nabla f(\vec{x}_t)\|^2\Big).
\end{aligned}$$

By combining the above three inequalities above we get:

$$\begin{aligned}
\mathbb{E}_s \|\vec{x}_{t+1} - x^*\|^2 &\overset{\text{Lemma 4.5.3}}{\leq} \|\vec{x}_t - x^*\|^2 - \frac{\eta|I_t|}{Lk}\|\nabla f(\vec{x}_t)\|^2 - \frac{\eta c|I_t|}{k} \\
&+ \frac{2\eta}{k}\sum_{i\in I_t}\langle \nabla f(\vec{x}_t) - \nabla f(\vec{v}_t^i), \vec{x}_t - x^*\rangle \\
&+ \frac{4\eta^2|I_t|}{k^2}\sum_{i\in I_t}\Big(\|\nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t)\|^2 + \|\nabla f(\vec{x}_t)\|^2\Big) + \frac{2\eta^2\sigma^2|I_t|}{k^2} \\
&\overset{Young}{\leq} \|\vec{x}_t - x^*\|^2 - \frac{\eta|I_t|}{Lk}\|\nabla f(\vec{x}_t)\|^2 - \frac{\eta c|I_t|}{k} \\
&+ \frac{2\eta}{ck}\sum_{i\in |I_t}\|\nabla f(\vec{x}_t) - \nabla f(\vec{v}_t^i)\|^2 + \frac{c\eta|I_t|}{2k}\|\vec{x}_t - x^*\|^2 \\
&+ \frac{4\eta^2|I_t|}{k^2}\sum_{i\in I_t}\Big(\|\nabla f(\vec{v}_t^i) - \nabla f(\vec{x}_t)\|^2 + \|\nabla f(\vec{x}_t)\|^2\Big) + \frac{2\eta^2\sigma^2|I_t|}{k^2} \\
&\overset{(4.7)}{\leq} (1 - \frac{\eta c|I_t|}{2k})\|\vec{x}_t - x^*\|^2 - \frac{\eta|I_t|}{Lk}\|\nabla f(\vec{x}_t)\|^2 + \frac{2\eta L^2}{ck}\sum_{i\in I_t}\|\vec{x}_t - \vec{v}_t^i\|^2 \\
&+ \frac{4\eta^2|I_t|}{k^2}\sum_{i\in I_t}\Big(L^2\|\vec{v}_t^i - \vec{x}_t\|^2 + \|\nabla f(\vec{x}_t)\|^2\Big) + \frac{2\eta^2\sigma^2|I_t|}{k^2} \\
&\leq (1 - \frac{\eta c|I_t|}{2k})\|\vec{x}_t - x^*\|^2 + \frac{2\eta L^2}{ck}\sum_{i\in I_t}\|\vec{x}_t - \vec{v}_t^i\|^2 + \frac{4\eta^2|I_t|}{k^2}\sum_{i\in I_t}\Big(L^2\|\vec{v}_t^i - \vec{x}_t\|^2\Big) \\
&+ \frac{2\eta^2\sigma^2}{k}.
\end{aligned}$$

Where, in the last inequality, we used $\eta \leq \frac{1}{4L}$ and $|I_t| \leq k$. Next, we use elastic consistency bound and $k/2 \leq |I_t| \leq k$:

$$\begin{aligned}
\mathbb{E}\|\vec{x}_{t+1} - x^*\|^2 &= \mathbb{E}[\mathbb{E}[\|\vec{x}_{t+1} - x^*\|^2 | \vec{x}_t, \{\vec{v}_t^i | i \in I_t\}]] \\
&\leq (1 - \frac{\eta c|I_t|}{2k})\mathbb{E}\|\vec{x}_t - x^*\|^2 + \frac{2\eta L^2}{ck}\sum_{i\in I_t}\mathbb{E}\|\vec{x}_t - \vec{v}_t^i\|^2 + \frac{4\eta^2|I_t|L^2}{k^2}\sum_{i\in I_t}\mathbb{E}\|\vec{v}_t^i - \vec{x}_t\|^2 + \frac{2\eta^2\sigma^2}{k} \\
&\leq (1 - \frac{\eta c}{4})\mathbb{E}\|\vec{x}_t - x^*\|^2 + \frac{2\eta^3 L^2 B^2}{c} + \frac{2\eta^2\sigma^2}{k} + 4L^2\eta^4 B^2.
\end{aligned}$$

□

Theorem 4.5.5 *Consider SGD iterations defined in (4.13) and satisfying elastic consistency bound (4.12). For a smooth, strongly convex function f, whose minimum x^* we are trying to find and the constant learning rate $\eta = \frac{2(\log T + \log k)}{cT}$, where $T \geq \frac{256L^2k}{c^2}$ is a number of iterations:*

$$\mathbb{E}\|\vec{x}_T - x^*\|^2 \leq \frac{\|\vec{x}_0 - x^*\|^2}{Tk} + \frac{16(\log T + \log k)^2 L^2 B^2}{c^4 T^2} + \frac{12\sigma^2(\log T + \log k)}{Tk} + \frac{48(\log T + \log k)^3 B^2 L^2}{c^4 T^3}.$$

Proof. Observe that since $\eta = \frac{2\log(Tk)}{Tc} \leq \frac{4\sqrt{k}}{\sqrt{T}c} \leq 1/4L$, By Lemma 4.5.4, we have that:

$$\mathbb{E}\|\vec{x}_{t+1} - x^*\|^2 \leq (1 - \frac{\eta c}{4})\mathbb{E}\|\vec{x}_t - x^*\|^2 + \frac{2\eta^3 L^2 B^2}{c} + \frac{2\eta^2\sigma^2}{k} + 4L^2\eta^4 B^2.$$

By using induction, it is easy to show that:

$$
\begin{aligned}
\mathbb{E}\|\vec{x}_T - x^*\|^2 &\leq (1-\frac{\eta c}{4})^T\|\vec{x}_0 - x^*\|^2 + \sum_{t=0}^{T-1}(1-\frac{\eta c}{4})^t\Big(\frac{2\eta^3L^2B^2}{c} + \frac{2\eta^2\sigma^2}{k} + 4L^2\eta^4B^2\Big) \\
&\leq (1-\frac{\eta c}{2})^T\|\vec{x}_0 - x^*\|^2 + \sum_{t=0}^{\infty}(1-\frac{\eta c}{4})^t\Big(\frac{2\eta^3L^2B^2}{c} + \frac{2\eta^2\sigma^2}{k} + 4L^2\eta^4B^2\Big) \\
&\leq e^{-\frac{\eta cT}{2}}\|\vec{x}_0 - x^*\|^2 + \frac{8\eta^2L^2B^2}{c^2} + \frac{8\eta\sigma^2}{kc} + \frac{16L^2\eta^3B^2}{c} \\
&= \frac{\|\vec{x}_0 - x^*\|^2}{Tk} + \frac{16(\log T + \log k)^2L^2B^2}{c^4T^2} + \frac{12\sigma^2(\log T + \log k)}{Tk} + \frac{48(\log T + \log k)^3B^2L^2}{c^4T^3}.
\end{aligned}
$$

□

Theorem 4.3.1 follows by setting $k = 1$ and Theorem 4.3.2 follows by setting $k = n$.

4.6 Elastic Consistency Bounds

4.6.1 Synchronous message passing with crash faults and variance bound

Consider the algorithm 6 for node i at iteration t. Recall that $\mathcal{L}_t^i$ is a set of nodes which send their computed gradients to node i at iteration t (for the convenience assume that $\mathcal{L}_{-1}^i = \mathcal{P}$). We assume that $i \in \mathcal{L}_t^i$. In the model with crash faults we have that if some node $j \notin \mathcal{L}_t^i$, then this means that node j has crashed. Further, if $j \in \mathcal{L}_{t-1}^i \backslash \mathcal{L}_t^i$, this means that j crashed during iteration t. More specifically this means that j generated it's stochastic gradient and crashed before sending it to i. In this case i uses it's own stochastic gradient $\widetilde{G}(\vec{v}_t^i)$ as a substitute. We define the auxiliary variable $\vec{x}_t$ as sum of all stochastic gradients which were

Algorithm 6 Iteration $t+1$ at node $i \in \mathcal{P}$, for Crash faults model with variance

```
Compute $\widetilde{G}(\vec{v}_t^i)$   % Compute SG using the local view.
Broadcast $\widetilde{G}(\vec{v}_t^i)$   % Broadcast SG to all the nodes.
$\widetilde{G} \leftarrow 0$   % Prepare to collect stochastic gradients from the nodes.
for $j \in \mathcal{L}_t^i$ do
    $\widetilde{G} \leftarrow \widetilde{G} + \widetilde{G}\left(\vec{v}_t^j\right)$
end for
for $j \in \mathcal{L}_{t-1}^i \backslash \mathcal{L}_t^i$ do
    $\widetilde{G} \leftarrow \widetilde{G} + \widetilde{G}(\vec{v}_t^i)$   % if $j$ crashed during iteration $t$, substitute it's SG.
end for
$\vec{v}_{t+1}^i \leftarrow \vec{v}_t^i - \frac{\eta}{n}\widetilde{G}$   % Update parameter vector.
```

generated up to and excluding iteration t and sent to at least one node multiplied by $-\frac{\eta}{n}$. This means that we use the update rule (4.11) with $I_t = \{j|\exists i, j \in \mathcal{L}_t^i\}$. Notice that since at most $n/2$ nodes can crash we have that $n/2 \leq |I_t| \leq n$. Let $f_t \leq f$ be the total number of crashed nodes up to and including iteration t. We proceed by proving the elastic consistency bound with $B = \frac{3f\sigma}{n}$:

Lemma 4.6.1 *In a synchronous message-passing system consisting of n nodes with failure bound $f \leq n/2$, For any iteration t and node i which has not crashed yet (it computes* *stochastic gradient at iteration t), if $\eta \leq \frac{1}{6L}$(Notice that this is consistent with the upper bound used in the convergence proofs) we have:*

$$\mathbb{E}\,\|\vec{v}_t^i - \vec{x}_t\|^2 \leq \frac{9\eta^2 f_t^2 \sigma^2}{n^2}. \tag{4.17}$$

Proof. We prove the claim by using induction on the number of iterations. Base case holds trivially. For the induction step assume that the lemma holds for iteration t: Let $h = |\mathcal{L}_t^i \backslash \mathcal{L}_{t+1}^i|$ be the number of nodes which crashed during iteration $t+1$ (we assume the worst case, which means that every node which crashed during iteration $t+1$ failed to send stochastic gradient to the node i). Assume that $h > 0$, since otherwise the proof is trivial. Notice that

$$\vec{x}_{t+1} - \vec{v}_{t+1}^i = \vec{x}_t - \vec{v}_t^i + \sum_{j \in \mathcal{L}_t^i \backslash \mathcal{L}_{t+1}^i} \frac{\eta}{n}(\widetilde{G}(\vec{v}_t^i) - \widetilde{G}(\vec{v}_t^j)) \tag{4.18}$$

Hence we get that

$$\begin{aligned}
\mathbb{E}\,\|\vec{x}_{t+1} - \vec{v}_{t+1}^i\|^2 &= \mathbb{E}\left\|\vec{x}_t - \vec{v}_t^i + \sum_{j \in \mathcal{L}_t^i \backslash \mathcal{L}_{t+1}^i} \frac{\eta}{n}(\widetilde{G}(\vec{v}_t^i) - \widetilde{G}(\vec{v}_t^j))\right\|^2 \\
&\overset{Young}{\leq} (1+\frac{h}{f_t})\,\mathbb{E}\,\|\vec{x}_t - \vec{v}_t^i\|^2 + (1+\frac{f_t}{h})\,\mathbb{E}\left\|\sum_{j \in \mathcal{L}_t^i \backslash \mathcal{L}_{t+1}^i} \frac{\eta}{n}(\widetilde{G}(\vec{v}_t^i) - \widetilde{G}(\vec{v}_t^j))\right\|^2 \\
&\overset{Cauchy-Schwarz}{\leq} (1+\frac{h}{f_t})\,\mathbb{E}\,\|\vec{x}_t - \vec{v}_t^i\|^2 + (1+\frac{f_t}{h})h\frac{\eta^2}{n^2}\sum_{j \in \mathcal{L}_t^i \backslash \mathcal{L}_{t+1}^i} \mathbb{E}\,\|\widetilde{G}(\vec{v}_t^i) - \widetilde{G}(\vec{v}_t^j)\|^2 \\
&= (1+\frac{h}{f_t})\,\mathbb{E}\,\|\vec{x}_t - \vec{v}_t^i\|^2 \\
&\quad + (1+\frac{f_t}{h})h\frac{\eta^2}{n^2}\sum_{j \in \mathcal{L}_t^i \backslash \mathcal{L}_{t+1}^i} \mathbb{E}\,\|\widetilde{G}(\vec{v}_t^i) - \nabla f(\vec{v}_t^i) + \nabla f(\vec{v}_t^i) - \nabla f(\vec{v}_t^j) + \nabla f(\vec{v}_t^j) - \widetilde{G}(\vec{v}_t^j)\|^2 \\
&\overset{Cauchy-Schwarz}{\leq} (1+\frac{h}{f_t})\,\mathbb{E}\,\|\vec{x}_t - \vec{v}_t^i\|^2 \\
&\quad + (1+\frac{f_t}{h})h\frac{\eta^2}{n^2}\sum_{j \in \mathcal{L}_t^i \backslash \mathcal{L}_{t+1}^i} \Big(3\,\mathbb{E}\,\|\widetilde{G}(\vec{v}_t^i) - \nabla f(\vec{v}_t^i)\|^2 + \\
&\qquad 3\,\mathbb{E}\,\|\nabla f(\vec{v}_t^i) - \nabla f(\vec{v}_t^j)\|^2 + 3\,\mathbb{E}\,\|\nabla f(\vec{v}_t^j) - \widetilde{G}(\vec{v}_t^j)\|^2\Big) \\
&\overset{(4.5)}{\leq} (1+\frac{h}{f_t})\,\mathbb{E}\,\|\vec{x}_t - \vec{v}_t^i\|^2 + (1+\frac{f_t}{h})h\frac{\eta^2}{n^2}\sum_{j \in \mathcal{L}_t^i \backslash \mathcal{L}_{t+1}^i} \Big(3\,\mathbb{E}\,\|\nabla f(\vec{v}_t^i) - \nabla f(\vec{v}_t^j)\|^2 + 6\sigma^2\Big) \\
&\overset{(4.7)}{\leq} (1+\frac{h}{f_t})\,\mathbb{E}\,\|\vec{x}_t - \vec{v}_t^i\|^2 + (1+\frac{f_t}{h})h\frac{\eta^2}{n^2}\sum_{j \in \mathcal{L}_t^i \backslash \mathcal{L}_{t+1}^i} \Big(3L^2\,\mathbb{E}\,\|\vec{v}_t^i - \vec{x}_t + \vec{x}_t - \vec{v}_t^j\|^2 + 6\sigma^2\Big) \\
&\overset{Cauchy-Schwarz}{\leq} (1+\frac{h}{f_t})\,\mathbb{E}\,\|\vec{x}_t - \vec{v}_t^i\|^2 \\
&\quad + (1+\frac{f_t}{h})h\frac{\eta^2}{n^2}\sum_{j \in \mathcal{L}_t^i \backslash \mathcal{L}_{t+1}^i} \Big(6L^2(\mathbb{E}\,\|\vec{v}_t^i - \vec{x}_t\| + \mathbb{E}\,\|\vec{x}_t - \vec{v}_t^j\|^2) + 6\sigma^2\Big)
\end{aligned}$$

Next, we use the assumption that lemma holds for nodes at iteration t. We get that:

$$\begin{aligned}\mathbb{E}\,\|\vec{x}_{t+1} - \vec{v}^i_{t+1}\|^2 &\leq (1 + \frac{h}{f_t})\frac{9\eta^2\sigma^2 f_t^2}{n^2} \\ &+ (1 + \frac{f_t}{h})h\frac{9\eta^2}{n^2}\sum_{j \in \mathcal{L}^i_t \setminus \mathcal{L}^i_{t+1}} \Big(12L^2\frac{\eta^2\sigma^2 f_t^2}{n^2} + 6\sigma^2\Big) \\ &= (1 + \frac{h}{f_t})\frac{9\eta^2\sigma^2 f_t^2}{n^2} + (1 + \frac{f_t}{h})h^2\frac{\eta^2}{n^2}\Big(12L^2\frac{9\eta^2\sigma^2 f_t^2}{n^2} + 6\sigma^2\Big)\end{aligned}$$

Finally, we use $f_t \leq f \leq n$ and $\eta \leq \frac{1}{6L}$, to get:

$$\begin{aligned}\mathbb{E}\,\|\vec{x}_{t+1} - \vec{v}^i_{t+1}\|^2 &\leq (1 + \frac{h}{f_t})\frac{9\eta^2\sigma^2 f_t^2}{n^2} + (1 + \frac{f_t}{h})h^2\frac{\eta^2}{n^2}\Big(3\sigma^2 + 6\sigma^2\Big) \\ &= \frac{9\eta^2\sigma^2}{n^2}(f_t + h)^2 = \frac{9\eta^2 f_{t+1}^2\sigma^2}{n^2}.\end{aligned}$$

□

4.6.2 Crash faults with second moment bound

Consider the algorithm 7 for node i at iteration t. Recall that $\mathcal{L}^i_t$ is a set of nodes which send their computed gradients to node i at iteration t. We assume that $i \in \mathcal{L}^i_t$. In the model with crash faults we have that if some node $j \notin \mathcal{L}^i_t$, then this means that node j has crashed.

Algorithm 7 Iteration $t+1$ at node $i \in \mathcal{P}$, for Crash faults model

```
Compute $\widetilde{G}(\vec{v}^i_t)$   % Compute SG using the local view.
Broadcast $\widetilde{G}(\vec{v}^i_t)$   % Broadcast SG to all the nodes.
$\widetilde{G} \leftarrow 0$   % Prepare to collect stochastic gradients from the nodes.
for $j \in \mathcal{L}^i_t$ do
    $\widetilde{G} \leftarrow \widetilde{G} + \widetilde{G}\left(\vec{v}^j_t\right)$
end for
$\vec{v}^i_{t+1} \leftarrow \vec{v}^i_t - \frac{\eta}{n}\widetilde{G}$   % Update parameter vector.
```

We define the auxiliary variable $\vec{x}_t$ as sum of all stochastic gradients which were generated up to and excluding iteration t and sent to at least one node multiplied by $-\frac{\eta}{n}$. This means that we use the update rule (4.11) with $I_t = \{j | \exists i, j \in \mathcal{L}^i_t\}$. Notice that since at most $n/2$ nodes can crash we have that $n/2 \leq |I_t| \leq n$. We proceed by proving the elastic consistency bound with $B = \frac{fM}{n}$:

Lemma 4.6.2 *In a synchronous message-passing system consisting of n nodes with failure bound $f \leq p/2$, For any iteration t and node i which has not crashed yet (it computes stochastic gradient at iteration t) we have:*

$$\mathbb{E}\,\|\vec{v}^i_t - \vec{x}_t\|^2 \leq \frac{\eta^2 f^2 M^2}{n^2} \tag{4.19}$$

Proof. Notice that $\vec{v}^i_t - \vec{x}_t = \sum_{s=0}^{t-1}\sum_{j \in I_s \setminus \mathcal{L}^i_s} \frac{\eta}{n}\widetilde{G}(\vec{v}^j_s)$, where $j \in I_s \setminus \mathcal{L}^i_s$ means that node j crashed at iteration s, before sending it's stochastic gradient to node i. Since each node can crash at most once, we have that $\sum_{s=0}^{t-1} |I_s \backslash \mathcal{L}_s^i| \leq f$. We get that:

$$\mathbb{E}\,\|\vec{v}_t^i - \vec{x}_t\|^2 = \mathbb{E}\,\|\sum_{s=0}^{t-1}\sum_{j\in I_s\backslash\mathcal{L}_s^i}\frac{\eta}{n}\widetilde{G}(\vec{v}_s^j)\|^2 \overset{Cauchy-Schwarz}{\leq} f\sum_{s=0}^{t-1}\sum_{j\in I_s\backslash\mathcal{L}_s^i}\frac{\eta^2}{n^2}\mathbb{E}\,\|\widetilde{G}(\vec{v}_s^j)\|^2$$
$$\overset{(4.6)}{\leq} f\sum_{s=0}^{t-1}\sum_{j\in I_s\backslash\mathcal{L}_s^i}\frac{\eta^2}{n^2}M^2 \leq \frac{\eta^2 f^2 M^2}{n^2}.$$

□

4.6.3 Synchronous message passing with message-omission failures

Algorithm 8 Iteration $t+1$ at node $i \in \mathcal{P}$, for message delays

```
Compute $\widetilde{G}(\vec{v}_t^i)$   % Compute SG using the local view.
Broadcast $\widetilde{G}(\vec{v}_t^i)$   % Broadcast SG to all the nodes.
$\widetilde{G} \leftarrow 0$   % Prepare to collect stochastic gradients from the nodes.
for $(s,j) \in \mathcal{L}_t^i$ do
    $\widetilde{G} \leftarrow \widetilde{G} + \widetilde{G}(\vec{v}_s^j)$
end for
$\vec{v}_{t+1}^i \leftarrow \vec{v}_t^i - \frac{\eta}{n}\widetilde{G}$   % Update parameter vector.
```

Consider algorithm 8. Here, we have that the set $\mathcal{L}_t^i$ might contain delayed messages (which were generated before the iteration t). Hence, we assume that $\mathcal{L}_t^i$ is a set of pairs (s,j) such that the stochastic gradient generated by the node j at iteration $s \leq t$ was delivered to the node i at iteration t. The important thing is that, at any iteration, the number of delayed messages (which might or might not be delivered in the future) is at most f. We assume that nodes always "send" stochastic gradients to themselves without any delay, that is, for any node i, $(t,i) \in \mathcal{L}_t^i$. Further, let $\mathcal{K}_t^i = \bigcup_{t'\leq t}\mathcal{L}_{t'}^i$. Notice that, $\vec{v}_t^i = -\sum_{(s,j)\in\mathcal{K}_{t-1}^i}\frac{\eta}{n}\widetilde{G}(\vec{v}_s^j)$. The auxiliary variable $\vec{x}_t = -\sum_{s=0}^{t-1}\sum_{j\in\mathcal{P}}\frac{\eta}{n}\widetilde{G}(\vec{v}_s^j)$. This means that we use the update rule (4.11) with $I_t = \mathcal{P}$. We following lemma proves the elastic consistency bound with $B = \frac{fM}{n}$:

Lemma 4.6.3 *In a synchronous message-passing system consisting of n nodes where message-omission failures can be upper bounded by f, we have that for any iteration t and node i:*

$$\mathbb{E}\,\|\vec{v}_t^i - \vec{x}_t\|^2 \leq \frac{\eta^2 f^2 M^2}{n^2} \tag{4.20}$$

Proof. For any iteration t, let $\mathcal{C}_t = \{0,1,...,t\} \times \mathcal{P}$, so that $\vec{x}_t = -\sum_{(s,j)\in\mathcal{C}_{t-1}}\frac{\eta}{n}\widetilde{G}(\vec{v}_s^j)$. Notice that $\vec{v}_t^i - \vec{x}_t = \sum_{(s,j)\in\mathcal{C}_{t-1}\backslash\mathcal{K}_{t-1}^i}\frac{\eta}{n}\widetilde{G}(\vec{v}_s^j)$. If $(s,j)\in\mathcal{C}_{t-1}\backslash\mathcal{K}_{t-1}^i$, then we have that the stochastic gradient generated by node j at iteration s, is still not delivered to node i before iteration t starts. Since this can happen to at most f messages, we have that $|\mathcal{C}_{t-1}\backslash\mathcal{K}_{t-1}^i| \leq f$. Hence, we get that:

$$\mathbb{E}\,\|\vec{v}_t^i - \vec{x}_t\|^2 = \mathbb{E}\left\|\sum_{(s,j)\in\mathcal{C}_{t-1}\backslash\mathcal{K}_{t-1}}\frac{\eta}{n}\widetilde{G}(\vec{v}_s^j)\right\|^2 \overset{Cauchy-Schwarz}{\leq} f\sum_{(s,j)\in\mathcal{C}_{t-1}\backslash\mathcal{K}_{t-1}}\frac{\eta^2}{n^2}\mathbb{E}\,\|\widetilde{G}(\vec{v}_s^j)\|^2$$
$$\overset{(4.6)}{\leq} f\sum_{(s,j)\in\mathcal{C}_{t-1}\backslash\mathcal{K}_{t-1}}\frac{\eta^2}{n^2}M^2 \leq \frac{\eta^2 f^2 M^2}{n^2}.$$

□

Removing the second moment bound. We can use the similar approach as in the case of crash faults and replace M in the elastic consistency bound with $O(\sigma)$. More precisely, for node i at iteration t, if stochastic gradient $\widetilde{G}(v_t^j)$ from node j is delayed node i uses it's own stochastic gradient $\widetilde{G}(v_t^i)$ and later corrects the error once it receives the actual gradient. In this case, error correction will ensure that

$$\mathbb{E}\left\|\vec{v}_t^i - \vec{x}_t\right\|^2 = \mathbb{E}\left\| \sum_{(s,j)\in\mathcal{C}_{t-1}\setminus\mathcal{K}_{t-1}} \frac{\eta}{n}(\widetilde{G}(\vec{v}_s^i) - \widetilde{G}(\vec{v}_s^j))\right\|^2.$$

and then we can use the similar approach as in the proof of Lemma 4.6.1 to show that if $\mathbb{E}\left\|\vec{v}_s^i - \vec{x}_s\right\|^2 \le \frac{8\eta^2 f^2\sigma^2}{n^2}$ for any $0 \le s < t$ and $\eta \le \frac{n}{5fL}$, then $\mathbb{E}\left\|\vec{v}_t^i - \vec{x}_t\right\|^2 \le \frac{8\eta^2 f^2\sigma^2}{n^2}$.

4.6.4 Asynchronous message passing

This case is very similar to the case with message-ommision failures. Consider algorithm 8 again. As before, $\mathcal{L}_t^i$ is a set of pairs (s,j) such that the stochastic gradient generated by the node j at iteration $t-\tau_{max} \le s \le t$ was delivered to the node i at iteration t(Recall that messages can be delayed by at most τ_{max} iterations). We assume that nodes always "send" stochastic gradients to themselves without any delay, that is, for any node i, $(t,i) \in \mathcal{L}_t^i$. Further, let $\mathcal{K}_t^i = \bigcup_{t'\le t}\mathcal{L}_{t'}^i$. Notice that, $\vec{v}_t^i = -\sum_{(s,j)\in\mathcal{K}_{t-1}^i}\frac{\eta}{n}\widetilde{G}\left(\vec{v}_s^j\right)$. The auxiliary variable $\vec{x}_t = -\sum_{s=0}^{t-1}\sum_{j\in\mathcal{P}}\frac{\eta}{n}\widetilde{G}\left(\vec{v}_s^j\right)$. This means that we use the update rule (4.11) with $I_t = \mathcal{P}$. We proceed by proving the elastic consistency bound with $B = \frac{\tau_{max}(n-1)M}{n}$:

Lemma 4.6.4 *In a asynchronous message-passing system consisting of n nodes and with delay bound τ_{max},we have that for any iteration t and node i:*

$$\mathbb{E}\left\|\vec{v}_t^i - \vec{x}_t\right\|^2 \le \frac{\eta^2(n-1)^2\tau_{max}^2 M^2}{n^2} \tag{4.21}$$

Proof. For any iteration t, let $\mathcal{C}_t = \{0,1,...,t\}\times\mathcal{P}$, so that $\vec{x}_t = -\sum_{(s,j)\in\mathcal{C}_{t-1}}\frac{\eta}{n}\widetilde{G}\left(\vec{v}_s^j\right)$. Notice that $\vec{v}_t^i - \vec{x}_t = \sum_{(s,j)\in\mathcal{C}_{t-1}\setminus\mathcal{K}_{t-1}^i}\frac{\eta}{n}\widetilde{G}(\vec{v}_s^j)$. If $(s,j)\in\mathcal{C}_{t-1}\setminus\mathcal{K}_{t-1}^i$, then we have that the stochastic gradient generated by node j at iteration s, is still not delivered to node i before iteration t starts. Since each message can be delayed by τ_{max} iterations at most, we have that for any $0 \le s \le t-1-\tau_{max}$ and node $j\in\mathcal{P}$, $(s,j)\in\mathcal{K}_{t-1}^i$. Also, we have that for any iteration $t-\tau_{max}\le s\le t-1$, $(s,i)\in\mathcal{K}_{t-1}^i$. This means that $|\mathcal{C}_{t-1}\setminus\mathcal{K}_{t-1}^i| \le (n-1)\tau_{max}$. We can finish the proof by using $f=(n-1)\tau_{max}$ and following exactly the same steps as in the proof of Lemma 4.6.3. □

Removing the second moment bound. Notice that for each agent the number of messages it has not received can be upper bounded by $\tau_{max}(n-1)$ at any step. Hence as in the case of message-omission failures we can prove that if $\eta \le \frac{n}{5(n-1)\tau_{max}L}$, then at any step t, $\mathbb{E}\left\|\vec{v}_t^i - \vec{x}_t\right\|^2 \le \frac{8\eta^2(n-1)^2\tau_{max}^2\sigma^2}{n^2}$.

4.6.5 Shared-Memory Systems

We consider a shared-memory system with n processors $\mathcal{P} = \{1,2,\dots,n\}$ that supports atomic `read` and `fetch&add` (`faa`). The parameter vector $\vec{x}\in\mathbb{R}^d$ is shared by the processes for concurrent lock-free read and write or update. The read/update at each of the indices $\vec{x}[i]$, $1\le i\le d$, of parameter, are atomic. By design, each process reads as well as writes over an inconsistent snapshot of $\vec{x}$, see [ADSK18]. We order the iterations of SGD by the atomic `faa` over the first index of $\vec{x}$. Note that, iterations by the processes are collectively ordered. Let q be the process, which computes stochastic gradient at iteration t, the inconsistent view (due to asynchrony) which is read by q at iteration t is denoted by $\vec{v}_t^q$. See Algorithm 9 for the formal description. In this case, the auxiliary variable $\vec{x}_t$ corresponds to

Algorithm 9 Iteration $t+1$, processor q.

```
for $1 \le i \le d$ do
    % Lock-free read
    $\vec{v}_t^q[i] \leftarrow \texttt{read}(\vec{x}[i])$
end for
Compute $\widetilde{G}\left(v_t^q\right)$
for $1 \le i \le d$ do
    % Lock-free Update
    $\texttt{faa}(\vec{x}[i], \eta\widetilde{G}\left(\vec{v}_t^q\right)[i])$
end for
```

the sum of first t stochastic gradients (according to the ordering described above) multiplied by $-\eta$. Hence, we use the update rule (4.10): $\vec{x}_{t+1} = \vec{x}_t - \eta\widetilde{G}(\vec{v}_t^q)$.

At iteration t, let τ_t^i be the delay in the stochastic gradient update at an arbitrary index $1 \le i \le d$, which essentially means that $\vec{v}_t^q[i] = \vec{x}_{t-\tau_t^i}[i]$. Let $\tau_t = \max\{\tau_t^1, \tau_t^2, ..., \tau_t^d\}$. It is standard to assume that for any $t \ge 0$, τ_t is upper bounded by τ_{max} [RRWN11, LHLL15, SZOR15, ADSK18]. Drawing from the literature of shared-memory distributed computing, considering an iteration as an operation, τ_{max} is essentially the maximum number of concurrent operations during the lifetime of any operation in the system, which is defined as *interval contention* [AAF$^+$99]. With these specifications in place, Lemma 4.6.5 shows that the shared-memory asynchronous SGD scheme satisfies elastic consistency with $B = \sqrt{d}\tau_{max}M$.

Lemma 4.6.5 *Given an asynchronous shared-memory system with maximum delay bound $\tau_{\max}$, we have that for processor q which generates stochastic gradient at iteration $t+1$.* $\mathbb{E}\|\vec{x}_t - \vec{v}_t^q\|^2 \le d\tau_{max}^2\eta^2M^2$

Proof. The 1-norm of the difference between the inconsistent snapshots $\vec{x}_t$ and $\vec{v}_t$ is bounded as follows:

$$\begin{aligned}\|\vec{x}_t - \vec{v}_t^q\|_1 &\le \sum_{j=1}^{\tau_{max}} \|\vec{x}_{t-j+1} - \vec{x}_{t-j}\|_1 \\ &\le \sqrt{d}\sum_{j=1}^{\tau_{max}} \|\vec{x}_{t-j+1} - \vec{x}_{t-j}\| \text{ (as } \|\vec{x}\|_1 \le \sqrt{d}\,\|\vec{x}\|\,,\ \text{for } \vec{x} \in \mathbb{R}^d).\end{aligned}$$

Then,

$$\begin{aligned}\|\vec{x}_t - \vec{v}_t^q\|^2 &\le \|\vec{x}_t - \vec{v}_t\|_1^2 \text{ (using } \|\vec{x}\| \le \|\vec{x}\|_1\,,\ \text{for } \vec{x} \in \mathbb{R}^d) \\ &\le \left(\sqrt{d}\sum_{j=1}^{\tau_{max}} \|\vec{x}_{t-j+1} - \vec{x}_{t-j}\|\right)^2 \\ &\overset{Cauchy-Schwarz}{\le} d\tau_{max}\sum_{j=1}^{\tau_{max}} \|\vec{x}_{t-j+1} - \vec{x}_{t-j}\|^2.\end{aligned}$$

Fix t and $1 \le j \le \tau_{max}$. Recall that $\vec{x}_{t-j+1} - \vec{x}_{t-j} = -\eta\widetilde{G}(\vec{v}_{t-j}^r)$, for some processor r, which computed stochastic gradient at iteration $t-j$. Hence,

$$\mathbb{E}\,\|\vec{x}_{t-j+1} - \vec{x}_{t-j}\|^2 = \eta^2\,\mathbb{E}\,\|\widetilde{G}(\vec{v}_{t-j}^r)\|^2 \overset{(4.6)}{\le} \eta^2M^2. \tag{4.22}$$

Thus, by combining the above two inequalities we get that:

$$\mathbb{E}\,\|\vec{x}_t - \vec{v}^q_t\|^2 \leq d\tau_{max} \sum_{j=1}^{\tau_{max}} \mathbb{E}\,\|\vec{x}_{t-j+1} - \vec{x}_{t-j}\|^2 \leq d\tau_{max}^2 \eta^2 M^2.$$

□

4.6.6 Communication-Efficient Methods

In this section, we consider a *synchronous* message-passing system of n nodes $\mathcal{P} = \{1, 2, \ldots, n\}$ executing SGD iterations. For simplicity, we assume that the system is *synchronous and fault-free*, In each iteration nodes broadcast a *compressed* version of the stochastic gradient, they computed, in order to reduce communication costs. Communication-efficiency can be achieved in two ways: for a computed stochastic gradient, (a) quantization: broadcast a quantized vector that requires fewer bits [SFJY14, AGL+17], or, (b) sparsification: broadcast a sparse vector [AHJ+18, SCJ18].

Clearly, this strategy leads to losses in the parameter updates at each iteration. A popular approach to control this error is using *error-feedback*: the accumulated residual error from the previous broadcasts is added to the stochastic gradient at the current iteration before applying quantization or sparsification. We will show that the residual error can be modelled in the context of elastic consistency, and that it stays bounded in the case of popular communication-efficient techniques, which implies their convergence.

In the description below, we will be using the term *lossy compression* collectively for both quantization and sparsification. A vector $\vec{x}$ which undergoes a lossy compression will be called a *compressed* vector, it's compressed value will be denoted by $Q(\vec{x})$.

Algorithm 10 Iteration $t+1$ at a node $i \in \mathcal{P}$.

```
1: Compute $\widetilde{G}\,(\vec{v}^i_t)$   % Compute SG using local view
2: Compute $\vec{w}^i_t \leftarrow \vec{\epsilon}^i_t + \eta\widetilde{G}(\vec{v}^i_t)$   % Add the accumulated error to the computed SG.
3: Compute and Broadcast $Q(\vec{w}^i_t)$   % Apply lossy compression and broadcast.
4: Compute $\vec{\epsilon}^i_{t+1} \leftarrow \vec{w}^i_t - Q(\vec{w}^i_t)$   % Update the accumulated error.
5: $\widetilde{G} \leftarrow 0$
6: for each $j \in \mathcal{P}$ (including $i$) do
7:     Receive $Q(\vec{w}^j_t)$; $\widetilde{G} \leftarrow \widetilde{G} + Q(\vec{w}^j_t)$
8: end for
9: $\vec{v}^i_{t+1} \leftarrow \vec{v}^i_t - \frac{\widetilde{G}}{n}$   Update parameter vector.
```

A typical iteration at a node i is shown in Algorithm 10. The algorithm works as follows. Each node i maintains a local model $\vec{v}^i_t$, and a local error accumulation ϵ^i_t, starting from $\vec{v}^i_t = \vec{0}^d = \vec{\epsilon}^i_t$.

At each iteration, a node i computes a stochastic gradient with respect to its local view $\vec{v}^i_t$, adds to it the accumulated error from the previous iterations, applies a lossy compression function Q to the result, and broadcasts to the parameter servers. The error accumulation is updated to reflect the lossy compression, see Line 4. The compression Q provably satisfies the following:

$$\|Q(\vec{w}) - w\|^2 \leq \gamma\|\vec{w}\|^2,\ \forall \vec{w} \in \mathbb{R}^d, \text{ and } 0 \leq \gamma < 1. \tag{4.23}$$

In the above inequality parameter γ, depends on the compression scheme that we use.

The error-feedback strategy particularly helps in asymptotically compensating for the biased stochastic gradient updates. However, if the compression scheme is unbiased, e.g. QSGD [AGL+17], the convergence theory works even without the error-feedback. For such a method, line 2 in Algorithm 10 changes to $\vec{w}_t^i \leftarrow \eta\widetilde{G}(\vec{v}_t^i)$; all other steps remain unchanged. Our discussion in this sub-section focuses on methods with error-feedback.

The local view of parameter $\vec{v}_t^i$ is inconsistent in the sense that it is updated with compressed stochastic gradients. To model this in the context of elastic consistency, we define the auxiliary parameter $\vec{x}_t$, as sum of all stochastic gradients generated by the algorithm up to and excluding iteration t, multiplied by $-\frac{\eta}{n}$. Hence, we use the update rule (4.11) with $I_t = \{1, 2, ..., n\} = \mathcal{P}$:

$$\vec{x}_{t+1} = \vec{x}_t - \frac{\eta}{n}\sum_{i\in\mathcal{P}}\widetilde{G}(\vec{v}_t^i). \tag{4.24}$$

The local view of each node $q \in n$ is updated as:

$$\vec{v}_{t+1}^q = \vec{v}_t^q - \frac{1}{n}\sum_{i\in\mathcal{P}} Q(\eta\widetilde{G}(\vec{v}_t^i) + \epsilon_t^i). \tag{4.25}$$

Using induction on the number of iterations, it is easy to show that for any $t \geq 0$ and node q:

$$\vec{v}_t^q - \vec{x}^t = \frac{1}{n}\sum_{i\in\mathcal{P}}\epsilon_t^i. \tag{4.26}$$

Thus, using the above equation we can derive the elastic consistency bounds.

Lemma 4.6.6 *For any node q and iteration $t \geq 0$. We have that*

$$\mathbb{E}\,\|\vec{x}_t - \vec{v}_t^q\|^2 \leq \frac{(2-\gamma)\gamma M^2\eta^2}{(1-\gamma)^3}. \tag{4.27}$$

Proof.

we start with

$$\sum_{i\in\mathcal{P}}\|\epsilon_{t+1}^i\|^2 = \|Q(\eta\widetilde{G}(\vec{v}_t^i) + \epsilon_t^i) - (\eta\widetilde{G}(\vec{v}_t^i) + \epsilon_t^i)\|^2 \overset{(4.23)}{\leq} \gamma\sum_{i\in\mathcal{P}}\|\eta\widetilde{G}(\vec{v}_t^i) + \epsilon_t^i\|^2$$
$$\overset{Young}{\leq} (1+1-\gamma)\gamma\sum_{i\in\mathcal{P}}\|\epsilon_t^i\|^2 + (1+\frac{1}{1-\gamma})\eta^2\gamma\sum_{i\in\mathcal{P}}\|\widetilde{G}(\vec{v}_t^i)\|^2$$

□

Next, using induction on the number of iterations, we upper bound $\sum_{i\in\mathcal{P}}\mathbb{E}\,\|\epsilon_t^i\|^2$ by $\frac{(2-\gamma)\gamma M^2\eta^2 n}{(1-\gamma)^3}$. The base case holds trivially. For the induction step we assume that the statement holds for t. We get that:

$$\sum_{i\in\mathcal{P}}\mathbb{E}\,\|\epsilon_{t+1}^i\|^2 \leq (1+1-\gamma)\gamma\sum_{i\in\mathcal{P}}\mathbb{E}\,\|\epsilon_t^i\|^2 + (1+\frac{1}{1-\gamma})\eta^2\gamma\sum_{i\in\mathcal{P}}\mathbb{E}\,\|\widetilde{G}(\vec{v}_t^i)\|^2$$
$$\overset{(4.6)}{\leq} (1+1-\gamma)\gamma\sum_{i\in\mathcal{P}}\mathbb{E}\,\|\epsilon_t^i\|^2 + (1+\frac{1}{1-\gamma})\eta^2\gamma M^2 n$$
$$\leq \gamma\eta^2 M^2 n\left(\frac{(2-\gamma)^2\gamma}{(1-\gamma)^3} + 1 + \frac{1}{1-\gamma}\right) = \frac{(2-\gamma)\gamma M^2\eta^2 n}{(1-\gamma)^3}.$$

This gives us that

$$\mathbb{E}\,\|\vec{x}_t - \vec{v}_t^q\|^2 \overset{(4.26)}{=} \mathbb{E}\,\|\frac{1}{n}\sum_{i\in\mathcal{P}}\epsilon_t^i\|^2 \overset{Cauchy-Schwarz}{\leq} \frac{1}{n}\sum_{i\in\mathcal{P}}\mathbb{E}\,\|\epsilon_t^i\|^2 \leq \frac{(2-\gamma)\gamma M^2\eta^2}{(1-\gamma)^3}.$$

In the following, we show that TopK and One-Bit quantization schemes satisfy elastic consistency bounds.

TopK quantization The *TopK* algorithm [Str15] presents a sparsification scheme for the stochastic gradients. Essentially, we select the top K of the indices of $\vec{w}$ sorted by their absolute value. Because $d-K$ indices of a vector, which are not the top ones by their absolute value, are discarded in this method, it clearly satisfies inequality (4.23) for $\gamma = \frac{d-K}{d}$. Hence, TopK quantization satisfies elastic consistency bound with constant $B = \sqrt{\frac{(1-K/d)(1+K/d)}{(K/d)^3}}M = \sqrt{\frac{d}{K}(\frac{d^2}{K^2}-1)}M$.

One-Bit quantization The one-bit SGD quantization was first described in [SFJY14]. We use the notation $[\vec{x}]_i$ to denote the i'th component of $\vec{x}$. Consider the vector $\vec{w}$ and let $S^+(\vec{w})$ be its index of positive components and $S^-(\vec{w})$ that of its negative components, i.e., $S^+(\vec{w}) = \{i : [\vec{w}]_i \geq 0\}$ and $S^-(\vec{w}) = \{i : [\vec{w}]_i < 0\}$. Then let $\overline{w}^+ = \frac{1}{|S^+(\vec{w})|}\sum_{i\in S^+(\vec{w})}[\vec{w}]_i$ and $\overline{w}^- = \frac{1}{|S^-(\vec{w})|}\sum_{i\in S^-(\vec{w})}[\vec{w}]_i$. Now define the one-bit quantization operation $Q(\cdot)$ as,

$$[Q(\vec{w})]_i = \begin{cases} \overline{w}^+ & \text{for } i \in S^+(\vec{w}). \\ \overline{w}^- & \text{for } i \in S^-(\vec{w}). \end{cases} \tag{4.28}$$

It easy to verify that one-bit quantization satisfies inequality (4.23) for $\gamma = 1 - \frac{1}{d}$. Hence, One-bit quantization satisfies elastic consistency bounds with $B = \sqrt{d(d^2-1)}M$.

4.7 Related Work

Distributed machine learning has recently gained significant practical adoption, e.g. [DCM+12, HCC+13, CSAK14, ZCL15, XHD+15, JWG+19, PZC+19]. Consequently, there has been significant work on introducing and analyzing distributed relaxations of SGD [MPP+15, RRWN11, HCC+13, SZOR15, LHLL15, CDR15, LPLJ16, ADSK18, WJ21, WWS+18, KRSJ19, KRSJ19, LNDS20]. Due to space constraints, we cover in detail only work that is technically close to ours.

Specifically, De Sa et al. [SZOR15] were the first to consider a unified analysis framework for asynchonous and communication-compressed iterations. Relative to it, our framework improves in three respects: (i) it does not require stringent gradient sparsity assumptions; (ii) it is also able to analyze the case where the updates are not unbiased estimators of the gradient, which allows extensions to error-feedback communication-reduction; and (3) it also tackles convergence for general non-convex objectives. Reference [LHLL15] presented the first general analysis of asynchronous non-convex SGD , without communication-reduction. Qiao et al. [QAZX19] model asynchrony and communication reduction as perturbations of the SGD iteration, and introduce a metric called "rework cost," which can be subsumed into the elastic consistency bound.

Karimireddy et al. [KRSJ19] analyze communication-compression with error feedback, and present a general notion of δ-compressor to model communication-reduced consistency relaxations; later, the framework was extended to include *asynchronous* iterations [KRSJ19]. Every method satisfying the δ-compressor property is elastically-consistent, although the converse is not true. Relative to this work, our framework generalizes in one important practical aspect, as it allows the analysis in *distributed* settings: [KRSJ19, KRSJ19] assume that the iterations are performed at a single processor, which may compress gradients or view inconsistent information only with respect to *its own* earlier iterations. This extension is non-trivial; tackling this more realistic setting previously required additional analytic assumptions [AHJ+18]. Sparsified methods would not be competitive with our elastic scheduler, since they only impose sparsity to reduce communication, which would not necessarily improve scheduling.

We have proposed a new and fairly general framework for analyzing inconsistent SGD iterations. Its main advantages are *generality* and *simplicity*. Inspired by this technical condition, we introduce two new, efficient scheduling mechanisms. More generally, we believe that elastic consistency could inspire new distributed algorithms, and be used to derive convergence in a streamlined manner. One key line of extension which we plan to pursue is to study whether elastic consistency can be extended to other first-order distributed optimization methods, or zeroth- or second-order methods.

CHAPTER 5

Asynchronous Decentralized SGD with Quantized and Local Updates

5.1 Introduction

Decentralized optimization has recently emerged as a promising approach for scaling the distributed training of machine learning models, in particular via stochastic gradient descent (SGD) [LZZ$^+$17, TZG$^+$18, KLSJ20]. Its key advantage is that it removes the need for a central coordinator node in distributed training, and therefore it can allow for extremely high scaling.

The general decentralized optimization setting is the following: we are given n nodes, each with a subset of data from some distribution, which can communicate over some underlying graph topology. In each global round, each node samples some local data, performs a local gradient step, and it is paired with a neighbor, which may be chosen randomly. The nodes exchange model information pairwise, and then update their models, often via direct model averaging. Variants of this setting have been analyzed since pioneering work by [Tsi84], for various estimation and optimization algorithms [XB04, NO09, JRJ09, SS14] and have seen renewed interest given its applicability to training deep neural networks (DNNs) at scale, e.g. [LZZ$^+$17, LZZL18, ALBR18].

Recently, there has been significant focus on reducing the *synchronization overheads* for decentralized training, usually employing three approaches: 1) implementing faster *non-blocking communication* between communication partners at a round [LZZL18, ALBR18], which may cause them to see stale versions of their models, 2) allowing nodes to take *local steps* in between their communication rounds [WJ21, KLB$^+$20], and 3) applying *quantization* to the communication [LDS20, TZG$^+$18, KLSJ20], .

The above impressive line of work contributes a rich set of algorithmic and analytic ideas; however, one common limitation is that the algorithms are usually set in the *synchronous gossip* model, which requires all nodes to perform their communication in lock-step rounds, and share a common notion of time, thus reducing their practicality. To mitigate this fact, some references, e.g. [LZZL18, ALBR18, LDS20] partially relax this requirement, although they do so at the cost of additional assumptions, or reduced guarantees, as we discuss in related work. Another relative limitation is that the analyses are usually customized to the bespoke communication-reduced methods being applied, and therefore are hard to generalize to other methods.

Our Contribution. In this chapter, we consider decentralized SGD-based optimization in the simpler, but harder to analyze, *asynchronous gossip* model [XB04], in which communication occurs in discrete, randomly chosen pairings among nodes, and thus does not require a common notion of time. We prove that a new variant of SGD we call *SwarmSGD* still converges in this setting, even though it supports all three communication-reduction approaches mentioned above *in conjunction*. Specifically, supporting both *non-blocking communication and quantization* requires new insights. In addition, our analysis generalizes to heterogeneous data distributions and communication topologies.

At a high level, SwarmSGD works as follows. Each node i maintains a local model estimate X_i based on which gradients are generated, and a shared buffer where quantized models are stored for communication with other nodes. In each step, node i first computes a sequence of H local gradient steps, which it does not yet apply. Next, the node chooses communication partner j, uniformly at random among its neighbors. Then, node i reads from its own communication buffer and from the communication buffer of j, obtaining quantized models Q_i and Q_j (a subtlety here is that Q_i is not necessarily the quantized version of the model X_i, since other nodes can write concurrently to i's buffer). The node i then averages Q_i with Q_j, and updates the neighbor's remote buffer to the quantized average. Finally, it applies its local gradient steps to the resulting average, adopts this as its next model X_i, and a writes quantized version of it in its own shared buffer. This procedure can be implemented in a deadlock-free, non-blocking manner, by using either shared-memory or the remote direct-memory access (RDMA) calls supported by MPI [WSB+06]. Importantly, the communication partner j does not need to block its computation during communication, and may be contacted by more than one interaction partner during a single local step, although we do assume that individual reads and writes are performed atomically.

A key component of this procedure is the *quantization scheme*: directly using an unbiased quantizer, e.g. [AGL+17] would destroy convergence guarantees, as the quantization error would be proportional to the model norm, which may not be bounded. Instead, we use a customized variant of the lattice-based quantization scheme of [DGM+21], which has the property that its error depends on the distance between the *point being quantized* (the model), and an arbitrary *reference point*, which is provided as a parameter to the quantization. One of our key technical insights is that each node can reliably use *its own model* as a *reference point* to quantize and de-quantize messages placed in its buffer by other nodes. In turn, this requires non-trivial care when analyzing the quantization.

Specifically, the key result behind our analysis is exactly in showing that the nodes' local models stay well-enough concentrated around their mean throughout optimization to allow for correct decoding of quantized models, which in turn implies joint convergence by the nodes towards a point of vanishing gradient. This concentration follows via a non-trivial super-martingale argument. If nodes take a constant number of local SGD steps on average between communication steps, then SwarmSGD has $\Theta(\sqrt{n})$ speedup to convergence for non-convex objectives. This matches results from previous work which considered decentralized dynamics but with global synchronization [LZZ+17]. Our analysis also extends to heterogenenous graph topologies, and data distributions.

5.2 Preliminaries

The Distributed System Model. We consider a model which consists of $n \geq 2$ nodes, each of which is able to perform local computation. We assume that communication network of nodes is a graph G with spectral gap λ_2, which denotes the second smallest eigenvalue of the Laplacian of G. Let ρ_{max}, ρ_{min} be the maximum and minimum degrees in G, respectively. We will focus on densely-connected topologies, which model supercomputing and cloud networks: for instance, the standard Dragonfly topology [KDSA08, BH14] is regular, densely connected and low-diameter, mimicking regular expanders.

The execution is modelled as occurring in discrete *steps*, where in each step a new node (the "initiator") is sampled, and can then contact one of its neighbors (the "responder") uniformly at random. (At the algorithm level, the initiator is "sampled" once it completes its current computational step, and seeks to interact with a neighbor.) We denote the number of steps for which we run by T. Globally, the communication steps can be seen as a sequence of sampled *directed* communication edges. Thus, the basic unit of time is a single pairwise interaction between two nodes. Notice however that in a real system $\Theta(n)$ of these interactions could occur in parallel. Thus, the standard global time measure is *parallel time*, defined as the total number of interactions divided by n, the number of nodes. Parallel time intuitively corresponds to the *average* number of interactions per node until convergence. This model is identical to the asynchronous gossip model [XB04], and to the population protocol model [AAD+06].

Stochastic Optimization. We assume that the agents wish to jointly minimize a d-dimensional, differentiable function $f : \mathbb{R}^d \to \mathbb{R}$. Specifically, we will assume the empirical risk minimization setting, in which agents are given access to a set of m data samples $S = \{s_1, \ldots, s_m\}$ coming from some underlying distribution $\mathcal{D}$, and to functions $\ell_i : \mathbb{R}^d \to \mathbb{R}$ which encode the loss of the argument at the sample s_i. The goal of the agents is to converge on a model x^* which minimizes the empirical loss over the m samples, that is $x^* = \text{argmin}_x f(x) = \text{argmin}_x (1/m)\sum_{i=1}^m \ell_i(x)$. We assume that each agent i has a local function f_i associated to its fraction of the data, i.e $\forall x \in \mathbb{R}^d$: $f(x) = \sum_{i=1}^n f_i(x)/n$.

Agents employ these samples to run a decentralized variant of SGD, described in detail in the next section. For this, we will assume that each agent i has access to *unbiased stochastic gradients* $\widetilde{g}_i$ of the function f_i, which are functions such that $\mathbb{E}[\widetilde{g}_i(x)] = \nabla f_i(x)$. Stochastic gradients can be computed by each agent by sampling i.i.d. the distribution $\mathcal{D}$, and computing the gradient of f at θ with respect to that sample. Our analysis also extends to the case where each agent is sampling from its own partition of data. We assume the following conditions about the objective function, although not all our results require the second moment bound:

1. **Smooth Gradients**: The gradient $\nabla f_i(x)$ is L-Lipschitz continuous for some $L > 0$, i.e. for all $x, y \in \mathbb{R}^d$ and agent i:
$$\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x - y\|. \tag{5.1}$$
2. **Bounded Variance**: The variance of the stochastic gradients is bounded by some $\sigma^2 > 0$, i.e. for all $x \in \mathbb{R}^d$ and agent i:
$$\mathbb{E}\left\|\widetilde{g}_i(x) - \nabla f_i(x)\right\|^2 \leq \sigma^2. \tag{5.2}$$
3. **Bounded Local Function Variance**: There exists $\varsigma^2 > 0$, such that for all $x \in \mathbb{R}^d$:
$$\sum_{i=1}^n \frac{\left\|\nabla f(x) - \nabla f_i(x)\right\|^2}{n} \leq \varsigma^2. \tag{5.3}$$

4. **Bounded Second Moment**: The second moment of the stochastic gradients is bounded by some $M^2 > 0$, i.e. for all $x \in \mathbb{R}^d$ and agent i:
$$\mathbb{E}\left\|\widetilde{g}_i\left(x\right)\right\|^2 \leq M^2. \tag{5.4}$$

Note that throughout this chapter for any random variable X, by $\mathbb{E}\left\|X\right\|^2$ we mean $\mathbb{E}[\|X\|^2]$.

Each node has a communication buffer, which, for simplicity, we assume can be read and written atomically by each node; Importantly, buffers can only hold *quantized* vectors.

Quantization Procedure. We use a quantization function which follows from Lemma 23 in (the full version of) [DGM+21].

Corollary 5.2.1 *(Quantization for Communication Buffers) Fix parameters R and $\epsilon > 0$. There exists a quantization procedure defined by an encoding function $Enc_{R,\epsilon} : \mathbb{R}^d \to \{0,1\}^*$ and a decoding function $Dec_{R,\epsilon} = \mathbb{R}^d \times \{0,1\}^* \to \mathbb{R}^d$ such that, for any vector $x \in \mathbb{R}^d$ which we are trying to quantize, and any vector y which is used by decoding, which we call the* decoding key, *if $\|x-y\| \leq R^{R^d}\epsilon$ then with probability at least $1 - \log\log(\frac{\|x-y\|}{\epsilon})O(R^{-d})$, the function $Q_{R,\epsilon}(x) = Dec_{R,\epsilon}(y, Enc_{R,\epsilon}(x))$ has the following properties:*

1. *(Unbiased decoding)* $\mathbb{E}[Q_{R,\epsilon}(x)] = \mathbb{E}[Dec_{R,\epsilon}(y, Enc_{R,\epsilon}(x))] = x$;
2. *(Error bound)* $\|Q_{R,\epsilon}(x) - x\| \leq (R^2+7)\epsilon$;
3. *(Communication bound) To compute $Dec_{R,\epsilon}(y, Enc_{R,\epsilon}(x))$, only the first B bits of $Enc_{R,\epsilon}(x)$ are needed, where $B = O\left(d\log(\frac{R}{\epsilon}\|x-y\|)\right)$.*

Proof. Lemma 23 of the full version of [DGM+21] provides similar guarantees as the ones we want to prove, but they assume interactive message-passing communication between an encoding node u and a decoding node v. However, in their setting, the messages sent by u are *non-adaptive*: u simply sends quantizations using an increasing number of bits, until v replies confirming that it has decoded successfully. The number of bits sent during communication is upper bounded by $O\left(d\log(\frac{R}{\epsilon}\|x-y\|)\right)$, where x is a vector node u is sending and y is vector node v is using for decoding. In our setting, we use communication buffers which, so node u can simply append all of its potential messages together as $Q_{R,\epsilon}(x)$.

Critically, notice that node u should append enough bits so that the decoding is possible (Since in our setting there is no way for v to acknowledge that it received enough number of bits). This can be done in two ways. If u knows the distance between x and y. then u can simply write $O\left(d\log(\frac{R}{\epsilon}\|x-y\|\right)$ bits in the register.

In the second case, u does not know the distance. Let T be the total number of times nodes communicate throughout our algorithm. We will show that with high probability all distances between encoded and decoding vectors will be at most $\frac{\epsilon T^{17}}{R}$ (dependence on T stems from the fact that we wish to show an upper bound with high probability, please see Lemma 5.5.19), and therefore at most $O(d\log T)$ bits for quantization will suffice in the worst case. Thus, the node writes $O(d\log T)$ bits in the register , but when v tries to decode, it does not need all those bits: it reads and uses only the first $O\left(\log(\frac{R}{\epsilon}\|x-y\|\right)$ bits.

Counting Communication Cost. We emphasize that, when we calculate the number of bits needed by quantization we actually aim to measure the number of bits *exchanged between* u and v. In the setting we consider, which has local registers/communication buffers, this is the number of bits spent to read from (or to write to) the non-local register. Since the second case above involves writing a relatively large number of bits, we will use it only when u is writing a quantized value to its *own* register/buffer, and so does not need to communicate the bits. Then, only the $O\left(\log(\frac{R}{\epsilon}\|x-y\|\right)$ bits read by v need to be communicated.

To summarize, in our algorithm we will always ensure that whenever some node u writes a quantized value, it either knows the key which will be used for decoding it, or is writing to its local register. In the second case, we have to guarantee that $O(d \log T)$ bits suffice in the worst case. That is, we will have to show that $\|x-y\| = \frac{\epsilon \cdot poly(t)}{R}$. In the first case, there are no restrictions.

□

5.3 The SwarmSGD Algorithm

We now describe a decentralized variant of SGD, designed to be executed by a population of n nodes, interacting over the edges of communication graph G. We assume that the largest degree of a vertex in G is ρ_{max} and the smallest one is ρ_{min}, additionally we assume that the second smallest eigenvalue of the Laplacian matrix of G, is λ_2. As noted above, the algorithm proceeds in individual communication steps, where in each step a node which has completed its local computation, seeks a random neighbor to communicate with. We will alternatively say that node gets activated (once it finished computation) and then becomes initiator of the interaction. We start by describing the model in more detail.

The Communication Registers. The communication buffer of each node i consists of two registers: one containing an encoded version of its own (possibly outdated) model, which will only be written to by node i itself, and one for holding an encoded version of its current model, which will only be written to by other nodes. (This second register can be seen as a "communication queue" for the nodes wishing to communicate with i.) Initially all registers contain zero vectors.

Parallel Execution. For the simplicity we will skip the details of quantization, and assume that nodes write and read quantized models directly, without encoding and decoding step. As mentioned above, both current and outdated models are zero vectors initially. Each node i, computes random number of local gradients using outdated mode and other nodes update its current model while i is in compute. Note that even though we call the model other nodes update-current, node i does not have access to it while in compute. Hence, only after node i is done with computing local gradients does it read its current model. Let $\hat{X}_i$ be the value of the outdated model and let X_i be the value of current model. Node i computes average of quantized models $\frac{Q(X_i)+Q(X_j)}{2}$ and writes it in a register which contains current model of node j. Next, it computes $\frac{Q(X_i)+Q(X_j)}{2} - \eta\widetilde{h}(\hat{X}_i)$ (where η is a learning rate and $\widetilde{h}(\hat{X}_i)$ is a sum of local gradients), and writes it in both of its local registers (one containing current model and one containing) outdated model. Once the write is finished, it proceeds by once again computing local gradients, using model $\frac{Q(X_i)+Q(X_j)}{2} - \eta\widetilde{h}(\hat{X}^i_t)$.

Sequential model. We proceed by describing the algorithm sequentially, that is, we map parallel interactions to the sorted sequence of sequential ones. Thus, time tracks the interactions between agents, and each interaction consists of random number of local steps steps which activated node performs, plus one averaging step where activated node (or initiator node) contacts its random neighbour.

The theory assumes that nodes get activated randomly, by independent Poisson clocks, which leads to a uniform global sampling distribution. We approximate this in practice by having the number of local gradient steps executed by each node be a geometric random variable of mean H.[1] Due to the memoryless property, this leads to nodes being sampled uniformly at random in the sequential model.

We start by from the point of view of a single node i which was activated at step $t+1$. The pseudocode can be seen in Algorithm 11.

For $t \geq 0$, let $Enc(\hat{X}_t^i)$ and $Enc(X_t^i)$ be the values written in the registers containing outdated and current model of agent i after t steps. That is, X_t^i is the current model of agent i and $\hat{X}_t^i$ is the outdated model.

The Communication Procedure. Since i was activated at step $t+1$ we will assume that it has already computed H_i local gradients using the outdated model $\hat{X}_t^i$, where H_i is a geometric random variable with mean H, as follows. Let $\widetilde{h}_i^0(\hat{X}_t^i) = 0^d$; for indices $1 \leq q \leq H_i$, let $\widetilde{h}_i^q(\hat{X}_t^i) = \widetilde{g}_i(\hat{X}_t^i - \sum_{s=0}^{q-1} \eta \widetilde{h}_i^s(\hat{X}_t^i))$ be the q-th local gradient. Then, let $\widetilde{h}_i(\hat{X}_t^i) = \sum_{q=1}^{H_i} \widetilde{h}_i^q(\hat{X}_t^i)$ be the sum of all computed local gradients. Or alternatively, since we are in a sequential setting, we can assume that i does computation at step $t+1$.

First, i retrieves $Q(X_t^i)$ (the quantized version of its current model), by decoding $Enc(X_t^i)$ using key $Q(\hat{X}_t^i)$. We would like to note that i can obtain $Q(\hat{X}_t^i)$ simply by decoding $Enc(\hat{X}_t^i)$, using key $\hat{X}_t^i$ (which it knows, to full precision, since it calculated the value itself), and this step does not cost any communication bits since all of the terms involved are local to i's registers.

Then, it contacts its interaction partner j. Node i calculates $Q(\hat{X}_t^j)$ by decoding $Enc(\hat{X}_t^j)$, again using $\hat{X}_t^i$ as a key, and then it retrieves $Q(X_t^j)$ by decoding $Enc(X_t^j)$ with key $Q(\hat{X}_t^j)$.

Then, i calculates

$$X_{t+1}^i = \frac{Q(X_t^i)}{2} + \frac{Q(X_t^j)}{2} - \eta \widetilde{h}_i(\hat{X}_t^i),$$
$$X_{t+1}^j = \frac{Q(X_t^j)}{2} + \frac{Q(X_t^i)}{2}.$$

Next, node i calculates $Enc(X_{t+1}^i)$ and writes to its own register for its outdated models. Here, we use the first case for quantization using Corollary 5.2.1: i is not aware of the key that other nodes will use for decoding, but since it is writing to its own local register, it can afford to use the worst-case $O(d \log T)$ bits. Additionally, it writes $Enc(X_{t+1}^i)$ to its own register containing current model, so that there are enough bits in order for $Q(\hat{X}_{t+1}^i)$ (Note that $\hat{X}_{t+1}^i = X_{t+1}^i$) to be used as decoding key.

Finally, it calculates $Enc(X_{t+1}^j)$ and writes it in the register which contains the current model of j, using enough bits that it can be decoded using $Q(\hat{X}_{t+1}^j)$ (we have that $\hat{X}_{t+1}^j = \hat{X}_t^j$) . Notice that, the way our algorithm is specified, every node which tries to decode $Enc(X_{t+1}^j)$ will use $Q(\hat{X}_{t+1}^j)$ as a key (which i knows), hence Corollary 5.2.1 holds in this case as well.

We emphasize the fact that all this communication is one-way, as it does not require j's intervention.

By Corollary 5.2.1 the total number of bits used is :

[1]Experimentally, results are practically identical if the number of local steps is a small *constant* H.

$$O\Big(d\log(\frac{R}{\epsilon}\|\hat{X}_t^i - \hat{X}_t^j\|)\Big) + O\Big(d\log(\frac{R}{\epsilon}\|Q(\hat{X}_t^j) - X_t^j\|)\Big) + O\Big(d\log(\frac{R}{\epsilon}\|Q(\hat{X}_t^j) - X_{t+1}^j\|)\Big). \quad (5.5)$$

(Recall that we count only reading and writing to *other* registers, and do not count operations i performs on its *own* registers.)

We will show that we can make the probability of any instance of quantization failing less than T^{-c}, for some sufficiently large constant c (by setting the constant factor in number of bits sufficiently high). Then, we can take a union bound over all instances of quantization throughout the algorithm, to show that none fail with high probability in T. Henceforth, we will then be able to prove the convergence of our algorithm conditioned on this event.

Algorithm 11 Sequential SwarmSGD pseudocode for each interaction between nodes i and j.

```
1: % Let $G$ be a communication graph.
2: % Initial models $X_0^1 = X_0^2 = ... = X_0^n$
3: for $t = 0$ to $T - 1$ do
4:     Sample the initiator node $i$ uniformly at random.
5:     Node $i$ samples a node $j$, adjacent to it in $G$, uniformly at random.
6:     Let $t - \tau_t^i$ be the last step at which node $i$ was chosen as initiator.
7:     Let $\hat{X}_t^i = X_{t-\tau_t^i}^i$ be its model from that step.
8:     $Q(X_t^i) \leftarrow Dec(Q(\hat{X}_t^i), Enc(X_t^i))$
9:     $Q(\hat{X}_t^j) \leftarrow Dec(\hat{X}_t^i, Enc(\hat{X}_t^j))$
10:    $Q(X_t^j) \leftarrow Dec(Q(\hat{X}_t^j), Enc(X_t^j))$
11:    $X_{t+1}^i \leftarrow Q(X_t^i)/2 + Q(X_t^j)/2 - \eta\widetilde{h}_i(\hat{X}_{t-1}^i)$
12:    $X_{t+1}^j \leftarrow Q(X_t^i)/2 + Q(X_t^j)/2$
13:    Write $Enc(X_{t+1}^i)$ to the registers containing current and outdated models of node $i$
14:    Write $Enc(X_{t+1}^j)$ to the register containing current model of node $j$
15:    For $k \neq i, j$, $X_{t+1}^k = X_t^k$.
16: end for
```

Avoiding race condition in parallel case. An interesting question is what happens when *multiple nodes* contact j concurrently. For conciseness, our pseudocode assumes that the sequence in lines 8–14 happens atomically, but this sequence can cause a data race. To mitigate this, we can use a bounded non-blocking queue [MS96] at each node instead of a single buffer. Thus, instead of overwriting the buffer value, each node simply appends the corresponding quantized model mean to j's communication queue. In practice, this queue is extremely unlikely to be contended, since communication collisions are rare. Critically, all the above operations are non-blocking, and so the algorithm is deadlock-free.

5.4 The Convergence of SwarmSGD

Let $\mu_t = \sum_{i=1}^n X_t^i/n$ be the mean over node models at time t (after t steps). Our main result is the following:

Theorem 5.4.1 *For the total number of steps $T \geq 10n$, learning rate $\eta = n/\sqrt{T}$, and quantization parameters $R = 2 + T^{\frac{3}{d}}$ and $\epsilon = \frac{\eta H M}{(R^2+7)}$, with probability at least $1 - O(\frac{1}{T})$ we have that Algorithm 11 converges at rate*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\, \|\nabla f(\mu_t)\|^2 \leq \frac{2(f(\mu_0) - f(x^*))}{H\sqrt{T}} + \frac{6(\sigma^2 + 6H\varsigma^2)}{\sqrt{T}} + \frac{12HM^2}{\sqrt{T}} + C\frac{n^2\rho_{max}^3 H^2 L^2 M^2}{T\rho_{min}\lambda_2^2},$$

for constant C, and uses $O\left(d \log\left(\frac{\rho_{max}^2}{\rho_{min}\lambda_2}\right) + \log T\right)$ expected communication bits per step.

Discussion. First, this notion of convergence is standard in the non-convex case [LHLL15, LZZ+17, LZZL18], and each of the upper bound terms has an intuitive interpretation: *the first* represents the reduction in loss relative to the initialization, and gets divided by the number of local steps H, since progress is made in this term in every local step; *the second* represents the noise due to stochasticity, and is naturally linear in H, as H steps are taken in expectation between two interactions. (Recall that in our model T is the number of interactions, and TH is the expected number of gradient steps.) The *fourth* term encodes overheads caused by local steps, quantization, and graph structure; however, it is usually seen as *negligible* [LDS20], due to division by T. We will therefore also ignore this term.

The third term is the critical one, as it implies a dependence on the second-moment bound. Intuitively, this term appears because our algorithm combines both non-blocking communication, *and* quantization: first, unlike prior work, we do not assume an explicit delay upper bound τ on communication; in conjunction with quantization, the unbounded delay this implies that our estimate on the model average μ_t may become dependent on M for large delays, which causes this dependency. While this limitation appears inherent, we are able to remove it if we eliminate quantization: in this case, we get a negligible dependency on M. We formalize this in Corollary 5.4.2.

Second, if we focus on the total number of steps to reach some error bound, we notice an interesting trade-off between the linear reduction in H in the first term, due to local steps, and the linear increase in H in the other terms. Notice that, for dense and low-diameter graphs, such as the regular expanders popular in cluster networks, our convergence bound has no dependence in the graph parameters, and communication is linear in d. However, one limitation is that we could have a $\log n$ dependency in the communication for highly irregular and poorly-connected graphs.

Finally, note that time T here counts *total interactions*. However, $\Theta(n)$ pairwise interactions occur independently in parallel, and so we can replace T by nT in the above formula, to obtain optimal $\Theta(\sqrt{n})$ speedup in terms of wall-clock time. Yet, this speedup is dampened by the variance due to noisy local gradient steps, a fact which we will revisit in the experimental section.

Proof Overview. At a high level, the argument rests on two technical ideas. The first is that, in spite of noise and local steps, the nodes' parameters remain concentrated around the mean μ_t. The second is to leverage this, and bound the impact of stochastic noise and model staleness on convergence. In particular, the main technical difficulty in the proof is to correctly "encode" the fact that parameters are well concentrated around the mean. A natural approach is to bound the model variance Γ_t after t interactions. Formally, we define $\Gamma_t = \sum_{i=1}^{n} \|X_t^i - \mu_t\|^2$, where $\mu_t = \sum_{i=1}^{n} X_t^i/n$, as before.

We bound the expected evolution of Γ_t over time, depending on the learning rate, number of local steps, quantization parameter and the bound provided by the assumption on the stochastic gradients (the bound M^2). The critical point is that the upper bound on the expectation of Γ_t *does not depend* on the number of interactions t. More precisely, if all the above hyper-parameters are constant, we get that $\mathbb{E}[\Gamma(t)] = O(n)$. Our approach brings over tools from classic load-balancing [BFH09], to the multi-dimensional case.

Three key elements of novelty in our case are that (1) for us the load balancing process is *dynamic*, in the sense that new loads, i.e. gradients, get continually added; (2) the load-balancing process we consider is multi-dimensional, whereas usually the literature considers simple scalar weights; (3) the models can be *outdated* and *quantized*, which leads to a complex, noisy load-balancing process. We resolve the this third and most challenging issue by using carefully-defined auxiliary potentials.

Removing the Second-Moment Bound. Upon reflection, we notice that can render the dependency on M^2 negligible if we do not use quantization, but otherwise keep the algorithm the same:

Corollary 5.4.2 *Given the previous assumptions and learning rate $\eta = n/\sqrt{T}$, for some constant C, we have that the Algorithm 11 where quantization is the identity converges at rate*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2 \leq \frac{2(f(\mu_0)-f(x^*))}{H\sqrt{T}} + \frac{6(\sigma^2+6H\varsigma^2)}{\sqrt{T}} + \frac{Cn^2\rho_{max}^3 H^2L^2M^2}{T\rho_{min}\lambda_2^2}.$$

Notice that in this case all the term containing the second moment bound M^2 is dampened by a factor of $\frac{1}{T}$, hence we can assume that Algorithm 11 converges at close-to optimal rate $O\left(\frac{2(f(\mu_0)-f(x^*))}{H\sqrt{T}} + \frac{6H(\sigma^2+6\varsigma^2)}{\sqrt{T}}\right)$. This result still improves upon previous analyses [LZZL18, ALBR18, LDS20] in the sense that communication is completely non-blocking (there is no τ), and we allow for local steps.

5.5 The Complete Analysis

5.5.1 Technical Lemmas on Load Balancing and Graph Properties

In this section provide the useful lemmas which will help as in the later sections.

We are given a simple undirected graph G, with n nodes (for convenience we number them from 1 to n) and edge set $E(G)$. Let ρ_i be a degree of vertex i and let ρ_i be a set of neighbours of i ($|\rho_i| = \rho_i$). Also, we assume that the largest degree among nodes is ρ_{max} and the smallest degree is ρ_{min}.

Each node i of graph G keeps a local vector model $X_t^i \in \mathbb{R}^d$ (t is the number of interactions or steps); let $X_t = (X_t^1, X_t^2, ..., X_t^n)$ be the vector of local models at step t.

Let $\mu_t = \sum_{i=1}^n X_t^i/n$ be the average of models at step t and let $\Gamma_t = \sum_{i=1}^n \|X_t^i - \mu_t\|^2$ be a potential at time step t.

Let $\mathcal{L}$ be the Laplacian matrix of G and let let λ_2 be a second smallest eigenvalue of $\mathcal{L}$. For example, if G is a complete graph $\lambda_2 = n$. In general we have that

$$\lambda_2 \leq 2\rho_{max}. \tag{5.6}$$

First we restate the following lemma from [GM96]:

Lemma 5.5.1

$$\lambda_2 = \min_{v=(v_1,v_2,...,v_n)} \left\{ \frac{v^T \mathcal{L} v}{v^T v} | \sum_{i=1}^{n} v_i = 0 \right\} .$$

Now, we show that Lemma 5.5.1 can be used to lower bound $\sum_{(i,j)\in E(G)} \|X_t^i - X_t^j\|^2$:

Lemma 5.5.2

$$\sum_{(i,j)\in E(G)} \|X_t^i - X_t^j\|^2 \geq \lambda_2 \sum_{i=1}^{n} \|X_t^i - \mu_t\|^2 = \lambda_2 \Gamma_t.$$

Proof. Observe that

$$\sum_{(i,j)\in E(G)} \|X_t^i - X_t^j\|^2 = \sum_{(i,j)\in E(G)} \|(X_t^i - \mu_t) - (X_t^j - \mu_t)\|^2. \tag{5.7}$$

Also, notice that Lemma 5.5.1 means that for every vector $v = (v_1, v_2, ..., v_n)$ such that $\sum_{i=1}^{n} v_i = 0$, we have:

$$\sum_{(i,j)\in E(G)} (v_i - v_j)^2 \geq \lambda_2 \sum_{i=1}^{n} v_i^2.$$

Since $\sum_{i=1}^{n}(X_t^i - \mu_t)$ is a 0 vector, we can apply the above inequality to the each of d components of the vectors $X_t^1 - \mu_t, X_t^2 - \mu_t, ..., X_t^n - \mu_t$ separately, and by elementary properties of 2-norm we prove the lemma.

Let ρ_i be a degree of vertex i; we denote largest degree among nodes by ρ_{max} and the smallest degree by ρ_{min}.

□

5.5.2 Notation and Auxiliary Potential Functions

Recall that $t - \tau_t^i$ is the last time i was chosen as initiator up to and including step t. We would like to emphasize that τ_t^i is a random variable and we do not make any additional assumptions about it. Initially, $\tau_0^i = 0$ for every i. Then, if node i is chosen as initiator at step $t+1$ we have that

$$\tau_{t+1}^i = 0 \tag{5.8}$$

and for each

$$\tau_{t+1}^j = \tau_t^j + 1. \tag{5.9}$$

Next, we provide the formal definition of the local steps performed by our algorithms. Recall that X_t^i is a local model of node i at step t. Let H_t^i be the number of local steps node i performs in the case when it is chosen for interaction at step $t+1$. A natural case is for H_t^i to be fixed throughout the whole algorithm, that is: for each time step t and node i, $H_t^i = H$ (or alternatively we might to try that H_t^i can be a geometric r.v with mean H).

$$\widetilde{h}_i^0(X_t^i) = 0.$$

and for $1 \leq q \leq H_t^i$ let:

$$\widetilde{h}_i^q(X_t^i) = \widetilde{g}_i(X_t^i - \sum_{s=0}^{q-1} \eta \widetilde{h}_i^s(X_t^i)),$$

Note that stochastic gradient is recomputed at each step, but we omit the superscript for local step and global step for simplicity (Whenever we write $\widetilde{g}_i$, we mean that gradient is computed freshly by choosing sample u.a.r from the data available to node i). that is: $\widetilde{h}_i^{q,t}(X_t^i) = \widetilde{g}_i^{q,t}(X_t^i - \sum_{s=0}^{q-1} \eta \widetilde{h}_i^{s,t}(X_t^i))$. Further , for $1 \leq q \leq H_t^i$, let

$$h_i^q(X_t^i) = \mathbb{E}[\widetilde{g}_i(X_t^i - \sum_{s=0}^{q-1} \eta \widetilde{h}_i^s(X_t^i))] = \nabla f(X_t^i - \sum_{s=0}^{q-1} \eta \widetilde{h}_i^s(X_t^i))$$

be the expected value of $\widetilde{h}_i^q(X_t^i)$ taken over the randomness of the stochastic gradient $\widetilde{g}_i$. Let $\widetilde{h}_i(X_t^i)$ be the sum of H_t^i local stochastic gradients we computed:

$$\widetilde{h}_i(X_t^i) = \sum_{q=1}^{H_t^i} \widetilde{h}_i^q(X_t^i).$$

In summary, omitting local step number q means that we compute the sum of all generated gradients (this is the entire update during compute). and omitting tilde sign, means that we compute expectation over the randomness of the samples.

Similarly, for simplicity we avoid using index t in the left side of the above definition, since it is clear that if the local steps are applied to model X_t^i we compute them in the case when node i is chosen as initiator at step $t+1$.

In the case of outdated models this means that

$$\widetilde{h}_i(\hat{X}_t^i) = \widetilde{h}_i^{t-\tau_{ti}}(X_{t-\tau_{ti}})$$

Potential Functions. In order to deal with asynchrony we define the potential function: $\hat{\Gamma}_t = \sum_{i=1}^n \|\hat{X}_t^i - \mu_t\|^2$. This potential helps us to measure how far are the outdated models from the current average. In order to bound $\hat{\Gamma}_t$ in expectation, we will need additional auxiliary potential functions:

$$A_t = \sum_{i=1}^n \|X_{t-\tau_{ti}} - \mu_{t-\tau_{ti}}\|^2$$
$$B_t = \sum_{i=1}^n \|\mu_t - \mu_{t-\tau_{ti}}\|^2$$

Notice that by definition of $\hat{X}_t^i$ and Couchy-Schwarz inequality we get that

$$\hat{\Gamma}_t \leq 2A_t + 2B_t. \tag{5.10}$$

5.5.3 Properties of Local Steps

Lemma 5.5.3 *For any agent i and step t*

$$\mathbb{E}\,\|\widetilde{h}_i(X_t^i)\|^2 \leq 2H^2M^2.$$

Proof.

$$\begin{aligned}\mathbb{E}\,\|\widetilde{h}_i(X_t^i)\|^2 &= \sum_{K=1}^{\infty} Pr[H_t^i = K]\,\mathbb{E}\,\|\sum_{q=1}^{K} \widetilde{h}_i^q(X_t^i)\|^2 \\ &\overset{Cauchy-Schwarz}{\leq} \sum_{K=1}^{\infty} Pr[H_t^i = K]K \sum_{q=1}^{K} \mathbb{E}\,\|\widetilde{h}_i^q(X_t^i)\|^2 \\ &\overset{(5.4)}{\leq} \sum_{K=1}^{\infty} Pr[H_t^i = K]K^2M^2 \leq 2H^2M^2.\end{aligned}$$

Where in the last step we used

$$\sum_{K=1}^{\infty} Pr[H_t^i = K]K^2 = \mathbb{E}[(H_t^i)^2] = 2H^2 - H \leq 2H^2.$$

□

Lemma 5.5.4 *For any agent $1 \leq i \leq n$, number of local steps $1 \leq K$ and step t, we have that*

$$\mathbb{E}\,\|\sum_{q=1}^{K}\widetilde{h}_i^q(\hat{X}_t^i)\|^2 \leq K\sigma^2 + 6L^2K\,\mathbb{E}\,\|\hat{X}_t^i-\mu_t\|^2 + \eta^2L^2K^2(K+1)(2K+1)M^2 + 3K^2\,\mathbb{E}\,\|\nabla f_i(\mu_t)-\nabla f(\mu_t)\|^2 + 3K^2\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2.$$

Proof.

$$\begin{aligned}
\mathbb{E}\,\|\sum_{q=1}^{K}\widetilde{h}_i^q(\hat{X}_t^i)\|^2 &\overset{(5.2)}{\leq} (K\sigma^2+\mathbb{E}\,\|\sum_{q=1}^{K}h_i^q(\hat{X}_t^i)\|^2) = K\sigma^2+\mathbb{E}\left\|\sum_{q=1}^{K}\nabla f_i(\hat{X}_t^i-\sum_{s=0}^{q-1}\eta\widetilde{h}_i^s(\hat{X}_t^i))\right\|^2\\
&\overset{Cauchy-Schwarz}{\leq} K\sigma^2+\sum_{q=1}^{K}K\,\mathbb{E}\left\|\Big(\nabla f_i(\hat{X}_t^i-\sum_{s=0}^{q-1}\eta\widetilde{h}_i^s(\hat{X}_t^i))-\nabla f_i(\mu_t)\Big)\right.\\
&\qquad\left.+\nabla f_i(\mu_t)-\nabla f(\mu_t)+\nabla f(\mu_t)\right\|^2\\
&\overset{Cauchy-Schwarz}{\leq} K\sigma^2+3K\sum_{q=1}^{K}\mathbb{E}\left\|\nabla f_i(\hat{X}_t^i-\sum_{s=0}^{q-1}\eta\widetilde{h}_i^s(\hat{X}_t^i))-\nabla f_i(\mu_t)\right\|^2\\
&\qquad+3K^2\,\mathbb{E}\,\|\nabla f_i(\mu_t)-\nabla f(\mu_t)\|^2+3K^2\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2\\
&\overset{Cauchy-Schwarz,(5.1)}{\leq} K\sigma^2+3L^2K\sum_{q=1}^{K}\mathbb{E}\left\|\hat{X}_t^i-\sum_{s=0}^{q-1}\eta\widetilde{h}_i^s(\hat{X}_t^i)-\mu_t\right\|^2\\
&\qquad+3K^2\,\mathbb{E}\,\|\nabla f_i(\mu_t)-\nabla f(\mu_t)\|^2+3K^2\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2\\
&\overset{Cauchy-Schwarz}{\leq} K\sigma^2+6L^2K\,\mathbb{E}\,\|\hat{X}_t^i-\mu_t\|^2+6\eta^2L^2K\sum_{q=1}^{K}\mathbb{E}\left\|\sum_{s=0}^{q-1}\widetilde{h}_i^s(\hat{X}_t^i))\right\|^2\\
&\qquad+3K^2\,\mathbb{E}\,\|\nabla f_i(\mu_t)-\nabla f(\mu_t)\|^2+3K^2\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2
\end{aligned}$$

To finish the proof, we need to upper bound $\sum_{q=1}^{K}\mathbb{E}\left\|\sum_{s=0}^{q-1}\widetilde{h}_i^s(\hat{X}_t^i))\right\|^2$:

$$\begin{aligned}
\sum_{q=1}^{K}\mathbb{E}\left\|\sum_{s=0}^{q-1}\widetilde{h}_i^s(\hat{X}_t^i))\right\|^2 &\overset{Cauchy-Schwarz}{\leq} \sum_{q=1}^{K}q\left(\sum_{s=0}^{q-1}\mathbb{E}\left\|\widetilde{h}_i^s(\hat{X}_t^i))\right\|^2\right)\\
&\overset{(5.4)}{\leq} \sum_{q=1}^{K}q^2M^2 = K(K+1)(2K+1)M^2/6.
\end{aligned}$$

□

Next, we sum up the upper bound given by the above lemma and take the randomness of the number local steps into the account:

Lemma 5.5.5 *For any step t, we have that*

$$\sum_{i=1}^{n}\mathbb{E}\,\|\widetilde{h}_i(\hat{X}_t^i)\|^2 \leq nH\sigma^2+6L^2H\,\mathbb{E}[\hat{\Gamma}_t]+144n\eta^2L^2H^4M^2+6nH^2\varsigma^2+6nH^2\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2.$$

Proof. Using lemma 5.5.4

$$\begin{aligned}\sum_{i=1}^{n}\mathbb{E}\,\|\widetilde{h}_i(\hat{X}_t^i)\|^2 &= \sum_{i=1}^{n}\sum_{K=1}^{\infty} Pr[H^i_{t-\tau^i_t}=K]\,\mathbb{E}\,\|\sum_{q=1}^{K}\widetilde{h}^q_i(\hat{X}^i_t)\|^2\\ &\le \sum_{i=1}^{n}\sum_{K=1}^{\infty} Pr[H^i_{t-\tau^i_t}=K]\Big(K\sigma^2+6L^2K\,\mathbb{E}\,\|\hat{X}^i_t-\mu_t\|^2\\ &\qquad +\eta^2L^2K^2(K+1)(2K+1)M^2\\ &\qquad +3K^2\,\mathbb{E}\,\|\nabla f_i(\mu_t)-\nabla f(\mu_t)\|^2\\ &\qquad +3K^2\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2\Big)\end{aligned}$$

Notice that $\sum_{K=1}^{\infty} Pr[H^i_{t-\tau^i_t}=K]K=H$, by the definition of expectation. Also,

$$\sum_{K=1}^{\infty} Pr[H^i_{t-\tau^i_t}=K]K^2=\mathbb{E}[(H^i_t)^2]\le 2H^2$$

and

$$\begin{aligned}\sum_{K=1}^{\infty} Pr[H^i_{t-\tau^i_t}=K]K^2(K+1)(2K+1) &\le 6\sum_{K=1}^{\infty} Pr[H^i_{t-\tau^i_t}=K]K^4\\ &=6\,\mathbb{E}[(H^i_t)^4]\le 144H^4.\end{aligned}$$

Thus we get that:

$$\begin{aligned}\sum_{i=1}^{n}\mathbb{E}\,\|\widetilde{h}_i(\hat{X}_t^i)\|^2 &\le \sum_{i=1}^{n}\Big(H\sigma^2+6L^2K\,\mathbb{E}\,\|\hat{X}^i_t-\mu_t\|^2+36\eta^2L^2H^3M^2\\ &\qquad +6H^2\,\mathbb{E}\,\|\nabla f_i(\mu_t)-\nabla f(\mu_t)\|^2\\ &\qquad +6H^2\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2\Big)\\ &\le nH\sigma^2+6L^2H\,\mathbb{E}[\hat{\Gamma}_t]+144n\eta^2L^2H^4M^2+6nH^2\varsigma^2+6nH^2\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2.\end{aligned}$$

Where in the last step we used the definition of $\hat{\Gamma}_t$ and (5.3). □

Lemma 5.5.6 *For any local step* $1\le q$*, and agent* $1\le i\le n$ *and step* t*:*

$$\mathbb{E}\,\|\nabla f_i(\mu_t)-h^q_i(\hat{X}^i_t)\|^2\le 2L^2\,\mathbb{E}\,\|\hat{X}^i_t-\mu_t\|^2+2L^2\eta^2q^2M^2.$$

Proof.

$$\begin{aligned}\mathbb{E}\,\|\nabla f_i(\mu_t)-h^q_i(\hat{X}^i_t)\|^2 &= \mathbb{E}\,\|\nabla f_i(\mu_t)-\nabla f_i(\hat{X}^i_t-\sum_{s=0}^{q-1}\eta\widetilde{h}^s_i(\hat{X}^i_t))\|^2\\ &\overset{(5.1)}{\le} L^2\,\mathbb{E}\,\|\mu_t-X^i_t+\sum_{s=0}^{q-1}\eta\widetilde{h}^s_i(\hat{X}^i_t))\|^2\\ &\overset{Cauchy-Schwarz}{\le} 2L^2\,\mathbb{E}\,\|\hat{X}^i_t-\mu_t\|^2+2L^2\eta^2\,\mathbb{E}\,\|\sum_{s=0}^{q-1}\widetilde{h}^s_i(\hat{X}^i_t)\|^2.\\ &\overset{Cauchy-Schwarz}{\le} 2L^2\,\mathbb{E}\,\|\hat{X}^i_t-\mu_t\|^2+2L^2\eta^2q\sum_{s=0}^{q-1}\mathbb{E}\,\|\widetilde{h}^s_i(\hat{X}^i_t)\|^2\\ &\overset{Cauchy-Schwarz}{\le} 2L^2\,\mathbb{E}\,\|\hat{X}^i_t-\mu_t\|^2+2L^2\eta^2q^2M^2.\end{aligned}$$

□

Lemma 5.5.7 *For any time step t.*

$$\sum_{i=1}^{n} \mathbb{E}\langle \nabla f(\mu_t), -h_i(\hat{X}_t^i)\rangle \le 2HL^2\,\mathbb{E}[\hat{\Gamma}_t] - \frac{3Hn}{4}\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2 + 12H^3nL^2M^2\eta^2.$$

Proof.

$$\sum_{i=1}^{n} \mathbb{E}\langle \nabla f(\mu_t), -h_i(\hat{X}_t^i)\rangle = \sum_{i=1}^{n}\sum_{K=1}^{\infty} Pr[H_{t-\tau_t^i}^i = K]\,\mathbb{E}\langle \nabla f(\mu_t), -\sum_{q=1}^{K} h_i^q(\hat{X}_t^i)\rangle$$
$$= \sum_{i=1}^{n}\sum_{K=1}^{\infty} Pr[H_{t-\tau_t^i}^i = K]\sum_{q=1}^{K}\Big(\mathbb{E}\langle \nabla f(\mu_t), \nabla f_i(\mu_t) - h_i^q(\hat{X}_t^i)\rangle - \mathbb{E}\langle \nabla f(\mu_t), \nabla f_i(\mu_t)\rangle$$

Using Young's inequality we can upper bound $\mathbb{E}\langle \nabla f(\mu_t), \nabla f_i(\mu_t) - h_i^q(\hat{X}_t^i)\rangle$ by

$$\frac{\mathbb{E}\,\|\nabla f(\mu_t)\|^2}{4} + \mathbb{E}\left\|\nabla f_i(\mu_t) - h_i^q(\hat{X}_t^i)\right\|^2.$$

Plugging this in the above inequality we get:

$$\sum_{i=1}^{n} \mathbb{E}\langle \nabla f(\mu_t), -h_i(\hat{X}_t^i)\rangle$$
$$\le \sum_{i=1}^{n}\sum_{K=1}^{\infty} Pr[H_{t-\tau_t^i}^i = K]\sum_{q=1}^{K}\Big(\mathbb{E}\,\|\nabla f(\mu_t) - h_i^q(\hat{X}_t^i)\|^2 + \frac{\mathbb{E}\,\|\nabla f(\mu_t)\|^2}{4} - \mathbb{E}\langle \nabla f(\mu_t), \nabla f_i(\mu_t)\rangle\Big)$$
$$\overset{\text{Lemma 5.5.6}}{\le} \sum_{i=1}^{n}\sum_{K=1}^{\infty} Pr[H_{t-\tau_t^i}^i = K]\sum_{q=1}^{K}\Big(2L^2\,\mathbb{E}\,\|\hat{X}_t^i - \mu_t\|^2 + 2L^2\eta^2q^2M^2 + \frac{\mathbb{E}\,\|\nabla f(\mu_t)\|^2}{4} - \mathbb{E}\langle \nabla f(\mu_t), \nabla f_i(\mu_t)\rangle\Big)$$
$$= \sum_{i=1}^{n}\sum_{K=1}^{\infty} Pr[H_{t-\tau_t^i}^i = K]K\Big(2L^2\,\mathbb{E}\,\|\mu_t - \hat{X}_t^i\|^2 + \frac{\mathbb{E}\,\|\nabla f(\mu_t)\|^2}{4} - \mathbb{E}\langle \nabla f(\mu_t), \nabla f_i(\mu_t)\rangle\Big)$$
$$+ \sum_{i=1}^{n}\sum_{K=1}^{\infty} Pr[H_{t-\tau_t^i}^i = K]K(K+1)(2K+1)L^2M^2\eta^2/3$$

To finish the proof we upper bound the above two terms on the right hand side. Note that:

$$\sum_{i=1}^{n}\sum_{K=1}^{\infty} Pr[H_{t-\tau_t}^i = K]K\Big(2L^2\,\mathbb{E}\,\|\mu_t - \hat{X}_t^i\|^2 + \frac{\mathbb{E}\,\|\nabla f(\mu_t)\|^2}{4} - \mathbb{E}\langle \nabla f(\mu_t), \nabla f_i(\mu_t)\rangle\Big)$$
$$= \sum_{i=1}^{n} H\Big(2L^2\,\mathbb{E}\,\|\mu_t - \hat{X}_t^i\|^2 + \frac{\mathbb{E}\,\|\nabla f(\mu_t)\|^2}{4} - \mathbb{E}\langle \nabla f(\mu_t), \nabla f_i(\mu_t)\rangle\Big)$$
$$= H\Big(2L^2\,\mathbb{E}[\hat{\Gamma}_t] - \frac{3n\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2}{4}\Big)$$

Where in the last step we used that $\sum_{i=1}^{n} \frac{f_i(x)}{n} = f(x)$, for any vector x. Also:

$$\sum_{i=1}^{n}\sum_{K=1}^{\infty} Pr[H_{t-\tau_t^i}^i = K]K(K+1)(2K+1)L^2M^2\eta^2/3$$
$$\le \sum_{i=1}^{n}\sum_{u=1}^{\infty} Pr[H_{t-\tau_t}^i = K]2K^3L^2M^2\eta^2$$
$$\le 12H^3nL^2M^2\eta^2.$$

Where in the last step we used (Recall that $H^i_{t-\tau_i}$ is a geometric random variable with mean H):

$$\sum_{K=1}^{\infty} Pr[H^i_{t-\tau_i} = K]K^3 = \mathbb{E}[(H^i_{t-\tau_i})^3] \le 6H^3.$$

□

5.5.4 Upper Bounding Potential Functions

We proceed by proving the following lemma which upper bounds the expected change in potential:

Lemma 5.5.8 *For any time step t we have:*

$$\mathbb{E}[\Gamma_{t+1}] \le \Big(1 - \frac{\lambda_2}{2n\rho_{max}}\Big)\mathbb{E}[\Gamma_t] + \frac{20\rho^2_{max}}{\rho_{min}\lambda_2}(R^2+7)^2\epsilon^2 + \sum_i \frac{24\rho^2_{max}\eta^2}{\rho_{min}\lambda_2 n}\mathbb{E}\,\|\widetilde{h}_i(\hat{X}^i_t)\|^2.$$

Proof. First we bound change in potential $\Delta_t = \Gamma_{t+1} - \Gamma_t$ for some fixed time step $t > 0$.

For this, let $\Delta^{i,j}_t$ be the change in potential when agent i wakes up (is chosen as initiator) and chooses neighbouring agent j for interaction. Let $S^i_t = -\eta\widetilde{h}_i(\hat{X}^i_t) + \frac{Q(X^i_t)-X^i_t}{2} + \frac{Q(X^j_t)-X^j_t}{2}$ and $S^j_t = \frac{Q(X^i_t)-X^i_t}{2} + \frac{Q(X^j_t)-X^j_t}{2}$. We have that:

$$X^i_{t+1} = \frac{X^i_t + X^j_t}{2} + S^i_t.$$
$$X^j_{t+1} = \frac{X^i_t + X^j_t}{2} + S^j_t.$$
$$\mu_{t+1} = \mu_t + \frac{S^i_t + S^j_t}{n}.$$

This gives us that:

$$X^i_{t+1} - \mu_{t+1} = \frac{X^i_t + X^j_t}{2} + \frac{n-1}{n}S^i_t - \frac{1}{n}S^j_t - \mu_t.$$
$$X^i_{t+1} - \mu_{t+1} = \frac{X^i_t + X^j_t}{2} + \frac{n-1}{n}S^j_t - \frac{1}{n}S^i_t - \mu_t.$$

For $k \ne i, j$ we get that

$$X^k_{t+1} - \mu_{t+1} = X^k_t - \frac{1}{n}(S^i_t + S^j_t) - \mu_t.$$

Hence:

$$\begin{aligned}
\Delta_t^{i,j} &= \Big\|\frac{X_t^i + X_t^j}{2} + \frac{n-1}{n}S_t^i - \frac{1}{n}S_t^j - \mu_t\Big\|^2 - \Big\|X_t^i - \mu_t\Big\|^2 \\
&+ \Big\|\frac{X_t^i + X_t^j}{2} + \frac{n-1}{n}S_t^j - \frac{1}{n}S_t^i - \mu_t\Big\|^2 - \Big\|X_t^j - \mu_t\Big\|^2 \\
&+ \sum_{k\neq i,j}\Big(\Big\|X_t^k - \frac{1}{n}(S_t^i + S_t^j) - \mu_t\|^2 - \Big\|X_t^k - \mu_t\Big\|^2\Big) \\
&= 2\Big\|\frac{X_t^i - \mu_t}{2} + \frac{X_t^j - \mu_t}{2}\Big\|^2 - \Big\|X_t^i - \mu_t\Big\|^2 - \Big\|X_t^j - \mu_t\Big\|^2 \\
&+ \Big\langle X_t^i - \mu_t + X_t^j - \mu_t, \frac{n-2}{n}S_t^i + \frac{n-2}{n}S_t^j\Big\rangle \\
&+ \Big\|\frac{n-1}{n}S_t^i - \frac{1}{n}S_t^j\Big\|^2 + \Big\|\frac{n-1}{n}S_t^j - \frac{1}{n}S_t^i\Big\|^2 \\
&+ \sum_{k\neq i,j} 2\Big\langle X_t^k - \mu_t, -\frac{1}{n}(S_t^i + S_t^j)\Big\rangle \\
&+ \sum_{k\neq i,j}(\frac{1}{n})^2\|S_t^i + S_t^j\|^2.
\end{aligned}$$

Observe that:

$$\sum_{k=1}^{n}\Big\langle X_t^k - \mu_t, -\frac{1}{n}(S_t^i + S_t^j)\Big\rangle = 0.$$

After combining the above two equations, we get that:

$$\begin{aligned}
\Delta_t^{i,j} &= -\frac{\|X_t^i - X_t^j\|^2}{2} + \Big\langle X_t^i - \mu_t + X_t^j - \mu_t, S_t^i + S_t^j\Big\rangle \\
&+ \frac{n-2}{n^2}\Big\|S_t^i + S_t^j\Big\|^2 + \Big\|\frac{n-1}{n}S_t^i - \frac{1}{n}S_t^j\Big\|^2 + \Big\|\frac{n-1}{n}S_t^j - \frac{1}{n}S_t^i\Big\|^2 \\
&\overset{\text{Cauchy-Schwarz}}{\leq} -\frac{\|X_t^i - X_t^j\|^2}{2} + \Big\langle X_t^i - \mu_t + X_t^j - \mu_t, S_t^i + S_t^j\Big\rangle \\
&+ 2\Big(\frac{n-2}{n^2} + \frac{1}{n^2} + \frac{(n-1)^2}{n^2}\Big)\Big(\|S_t^i\|^2 + \|S_t^j\|^2\Big) \\
&\leq -\frac{\|X_t^i - X_t^j\|^2}{2} + \Big\langle X_t^i - \mu_t + X_t^j - \mu_t, S_t^i + S_t^j\Big\rangle \\
&+ 2\Big(\|S_t^i\|^2 + \|S_t^j\|^2\Big).
\end{aligned}$$

Let α be a parameter we will fix later:

$$\begin{aligned}
\Big\langle X_t^i - \mu_t + X_t^j - \mu_t, S_t^i + S_t^j\Big\rangle &\overset{\text{Young}}{\leq} \alpha\Big\|X_t^i - \mu_t + X_t^j - \mu_t\|^2 + \frac{\Big\|S_t^i + S_t^j\Big\|^2}{4\alpha} \\
&\overset{\text{Cauchy-Schwarz}}{\leq} 2\alpha\Big\|X_t^i - \mu_t\Big\|^2 + 2\alpha\Big\|X_t^j - \mu_t\Big\|^2 + \frac{\Big\|S_t^i\Big\|^2 + \Big\|S_t^j\Big\|^2}{2\alpha} \\
&\leq 2\alpha\Big\|X_t^i - \mu_t\Big\|^2 + 2\alpha\Big\|X_t^j - \mu_t\Big\|^2 + \frac{\|S_t^i\|^2 + \|S_t^j\|^2}{2\alpha}.
\end{aligned}$$

Finally, by combining the above two inequalities we get that

$$\begin{aligned}\Delta_t^{i,j} &\le -\frac{\|X_t^i - X_t^j\|^2}{2} + \Big\langle X_t^i - \mu_t + X_t^j - \mu_t, S_t^i + S_t^j \Big\rangle \\ &\quad + 2\Big(\frac{n-2}{n^2} + \frac{1}{n^2} + \frac{(n-1)^2}{n^2}\Big)\Big(\|S_t^i\|^2 + \|S_t^j\|^2\Big) \\ &\le -\frac{\|X_t^i - X_t^j\|^2}{2} + 2\alpha\Big\|X_t^i - \mu_t\Big\|^2 + 2\alpha\Big\|X_t^j - \mu_t\Big\|^2 \\ &\quad + (2 + \frac{1}{2\alpha})\Big(\|S_t^i\|^2 + \|S_t^j\|^2\Big).\end{aligned}$$

Using definitions of S_t^i and S_t^j, Cauchy-Schwarz inequality and properties of quantization we get that

$$\begin{aligned}\|S_t^i\|^2 &\le 3\eta^2\|\widetilde{h}_i(\hat{X}_t^i)\|^2 + \frac{3\|Q(X_t^i) - X_t^i\|^2}{4} + \frac{3\|Q(X_t^j) - X_t^j\|^2}{4} \\ &\le 3\eta^2\|\widetilde{h}_i(\hat{X}_t^i)\|^2 + \frac{3(R^2+7)\epsilon^2}{2}. \\ \|S_t^j\|^2 &\le \frac{\|Q(X_t^i) - X_t^i\|^2}{2} + \frac{\|Q(X_t^j) - X_t^j\|^2}{2} \le (R^2+7)\epsilon^2.\end{aligned}$$

Next, we plug this in the previous inequality:

$$\begin{aligned}\Delta_t^{i,j} &\le -\frac{\|X_t^i - X_t^j\|^2}{2} + 2\alpha\Big\|X_t^i - \mu_t\Big\|^2 + 2\alpha\Big\|X_t^j - \mu_t\Big\|^2 \\ &\quad + (2 + \frac{1}{2\alpha})\Big(\frac{5(R^2+7)\epsilon^2}{2} + 3\eta^2\|\widetilde{h}_i(\hat{X}_t^i)\|^2\Big).\end{aligned}$$

Next, we calculate probability of choosing edges from graph and upper bound Δ_t in expectation, for this we define $\mathbb{E}_t$ as expectation conditioned on the entire history up to and including step t

$$\begin{aligned}\mathbb{E}_t[\Delta_t] &= \sum_i \sum_{j\in\rho_i} \frac{1}{n\rho_i}\,\mathbb{E}_t[\Delta_t^{i,j}] \\ &\le \sum_i \sum_{j\in\rho_i} \frac{1}{n\rho_i}\Bigg(-\frac{\|X_t^i - X_t^j\|^2}{2} + 2\alpha\Big\|X_t^i - \mu_t\Big\|^2 + 2\alpha\Big\|X_t^j - \mu_t\Big\|^2 \\ &\qquad + (2 + \frac{1}{2\alpha})\Big(\frac{5(R^2+7)\epsilon^2}{2} + 3\eta^2\,\mathbb{E}_t\,\|\widetilde{h}_i(\hat{X}_t^i)\|^2\Big)\Bigg) \\ &= -\sum_i \sum_{j\in\rho_i} \frac{\|X_t^i - X_t^j\|^2}{2n\rho_i} \\ &\quad + \sum_i \frac{1}{n}(1 + \sum_{j\in\rho_i}\frac{1}{\rho_j})2\alpha\Big\|X_t^i - \mu_t\Big\|^2 + (5 + \frac{5}{4\alpha})(R^2+7)\epsilon^2 \\ &\quad + \sum_i \sum_{j\in\rho_i} \frac{1}{n\rho_i}(6 + \frac{3}{2\alpha})\eta^2\,\mathbb{E}_t\,\|\widetilde{h}_i(\hat{X}_t^i)\|^2\end{aligned}$$

Now, we use the upper and lower bounds on the degree of vertices

$$\begin{aligned}
\mathbb{E}_t[\Delta_t] &\leq -\sum_i \sum_{j \in \rho_i} \frac{\|X_t^i - X_t^j\|^2}{2n\rho_{max}} \\
&\quad + \sum_i \frac{1}{n}(1 + \sum_{j \in \rho_i} \frac{1}{\rho_{min}}) 2\alpha \Big\| X_t^i - \mu_t \Big\|^2 + (5 + \frac{5}{4\alpha})(R^2+7)\epsilon^2 \\
&\quad + \sum_i \sum_{j \in \rho_i} \frac{1}{n\rho_i}(6 + \frac{3}{2\alpha})\eta^2 \, \mathbb{E}_t \, \|\widetilde{h}_i(\hat{X}_t^i)\|^2 \\
&\leq - \sum_{(i,j) \in E(G)} \frac{\|X_t^i - X_t^j\|^2}{n\rho_{max}} \\
&\quad + \sum_j \frac{\rho_{max}}{\rho_{min} n} 4\alpha \Big\| X_t^j - \mu_t \Big\|^2 + (5 + \frac{5}{4\alpha})(R^2+7)\epsilon^2 \\
&\quad + \sum_i \frac{1}{n}(6 + \frac{3}{2\alpha})\eta^2 \, \mathbb{E}_t \, \|\widetilde{h}_i(\hat{X}_t^i)\|^2
\end{aligned}$$

Now, we use lemma 5.5.2:

$$\begin{aligned}
\mathbb{E}_t[\Delta_t] &\leq - \sum_{(i,j) \in E(G)} \frac{\lambda_2 \Gamma_t}{n\rho_{max}} \\
&\quad + \frac{4\alpha\Gamma_t \rho_{max}}{\rho_{min} n} + (5 + \frac{5}{4\alpha})(R^2+7)\epsilon^2 + \sum_i \frac{1}{n}(6 + \frac{3}{2\alpha})\eta^2 \, \mathbb{E}_t \, \|\widetilde{h}_i(\hat{X}_t^i)\|^2.
\end{aligned}$$

By setting $\alpha = \frac{\lambda_2}{8\rho_{max}^2}$, we get that:

$$\begin{aligned}
\mathbb{E}_t[\Delta_t] &\leq -\frac{\lambda_2 \Gamma_t}{2n\rho_{max}} \\
&\quad + (5 + 10\frac{\rho_{max}^2}{\rho_{min}\lambda_2}) S_t^i + \sum_{i=1}^n (6 + 12\frac{\rho_{max}^2}{\rho_{min}\lambda_2})\eta^2 \, \mathbb{E}_t \, \|\widetilde{h}_i(\hat{X}_t^i)\|^2.
\end{aligned}$$

Next we remove the conditioning , and use the definitions of Δ_i and S_t^i (for S_t^i we also use upper bound which come from the properties of quantization).

$$\begin{aligned}
\mathbb{E}[\mathbb{E}_t[\Gamma_{t+1}]] &= \mathbb{E}[\Delta_t + \Gamma_t] \\
&\leq \Big(1 - \frac{\lambda_2}{2n\rho_{max}}\Big) \mathbb{E}[\Gamma_t] + (5 + 10\frac{\rho_{max}^2}{\rho_{min}\lambda_2})(R^2+7)^2\epsilon^2 \\
&\qquad + \sum_{i=1}^n (6 + 12\frac{\rho_{max}^2}{\rho_{min}\lambda_2})\eta^2 \, \mathbb{E} \, \|\widetilde{h}_i(\hat{X}_t^i)\|^2.
\end{aligned}$$

Finally, we get the proof of the lemma after using $\frac{\rho_{max}^2}{\rho_{min}\lambda_2} \geq \frac{1}{2}$ (See (5.6)) and regrouping terms. □

This allows us to upper bound the potential in expectation for any step t.

Lemma 5.5.9

$$\mathbb{E}[\Gamma_t] \leq \frac{40n\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2 + \frac{96n\rho_{max}^3}{\rho_{min}\lambda_2^2} H^2 M^2 \eta^2.$$

Proof. We prove by using induction. Base case $t = 0$ trivially holds. For an induction step step we assume that $\mathbb{E}[\Gamma_t] \leq \frac{40n\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2 + \frac{96n\rho_{max}^3}{\rho_{min}\lambda_2^2} H^2 M^2 \eta^2$. We get that :

$$
\begin{aligned}
\mathbb{E}[\Gamma_{t+1}] &\leq \Big(1 - \frac{\lambda_2}{2n\rho_{max}}\Big)\mathbb{E}[\Gamma_t] + \frac{20\rho_{max}^2}{\rho_{min}\lambda_2}(R^2+7)^2\epsilon^2 + \sum_i \frac{24\rho_{max}^2}{\rho_{min}\lambda_2 n}\mathbb{E}\,\|\widetilde{h}_i(\hat{X}_t^i)\|^2 \\
&\overset{\text{Lemma 5.5.3}}{\leq} \Big(1 - \frac{\lambda_2}{2n\rho_{max}}\Big)\mathbb{E}[\Gamma_t] + \frac{20\rho_{max}^2}{\rho_{min}\lambda_2}(R^2+7)^2\epsilon^2 + \frac{48\rho_{max}^2\eta^2}{\rho_{min}\lambda_2}H^2M^2 \\
&\leq \Big(1 - \frac{\lambda_2}{2n\rho_{max}}\Big)\Big(\frac{40\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2 + \frac{96\rho_{max}^3}{\rho_{min}\lambda_2^2}H^2M^2\eta^2\Big) \\
&\qquad + \frac{20\rho_{max}^2}{\rho_{min}\lambda_2}(R^2+7)^2\epsilon^2 + \frac{48\rho_{max}^2\eta^2}{\rho_{min}\lambda_2}H^2M^2 \\
&= \frac{40n\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2 + \frac{96n\rho_{max}^3}{\rho_{min}\lambda_2^2}H^2M^2\eta^2.
\end{aligned}
$$

□

Lemma 5.5.10 *For any time step t:*

$$\mathbb{E}[A_{t+1}] \leq (1 - \frac{1}{n})\,\mathbb{E}[A_t] + \frac{1}{n}\,\mathbb{E}[\Gamma_{t+1}].$$

Proof. Recall that if node i is chosen as initiator at step t, then for each $j \neq i$,

$$\hat{X}^j_{t+1-\tau^j_{t+1}} - \mu_{t+1-\tau^j_{t+1}} = \hat{X}^j_{t-\tau^j_t} - \mu_{j-\tau^j_t},$$

since $\tau^j_{t+1} = \tau^j_t + 1$ and

$$\hat{X}^i_{t+1-\tau^i_{t+1}} - \mu_{t+1-\tau^i_{t+1}} = X^i_{t+1} - \mu_{t+1},$$

since $\tau^i_{t+1} = 0$ (see equations (5.8) and (5.9)). Thus, if $\mathbb{E}_t$ is expectation conditioned on the entire history up to and including step t then

$$
\begin{aligned}
\mathbb{E}_t[A_{t+1} - A_t] &= \sum_{i=1}^n \frac{1}{n}\Bigg(\mathbb{E}_t\,\|X^i_{t+1} - \mu_{t+1}\|^2 - \|\hat{X}^i_{i-\tau^i_t} - \mu_{i-\tau^i_t}\|\Bigg) \\
&= \frac{1}{n}\,\mathbb{E}_t[\Gamma_{t+1}] - \frac{1}{n}A_t.
\end{aligned}
$$

After removing conditioning and regrouping terms we get the proof of the lemma. □

Next, we upper bound A_t in expectation

Lemma 5.5.11 *For any time step t:*

$$\mathbb{E}[A_t] \leq \frac{40n\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2 + \frac{96n\rho_{max}^3}{\rho_{min}\lambda_2^2}H^2M^2\eta^2.$$

Proof. By combining Lemmas 5.5.9 and 5.5.10 we get that:

$$\mathbb{E}[A_{t+1}] \leq (1 - \frac{1}{n})A_t + \frac{40\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2 + \frac{96\rho_{max}^3}{\rho_{min}\lambda_2^2}H^2M^2\eta^2$$

and the proof follows by using the same type of induction as in the proof of Lemma $5.5.9$

□

Next we provide two different versions of upper bounding $\mathbb{E}\,\|\mu_{t+1} - \mu_t\|^2$, the first one will be useful for upper bounding $\mathbb{E}[B_t]$ and the second one will be used in the proof of convergence for SwarmSGD.

Lemma 5.5.12 *For any time step t:*

$$\mathbb{E}\left\|\mu_{t+1}-\mu_t\right\|^2 \leq \frac{6(R^2+7)^2\epsilon^2+6\eta^2H^2M^2}{n^2}.$$

Proof. Let i be the agent which is chosen as initiator at step $t+1$ and let j be the neighbour it selected for interaction, also let E_t be expectation which is condition on the entire history up to and including step t We have that

$$\begin{aligned}
\mathbb{E}_t\|\mu_{t+1}-\mu_t\|^2 &= \frac{1}{n^2}\mathbb{E}_t\left\|Q(X_t^i)-X_t^i+Q(X_t^j-X_t^j-\eta\widetilde{h}_i(\hat{X}_t^i)\right\|^2 \\
&\overset{Cauchy-Schwarz}{\leq} \frac{3}{n^2}\mathbb{E}_t\left\|Q(X_t^i)-X_t^i\right\|+\frac{3}{n^2}E_t\left\|Q(X_t^j)-X_t^j\right\|^2+\frac{3\eta^2}{n^2}\mathbb{E}_t\left\|\widetilde{h}_i(\hat{X}_t^i)\right\|^2 \\
&\leq \frac{6(R^2+7)^2\epsilon^2+6\eta^2H^2M^2}{n^2}.
\end{aligned}$$

Where in the last step we used property of quantization and Lemma 5.5.3. Since this upper bound holds for any agents i and j, after removing the conditioning, we get the proof of the lemma. □

Lemma 5.5.13 *For any step t*

$$\begin{aligned}
\mathbb{E}\left\|\mu_{t+1}-\mu_t\right\|^2 \leq \frac{6(R^2+7)^2\epsilon^2}{n^2}+\frac{3\eta^2H\sigma^2}{n^2}+\frac{18\eta^2L^2H\,\mathbb{E}[\hat{\Gamma}_t]}{n^3}+\frac{432\eta^4L^2H^4M^2}{n^2} \\
+\frac{18\eta^2H^2\varsigma^2}{n^2}+\frac{18\eta^2H^2\,\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2}{n^2}.
\end{aligned}$$

Proof. Following the same steps as the proof of Lemma 5.5.12 and taking the randomness of agents i (the initiator) and j interacting at step $t+1$ in to the account we get that

$$\begin{aligned}
\mathbb{E}\left\|\mu_{t+1}-\mu_t\right\|^2 &\leq \sum_{i=1}^{n}\sum_{j\in\rho_i}\frac{1}{n^3\rho_i}\Bigg(3\,\mathbb{E}\left\|Q(X_t^i)-X_t^i\right\|+3\,\mathbb{E}\left\|Q(X_t^j)-X_t^j\right\|^2 \\
&\qquad +3\eta^2\,\mathbb{E}\left\|\widetilde{h}_i(\hat{X}_t^i)\right\|^2\Bigg) \\
&\leq \frac{6(R^2+7)^2\epsilon^2}{n^2}+\frac{3\eta^2}{n^3}\sum_{i=1}^{n}\mathbb{E}\,\|\widetilde{h}_i(\hat{X}_t^i)\|^2 \\
&\overset{\text{Lemma 5.5.5}}{\leq} \frac{6(R^2+7)^2\epsilon^2}{n^2}+\frac{3\eta^2}{n^3}\Big(nH\sigma^2+6L^2H\,\mathbb{E}[\hat{\Gamma}_t]+144n\eta^2L^2H^4M^2 \\
&\qquad +6nH^2\varsigma^2+6nH^2\,\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2\Big) \\
&= \frac{6(R^2+7)^2\epsilon^2}{n^2}+\frac{3\eta^2H\sigma^2}{n^2}+\frac{18\eta^2L^2H\,\mathbb{E}[\hat{\Gamma}_t]}{n^3}+\frac{432\eta^4L^2H^4M^2}{n^2} \\
&\qquad +\frac{18\eta^2H^2\varsigma^2}{n^2}+\frac{18\eta^2H^2\,\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2}{n^2}.
\end{aligned}$$

□

Our next goal is to upper bound $\mathbb{E}[B_t]$, for which we will need the following lemma:

Lemma 5.5.14 *For any time step t and agent i:*

$$\mathbb{E}\left\|\mu_t-\mu_{t-\tau_t^i}\right\|^2 \leq \frac{\mathbb{E}[(\tau_t^i)^2]\Big(6(R^2+7)^2\epsilon^2+6\eta^2H^2M^2\Big)}{n^2}.$$

Proof. Let $\mathbb{E}_{\tau_t^i}$ be an expectation which is conditioned on τ_t^i

$$\mathbb{E}_{\tau_i} \|\mu_t - \mu_{t-\tau_t^i}\|^2 = \mathbb{E}_{\tau_t^i} \Big\| \sum_{s=t-\tau_t^i}^{t-1} \Big(\mu_{s+1} - \mu_s\Big)\Big\|^2 \overset{Cauchy-Schwarz}{\leq} \tau_t^i \sum_{s=t-\tau_t^i}^{t-1} \mathbb{E}_{\tau_t^i} \|\mu_{s+1} - \mu_s\|^2.$$

Note that for any $t-\tau_t^i \leq s \leq t-1$ we can use Lemma 5.5.12 to upper bound $\mathbb{E}_{\tau_t^i} \|\mu_{s+1}-\mu_s\|^2$, since it uses quantization property and Lemma 5.5.3 (which in turn uses (5.4) and the randomness of the number of local steps) which do not depend on τ_t^i. In fact, as proof of Lemma 5.5.12 suggests, for any $t - \tau_t^i \leq s \leq t-1$, we could condition $\mathbb{E}\|\mu_{s+1} - \mu_s\|^2$ on the entire history up to and including step s (this history includes τ_t^i as well) and the upper bound would still hold. Thus, we get that

$$\mathbb{E}_{\tau_t^i} \|\mu_t - \mu_{t-\tau_t^i}\|^2 \leq \frac{(\tau_t^i)^2\Big(6(R^2+7)^2\epsilon^2 + 6\eta^2H^2M^2\Big)}{n^2}.$$

Finally we remove the conditioning :

$$\mathbb{E}\|\mu_t - \mu_{t-\tau_t^i}\|^2 = \mathbb{E}[\mathbb{E}_{\tau_t^i} \|\mu_t - \mu_{t-\tau_t^i}\|^2] \leq \frac{\mathbb{E}[(\tau_t^i)^2]\Big(6(R^2+7)^2\epsilon^2 + 6\eta^2H^2M^2\Big)}{n^2}.$$

Next, we proceed to prove the following lemma: □

Lemma 5.5.15 *For any step t*

$$\sum_{i=1}^n \mathbb{E}[(\tau_t^i)^2] \leq 5n^3. \tag{5.11}$$

Proof. For a fixed step s, let $\mathbb{E}_s$ be an expectation conditioned on the entire history up to and including step s. If agent i is chosen as initiator at step $s+1$ then $\tau_{s+1}^i = 0$ and otherwise $\tau_{s+1}^i = \tau_s^i + 1$. Since i is chosen with probability $\frac{1}{n}$, we have that

$$\sum_{i=1}^n \mathbb{E}_s[(\tau_{s+1}^i)^2] = \sum_{i=1}^n (1-\frac{1}{n})\Big((\tau_s^i)^2 + 2\tau_s^i + 1\Big) \leq (1-\frac{1}{n})\sum_{i=1}^n (\tau_s^i)^2 + 2\sum_{i=1}^n \tau_s^i + \frac{n^2}{2}.$$

Where in the last step we used that $n \geq 2$.

Also, by using Cauchy-Schwarz inequality we get that

$$\Big(\sum_{i=1}^n \tau_s^i\Big)^2 \leq n\Big(\sum_{i=1}^n (\tau_s^i)^2\Big) \leftrightarrow \sum_{i=1}^n \tau_s^i \leq \sqrt{n\sum_{i=1}^n (\tau_s^i)^2}.$$

Thus,

$$\sum_{i=1}^n \mathbb{E}_s[(\tau_{s+1}^i)^2] \leq (1-\frac{1}{n})\sum_{i=1}^n (\tau_s^i)^2 + 2\sqrt{n\sum_{i=1}^n (\tau_s^i)^2} + \frac{n^2}{2}.$$

Next, we remove the conditioning:

$$\sum_{i=1}^n \mathbb{E}[(\tau_{s+1}^i)^2] = \sum_{i=1}^n \mathbb{E}[\mathbb{E}_s[(\tau_{s+1}^i)^2]] \leq (1-\frac{1}{n})\sum_{i=1}^n \mathbb{E}[(\tau_s^i)^2] + 2\,\mathbb{E}\sqrt{n\sum_{i=1}^n (\tau_s^i)^2} + \frac{n^2}{2}$$

$$\leq (1-\frac{1}{n})\sum_{i=1}^n \mathbb{E}[(\tau_s^i)^2] + 2\sqrt{n\sum_{i=1}^n \mathbb{E}[(\tau_s^i)^2]} + \frac{n^2}{2}.$$

Where in the last step with use Jensen's inequality and concavity of square root function. Finally, we finish the proof of the lemma using induction. Base case holds trivially, for induction step we assume that $\sum_{i=1}^n \mathbb{E}[(\tau_{s+1}^i)^2] \le 5n^3$. We have that

$$\sum_{i=1}^n \mathbb{E}[(\tau_{s+1}^i)^2] \le (1-\frac{1}{n})\sum_{i=1}^n \mathbb{E}[(\tau_s^i)^2] + 2\sqrt{n\sum_{i=1}^n \mathbb{E}[(\tau_s^i)^2]} + \frac{n^2}{2}$$
$$\le (1-\frac{1}{n})(5n^3) + 2\sqrt{5n^4} + \frac{n^2}{2} = 5n^3 + n^2(-5+2\sqrt{5}+\frac{1}{2}) \le 5n^3.$$

This finishes the proof of the Lemma. □

Finally, we are ready to upper bound B_t

Lemma 5.5.16 *For any step t:*

$$\mathbb{E}[B_t] \le 5n\Big(6(R^2+7)^2\epsilon^2 + 6\eta^2H^2M^2\Big).$$

Proof. Lemma 5.5.14 gives us that

$$\mathbb{E}[B_t] = \sum_{i=1}^n \mathbb{E}\,\|\mu_t - \mu_{t-\tau_t^i}\|^2 \le \sum_{i=1}^n \frac{\mathbb{E}[(\tau_t^i)^2]\Big(6(R^2+7)^2\epsilon^2 + 6\eta^2H^2M^2\Big)}{n^2}$$

After applying Lemma 5.5.15 we get the proof of the Lemma. □

The last lemma in this section upper bounds $\mathbb{E}[\hat{\Gamma}_t]$:

Lemma 5.5.17 *For any step t, we have that*

$$\mathbb{E}[\hat{\Gamma}_t] \le \frac{200n\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2 + \frac{312n\rho_{max}^3}{\rho_{min}\lambda_2^2}H^2M^2\eta^2$$

Proof. From (5.10), and Lemmas 5.5.11 and 5.5.16 we get that

$$\mathbb{E}[\hat{\Gamma}_t] \le 2\,\mathbb{E}[A_t] + 2\,\mathbb{E}[B_t] \le \frac{80n\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2 + \frac{192n\rho_{max}^3}{\rho_{min}\lambda_2^2}H^2M^2\eta^2$$
$$+ 5n\Big(6(R^2+7)^2\epsilon^2 + 6\eta^2H^2M^2\Big)$$
$$\le \frac{200n\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2 + \frac{312n\rho_{max}^3}{\rho_{min}\lambda_2^2}H^2M^2\eta^2$$

Where in the last step we used $\frac{\rho_{max}^2}{\rho_{min}\lambda_2} \ge \frac{1}{2}$ (See (5.6)). □

5.5.5 The Convergence of SwarmSGD

Theorem 5.5.18 *For learning rate $\eta = n/\sqrt{T}$, Algorithm 11 converges at rate:*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\,\|\nabla f(\mu_t)\|^2 \le \frac{2(f(\mu_0)-f(x^*))}{H\sqrt{T}} + \frac{6(\sigma^2+6H\varsigma^2)}{\sqrt{T}}$$
$$+ \frac{1600\rho_{max}^3(R^2+7)^2\epsilon^2L^2}{\rho_{min}\lambda_2^2} + \frac{2496n^2\rho_{max}^3H^2L^2M^2}{T\rho_{min}\lambda_2^2}$$
$$+ \frac{78H^2L^2M^2n^2}{T} + \frac{12(R^2+7)^2\epsilon^2\sqrt{T}}{Hn^2}.$$

Proof. Let $\mathbb{E}_t$ denote expectation conditioned on the entire history up to and including step t. By L-smoothness we have that

$$\mathbb{E}_t[f(\mu_{t+1})] \le f(\mu_t) + \mathbb{E}_t\langle\nabla f(\mu_t), \mu_{t+1}-\mu_t\rangle + \frac{L}{2}\mathbb{E}_t\,\|\mu_{t+1}-\mu_t\|^2. \tag{5.12}$$

First we look at $\mathbb{E}_t\langle \nabla f(\mu_t), \mu_{t+1} - \mu_t\rangle = \langle \nabla f(\mu_t), \mathbb{E}_t[\mu_{t+1} - \mu_t]\rangle$. If agent i is chosen as initiator at step $t+1$ and it picks its neighbour j to interact, We have that

$$\mu_{t+1} - \mu_t = -\frac{\eta}{n}\widetilde{h}_i(\hat{X}_t^i) - (X_t^i - Q(X_t^i)) - (X_t^j - Q(X_t^j)).$$

Thus, in this case:

$$\mathbb{E}_t[\mu_{t+1} - \mu_t] = -\frac{\eta}{n}h_i(\hat{X}_t^i).$$

Where we used unbiasedness of quantization and stochastic gradients. We would like to note that even though we do condition on the entire history up to and including step t and this includes conditioning on $\hat{X}_t^i$, the algorithm has not yet used $\widetilde{h}_i(\hat{X}_t^i)$ (it does not count towards computation of μ_t), thus we can safely use all properties of stochastic gradients. Hence, we can proceed by taking into the account that each agent i is chosen as initiator with probability $\frac{1}{n}$:

$$\mathbb{E}_t[\mu_{t+1} - \mu_t] = -\sum_{i=1}^{n}\frac{\eta}{n^2}h_i(\hat{X}_t^i).$$

and subsequently

$$\mathbb{E}_t\langle \nabla f(\mu_t), \mu_{t+1} - \mu_t\rangle = \sum_{i=1}^{n}\frac{\eta}{n^2}\,\mathbb{E}_t\langle \nabla f(\mu_t), -h_i(\hat{X}_t^i)\rangle.$$

Hence, we can rewrite (5.12) as:

$$\mathbb{E}_t[f(\mu_{t+1})] \le f(\mu_t) + \sum_{i=1}^{n}\frac{\eta}{n^2}\,\mathbb{E}_t\langle \nabla f(\mu_t), -h_i(\hat{X}_t^i)\rangle + \frac{L}{2}\,\mathbb{E}_t\,\|\mu_{t+1} - \mu_t\|^2.$$

Next, we remove the conditioning

$$\mathbb{E}[(\mu_{t+1})] = \mathbb{E}[\mathbb{E}_t[f(\mu_{t+1})]] \le \mathbb{E}[f(\mu_t)] + \sum_{i=1}^{n}\frac{\eta}{n^2}\,\mathbb{E}\langle \nabla f(\mu_t), -h_i(\hat{X}_t^i)\rangle + \frac{L}{2}\,\mathbb{E}\,\|\mu_{t+1} - \mu_t\|^2.$$

This allows us to use Lemmas 5.5.13 and 5.5.7:

$$\begin{aligned}\mathbb{E}[f(\mu_{t+1})] - \mathbb{E}[f(\mu_t)] \le{}& \frac{2\eta HL^2\,\mathbb{E}[\hat{\Gamma}_t]}{n^2} - \frac{3H\eta}{4n}\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2 + \frac{12H^3L^2M^2\eta^3}{n}\\ &+ \frac{6(R^2+7)^2\epsilon^2}{n^2} + \frac{3\eta^2H\sigma^2}{n^2}\\ &+ \frac{18\eta^2L^2H\,\mathbb{E}[\hat{\Gamma}_t]}{n^3} + \frac{432\eta^4L^2H^4M^2}{n^2}\\ &+ \frac{18\eta^2H^2\varsigma^2}{n^2} + \frac{18\eta^2H^2\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2}{n^2}.\end{aligned}$$

To simplify the above inequality we assume that $\eta \le \frac{1}{8H}$ and also use the fact that $n \ge 2$. We get:

$$\begin{aligned}\mathbb{E}[f(\mu_{t+1})] - \mathbb{E}[f(\mu_t)] \le{}& \frac{4\eta HL^2\,\mathbb{E}[\hat{\Gamma}_t]}{n^2} - \frac{H\eta}{2n}\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2 + \frac{39H^3L^2M^2\eta^3}{n}\\ &+ \frac{6(R^2+7)^2\epsilon^2}{n^2} + \frac{3\eta^2H(\sigma^2+6H\varsigma^2)}{n^2}.\end{aligned}$$

Here, important thing is that we used $\frac{18\eta^2H^2\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2}{n^2} - \frac{H\eta\,\mathbb{E}\,\|\nabla f(\mu_t)\|^2}{4n} \le 0$.

Further, we use Lemma 5.5.17:

$$
\begin{aligned}
\mathbb{E}[f(\mu_{t+1})] - \mathbb{E}[f(\mu_t)] \leq{} & \frac{4\eta H L^2 \left(\frac{200 n \rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2 + \frac{312 n\rho_{max}^3}{\rho_{min}\lambda_2^2} H^2 M^2 \eta^2 \right)}{n^2} \\
& - \frac{H\eta}{2n}\mathbb{E}\,\|\nabla f(\mu_t)\|^2 + \frac{39 H^3 L^2 M^2 \eta^3}{n} \\
& + \frac{6(R^2+7)^2\epsilon^2 H L^2}{n^2} + \frac{3\eta^2 H(\sigma^2 + 6H\varsigma^2)}{n^2} \\
={} & \frac{800\eta\rho_{max}^3(R^2+7)^2\epsilon^2 H L^2}{n\rho_{min}\lambda_2^2} + \frac{1248\eta^3\rho_{max}^3 H^3 L^2 M^2}{n\rho_{min}\lambda_2^2} \\
& - \frac{H\eta}{2n}\mathbb{E}\,\|\nabla f(\mu_t)\|^2 + \frac{39 H^3 L^2 M^2 \eta^3}{n} \\
& + \frac{6(R^2+7)^2\epsilon^2}{n^2} + \frac{3\eta^2 H(\sigma^2 + 6H\varsigma^2)}{n^2}.
\end{aligned}
$$

by summing the above inequality for $t = 0$ to $t = T-1$, we get that

$$
\begin{aligned}
\mathbb{E}[f(\mu_T)] - f(\mu_0) \leq{} & -\sum_{t=0}^{T-1}\frac{\eta H}{2n}\mathbb{E}\,\|\nabla f(\mu_t)\|^2 + \frac{3\eta^2 H(\sigma^2+6H\varsigma^2)T}{n^2} \\
& + \frac{800\eta\rho_{max}^3(R^2+7)^2\epsilon^2 H L^2 T}{n\rho_{min}\lambda_2^2} + \frac{1248\eta^3\rho_{max}^3 H^3 L^2 M^2 T}{n\rho_{min}\lambda_2^2} \\
& + \frac{39 H^3 L^2 M^2 \eta^3 T}{n} + \frac{6(R^2+7)^2\epsilon^2 T}{n^2}
\end{aligned}
$$

Next, we regroup terms, multiply both sides by $\frac{2n}{\eta H T}$ and use the fact that $f(\mu_T) \geq f(x^*)$:

$$
\begin{aligned}
\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\,\|\nabla f(\mu_t)\|^2 \leq{} & \frac{2n(f(\mu_0)-f(x^*))}{H\eta T} + \frac{6\eta(\sigma^2+6H\varsigma^2)}{n} \\
& + \frac{1600\rho_{max}^3(R^2+7)^2\epsilon^2 L^2}{\rho_{min}\lambda_2^2} + \frac{2496\eta^2\rho_{max}^3 H^2 L^2 M^2}{\rho_{min}\lambda_2^2} \\
& + 78H^2L^2M^2\eta^2 + \frac{12(R^2+7)^2\epsilon^2}{n\eta H}
\end{aligned}
$$

Finally, we set $\eta = \frac{n}{\sqrt{T}}$:

$$
\begin{aligned}
\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\,\|\nabla f(\mu_t)\|^2 \leq{} & \frac{2(f(\mu_0)-f(x^*))}{H\sqrt{T}} + \frac{6(\sigma^2+6H\varsigma^2)}{\sqrt{T}} \\
& + \frac{1600\rho_{max}^3(R^2+7)^2\epsilon^2 L^2}{\rho_{min}\lambda_2^2} + \frac{2496 n^2\rho_{max}^3 H^2 L^2 M^2}{T\rho_{min}\lambda_2^2} \\
& + \frac{78H^2L^2M^2n^2}{T} + \frac{12(R^2+7)^2\epsilon^2\sqrt{T}}{Hn^2}.
\end{aligned} \tag{5.13}
$$

□

Proof of Corollary 5.4.2. We get the proof by simply omitting quantization parameters R and ϵ from the convergence bound given by the above theorem.

Our next goal is to show **how quantization affects the convergence**.

First we prove that the probability of quantization failing during the entire run of the algorithm is negligible.

Lemma 5.5.19 *Let $T \geq 10n$, then for quantization parameters $R = 2 + T^{\frac{3}{d}}$ and $\epsilon = \frac{\eta HM}{(R^2+7)}$ we have that the probability of quantization never failing during the entire run of the Algorithm 11 is at least $1 - O\left(\frac{1}{T}\right)$.*

Proof. Let $\mathcal{L}_t$ be the event that quantization does not fail during step t. Our goal is to show that $Pr[\cup_{t=1}^{T}\mathcal{L}_t] \geq 1 - O\left(\frac{1}{T}\right)$. In order to do this, we first prove that $Pr[\neg\mathcal{L}_{t+1}|\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_t] \leq O\left(\frac{1}{T^2}\right)$ (O is with respect to T here).

Recall that up to this point we always assumed that quantization never fails, and we omitted conditioning on this event. Next, we rewrite our potential bounds but with the conditioning: Lemma 5.5.9 gives us that for any step t

$$\mathbb{E}[\Gamma_t|\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_t] \leq \frac{136n\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2. \tag{5.14}$$

and Lemma 5.5.17 gives us that

$$\mathbb{E}[\hat{\Gamma}_t|\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_t] \leq \frac{512n\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2 \tag{5.15}$$

Where we also used that $(R^2+7)\epsilon^2 = H^2\eta^2M^2$. Recall that quantization can fail because the distance between the vectors is larger than R^{R^d} (In this case we can not even use quantization algorithm), if the distance is larger than $\frac{\epsilon T^{17}}{R}$ (In this case a node will not be able to write enough bits in its own communication buffer), or if quantization algorithm fails itself. First, we will concentrate on lower bounding probability that the distance between the models is large. That is, we need need to lower bound probability that :

$$\|Q(\hat{X}_t^i) - X_t^i\|^2 \leq (R^{R^d}\epsilon)^2 \tag{5.16}$$

$$\|\hat{X}_t^i - \hat{X}_t^j\|^2 \leq (R^{R^d}\epsilon)^2 \tag{5.17}$$

$$\|\hat{X}_t^i - \hat{X}_t^j\|^2 \leq \frac{\epsilon^2T^{34}}{R^2} \tag{5.18}$$

$$\|Q(\hat{X}_t^j) - X_t^j\|^2 \leq (R^{R^d}\epsilon)^2. \tag{5.19}$$

We would like to point out that these conditions are necessary for decoding to succeed, we ignore encoding since it will be counted when someone will try to decode it.

Notice that by using Cauchy-Schwarz we get that

$$\begin{aligned}
&\|Q(\hat{X}_t^i) - X_t^i\|^2 + \|\hat{X}_t^i - \hat{X}_t^j\|^2 + \|Q(\hat{X}_t^j) - \hat{X}_t^j\|^2 \\
&\leq 3\|Q(\hat{X}_t^i) - \hat{X}_t^i\|^2 + 3\|\hat{X}_t^i - \mu_t\|^2 + 3\|\mu_t - X_t^i\|^2 \\
&+ 2\|\hat{X}_t^i - \mu_t\|^2 + 2\|\mu_t - \hat{X}_t^j\|^2 \\
&+ 3\|Q(\hat{X}_t^j) - \hat{X}_t^j\|^2 + 3\|\hat{X}_t^j - \mu_t\|^2 + 3\|\mu_t - X_t^j\|^2 \\
&\leq 10\hat{\Gamma}_t + 6\Gamma_t + 6(R^2+7)^2\epsilon^2.
\end{aligned}$$

Since, $R = 2 + T^{\frac{3}{d}}$ this means that $(R^{R^d})^2 \geq 2^{2T^3} \geq T^{34}$. Hence, to lower bound probability that (5.16), (5.17), (5.18), (5.19) are be satisfied it is suffices to upper bound the probability that $10\hat{\Gamma}_t + 6\Gamma_t + 6(R^2+7)^2\epsilon^2 \geq \frac{T^{34}\epsilon^2}{R^2}$:

For this, we use Markov's inequality:

$$\begin{aligned}
Pr\Big[(10\hat{\Gamma}_t + 6\Gamma_t + 6(R^2+7)^2\epsilon^2) \geq T^{34}\epsilon^2 | \mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_t\Big] \\
&\leq \frac{\mathbb{E}[10\hat{\Gamma}_t + 6\Gamma_t + 6(R^2+7)^2\epsilon^2 | \mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_t] R^2}{T^{34}\epsilon^2} \\
&\overset{(5.14),(5.15)}{\leq} \frac{\frac{5936 n \rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2R^2\epsilon^2 + 6(R^2+7)^2R^2\epsilon^2}{T^{34}\epsilon^2} \\
&\leq \frac{1}{T^2}.
\end{aligned}$$

In the last step we used that $T \geq 10n \geq 10$, $R^2+7 \leq 3R^2 \leq 6T^6$ and $\lambda_2 \geq \frac{1}{n^2}$ for a connected graph. Thus, the failure probability due to the models not being close enough for quantization to be applied is at most $O\left(\frac{1}{T^2}\right)$. Conditioned on the event that $\|Q(\hat{X}_t^i) - X_t^i\|$, $\|\hat{X}_t^i - \hat{X}_t^j\|$ and $\|Q(\hat{X}_t^j) - X_t^j\|$ are upper bounded by $T^{17}\epsilon$ (This is what we actually lower bounded the probability for using Markov), we get that the probability of quantization algorithm failing is at most

$$\begin{aligned}
\log\log(\frac{1}{\epsilon}\|Q(\hat{X}_t^i) - X_t^i\|) \cdot O(R^{-d}) \\
+ \log\log(\frac{1}{\epsilon}\|\hat{X}_t^i - \hat{X}_t^j\|) \cdot O(R^{-d}) \\
+ \log\log(\frac{1}{\epsilon}\|Q(\hat{X}_t^j) - X_t^j\|) \cdot O(R^{-d}) \leq O\left(\frac{\log\log T}{T^3}\right) \leq O\left(\frac{1}{T^2}\right).
\end{aligned}$$

Note that we do not need to union bound over all choices of i and j, since we have just one interaction and the above upper bound holds for any i and j. By the law of total probability (to remove conditioning) and the union bound we get that the total probability of failure, either due to not being able to apply quantization or by failure of quantization algorithm itself is at most $O\left(\frac{1}{T^2}\right)$. Finally we use chain rule to get that

$$\begin{aligned}
Pr[\cup_{t=1}^T \mathcal{L}_t] = \prod_{t=1}^T Pr[\mathcal{L}_t | \cup_{s=0}^{t-1} \mathcal{L}_s] = \prod_{t=1}^T \Big(1 - Pr[\neg\mathcal{L}_t | \cup_{s=0}^{t-1} \mathcal{L}_s]\Big) \\
\geq 1 - \sum_{t=1}^T Pr[\neg\mathcal{L}_t | \cup_{s=0}^{t-1} \mathcal{L}_s] \geq 1 - O\left(\frac{1}{T}\right).
\end{aligned}$$

In the end we would like to emphasize that we could get even better lower bound by scaling parameter R by constant factor. □

Lemma 5.5.20 *Let $T \geq 10n$, then for quantization parameters $R = 2 + T^{\frac{3}{d}}$ and $\epsilon = \frac{\eta H M}{(R^2+7)}$ we have that the expected number of bits used by Algorithm 11 per step is*

$$O\left(d \log\left(\frac{\rho_{max}^2}{\rho_{min}\lambda_2}\right)\right) + O\left(\log T\right).$$

Proof. If the initiator agent i and its neighbour j interact at step $t+1$, Corollary 5.2.1 (Please see (5.5)) gives us that the total number of bits used is at most

$$O\Big(d\log(\frac{R}{\epsilon}\|\hat{X}_t^i - \hat{X}_t^j\|)\Big) + O\Big(d\log(\frac{R}{\epsilon}\|Q(\hat{X}_t^j) - X_t^j\|)\Big) + O\Big(d\log(\frac{R}{\epsilon}\|Q(\hat{X}_t^j) - X_{t+1}^j\|)\Big).$$

By taking the randomness of agent interaction at step $t+1$ into the account, we get that the expected number of bits used is at most:

$$\sum_{i=1}^{n}\sum_{j\in\rho_i}\frac{1}{n\rho_i}\Bigg(O\Big(d\log(\frac{R}{\epsilon}\|\hat{X}_t^i-\hat{X}_t^j\|)\Big)+O\Big(d\log(\frac{R}{\epsilon}\|Q(\hat{X}_t^j)-X_t^j\|)\Big)$$
$$+O\Big(d\log(\frac{R}{\epsilon}\|Q(\hat{X}_t^j)-X_{t+1}^j\|)\Big)\Bigg). \qquad (5.20)$$

We proceed by upper bounding the first term:

$$\sum_{i=1}^{n}\sum_{j\in\rho_i}\frac{1}{n\rho_i}\Bigg(O\Big(d\log(\frac{R}{\epsilon}\|\hat{X}_t^i-\hat{X}_t^j\|)\Big)\Bigg)\leq\sum_{i=1}^{n}\sum_{j\in\rho_i}\frac{1}{n\rho_i}\Bigg(O\Big(d\log(\frac{R^2}{\epsilon^2}\|\hat{X}_t^i-\hat{X}_t^j\|^2)\Big)\Bigg)$$
$$\overset{Cauchy-Schwarz}{\leq}\sum_{i=1}^{n}\sum_{j\in\rho_i}\frac{1}{n\rho_i}\left(O\left(d\log\left(\frac{R^2}{\epsilon^2}(\|\hat{X}_t^i-\mu_t\|^2+\|\hat{X}_t^j-\mu_t\|^2)\right)\right)\right)$$
$$\overset{Jensen}{\leq}O\left(d\log\left(\frac{R^2}{\epsilon^2}\sum_{i=1}^{n}\sum_{j\in\rho_i}\frac{1}{n\rho_i}(\|\hat{X}_t^i-\mu_t\|^2+\|\hat{X}_t^j-\mu_t\|^2)\right)\right).$$

We have that

$$\sum_{i=1}^{n}\sum_{j\in\rho_i}\frac{1}{n\rho_i}(\|\hat{X}_t^i-\mu_t\|^2+\|\hat{X}_t^j-\mu_t\|^2)=\sum_{i=1}^{n}\frac{1}{n}\|\hat{X}_t^i-\mu_t\|^2+\sum_{i=1}^{n}\frac{1}{n}(\sum_{j\in\rho_i}\frac{1}{\rho_j})\|\hat{X}_t^j-\mu_t\|^2$$
$$\leq\sum_{i=1}^{n}\frac{1}{n}\|\hat{X}_t^i-\mu_t\|^2+\sum_{j=1}^{n}\frac{\rho_{max}}{\rho_{min}n}\|\hat{X}_t^j-\mu_t\|^2$$
$$\leq\frac{2\hat{\Gamma}_t\rho_{max}}{\rho_{min}n}.$$

By combining this with the previous inequality we get that

$$\sum_{i=1}^{n}\sum_{j\in\rho_i}\frac{1}{n\rho_i}\Bigg(O\Big(d\log(\frac{R}{\epsilon}\|\hat{X}_t^i-\hat{X}_t^j\|)\Big)\Bigg)\leq O\left(d\log\left(\frac{R^2\rho_{max}}{\epsilon^2\rho_{min}}(\frac{\hat{\Gamma}_t}{n})\right)\right)$$

Next, notice that

$$O\Big(d\log(\frac{R}{\epsilon}\|Q(\hat{X}_t^j)-X_{t+1}^j\|)\Big)\leq O\Big(d\log(\frac{R^2}{\epsilon^2}\|Q(\hat{X}_t^j)-X_{t+1}^j\|^2)\Big)$$
$$\overset{Cauchy-Schwarz}{\leq}O\Bigg(d\log\Big(\frac{R^2}{\epsilon^2}(\|Q(\hat{X}_t^j)-\hat{X}_t^j\|^2+\|\hat{X}_t^j-\mu_t\|^2$$
$$+\|\mu_t-\mu_{t+1}\|^2+\|X_{t+1}^j-\mu_{t+1}\|^2)\Big)\Bigg)$$
$$\leq O\Bigg(d\log\Big(\frac{R^2}{\epsilon^2}((R^2+7)^2\epsilon^2+\|\hat{X}_t^j-\mu_t\|^2$$
$$+\|\mu_t-\mu_{t+1}\|^2+\|X_{t+1}^j-\mu_{t+1}\|^2)\Big)\Bigg)$$

Where in the last step we used Corollary 5.2.1. By following similar argument as above we can upper bound the third term in (5.20):

$$\sum_{i=1}^{n}\sum_{j\in\rho_i}\frac{1}{n\rho_i}\Bigg(O\Big(d\log(\frac{R}{\epsilon}\|Q(\hat{X}_t^j)-X_{t+1}^j\|)\Big)\Bigg)$$
$$\leq O\Bigg(d\log\Big(\frac{R^2\rho_{max}}{\epsilon^2\rho_{min}}((R^2+7)^2\epsilon^2+\|\mu_t-\mu_{t+1}\|^2+\frac{\Gamma_{t+1}}{n}+\frac{\hat{\Gamma}_t}{n})\Big)\Bigg)$$

Analogously, by using $Q(\hat{X}_t^j)-X_t^j=(Q(\hat{X}_t^j)-\hat{X}_t^j)+(\hat{X}_t^j-\mu_t)+(\mu_t-X_t^j)$ we can upper bound the second term in (5.20):

$$\sum_{i=1}^{n}\sum_{j\in\rho_i}\frac{1}{n\rho_i}\Bigg(O\Big(d\log(\frac{R}{\epsilon}\|Q(\hat{X}_t^j)-X_t^j\|)\Big)\Bigg)$$
$$\leq O\Bigg(d\log\Big(\frac{R^2\rho_{max}}{\epsilon^2\rho_{min}}((R^2+7)^2\epsilon^2+\frac{\Gamma_t}{n}+\frac{\hat{\Gamma}_t}{n})\Big)\Bigg)$$

Hence, the expected number of bits used is at most

$$O\Bigg(d\log\Big(\frac{R^2\rho_{max}}{\epsilon^2\rho_{min}}((R^2+7)^2\epsilon^2+\|\mu_t-\mu_{t+1}\|^2+\frac{\Gamma_{t+1}}{n}+\frac{\Gamma_t}{n}+\frac{\hat{\Gamma}_t}{n})\Big)\Bigg),$$

since the above term is an upper bound for all the three terms in (5.20).

Next, we take the expectations of Γ_t, Γ_{t+1}, $\hat{\Gamma}_t$ and $\|\mu_t-\mu_{t+1}\|^2$ into the account. We get that the expected number of bits used is at most,

$$O\Bigg(d\mathbb{E}\Bigg[\log\Big(\frac{R^2\rho_{max}}{\epsilon^2\rho_{min}}((R^2+7)^2\epsilon^2+\|\mu_t-\mu_{t+1}\|^2+\frac{\Gamma_{t+1}}{n}+\frac{\Gamma_t}{n}+\frac{\hat{\Gamma}_t}{n})\Big)\Bigg]\Bigg)$$
$$\overset{Jensen}{\leq} O\Bigg(d\log\Big(\frac{R^2\rho_{max}}{\epsilon^2\rho_{min}}((R^2+7)^2\epsilon^2+\mathbb{E}\|\mu_t-\mu_{t+1}\|^2+\frac{\mathbb{E}[\Gamma_{t+1}]}{n}+\frac{\mathbb{E}[\Gamma_t]}{n}+\frac{\mathbb{E}[\hat{\Gamma}_t]}{n})\Big)\Bigg).$$

Notice that since $(R^2+7)^2\epsilon^2=\eta^2H^2M^2$, Lemma 5.5.9 gives us that both $\mathbb{E}[\Gamma_t]$ and $\mathbb{E}[\Gamma_{t+1}]$ are $O(\frac{\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2)$, Lemma 5.5.17 gives us that $\mathbb{E}[\hat{\Gamma}_t]=O(\frac{\rho_{max}^3}{\rho_{min}\lambda_2^2}(R^2+7)^2\epsilon^2)$ as well and finally Lemma 5.5.12 gives us that $\mathbb{E}\|\mu_t-\mu_{t+1}\|^2=O(\frac{(R^2+7)\epsilon^2}{n^2})$. Thus, by plugging these upper bounds in the above inequality we get that the expected number of bits used is at most

$$O\Bigg(d\log\Big(\frac{R^2\rho_{max}}{\epsilon^2\rho_{min}}((1+\frac{1}{n^2})(R^2+7)^2\epsilon^2+\frac{3\rho_{max}^3(R^2+7)^2\epsilon^2}{\rho_{min}\lambda_2^2})\Big)\Bigg)$$
$$=O\Bigg(d\log\Big(\frac{\rho_{max}^4(R^2+7)^2R^2}{\rho_{min}^2\lambda_2^2}\Big)\Bigg)$$
$$=O\Bigg(d\log\Big(\frac{\rho_{max}^2}{\rho_{min}\lambda_2}\Big)\Bigg)+O\left(d\log R\right)$$
$$=O\Bigg(d\log\Big(\frac{\rho_{max}^2}{\rho_{min}\lambda_2}\Big)\Bigg)+O\Big(d\log(T^{3/d})\Big)$$
$$=O\Bigg(d\log\Big(\frac{\rho_{max}^2}{\rho_{min}\lambda_2}\Big)\Bigg)+O\left(\log T\right).$$

□

The proof of Theorem 5.4.1 simply follows from using Lemmas 5.5.19 and 5.5.20, and plugging the value of $(R^2+7)\epsilon=\eta HM$ in Theorem 5.5.18.

5.6 Related Work

Decentralized optimization has a long history [Tsi84], and is related to the study of gossip algorithms, e.g. [KDG03, XB04, BGPS06]. Gossip is usually studied in one of two models [BGPS06]: *synchronous*, structured in global rounds, where each node interacts with a randomly chosen neighbor, forming a matching, and *asynchronous*, where each node wakes up at random times, e.g. given by a Poisson clock, and picks a random neighbor to interact with. Several classic optimization algorithms have been analyzed in the *asynchronous gossip model* [NO09, JRJ09, SS14]. In this chapter, we focus on analyzing decentralized SGD in this model.

As mentioned, the growing line of work on decentralized optimization for machine learning has mostly focused on variants of the *synchronous* gossip model. Specifically, [LZZ$^+$17] considered this setting in the context of DNN training, while and [TZG$^+$18] and [KLSJ20] also analyzed decentralized optimization with quantization in the *synchronous* model. [WJ21] and [KLB$^+$20] provided analysis frameworks for synchronous decentralized SGD with local updates, and possibly changing topologies.

[LZZL18] and [ALBR18] focused specifically on reducing *synchronization* costs in this setting, and proposed algorithms with *partially non-blocking communication*, in which nodes may read a *stale* version of the interaction partner's information, modelling e.g. a communication buffer. However, the maximum staleness must be bounded by a global variable τ, which must be enforced throughout the execution. Enforcing this bound can cause *blocking* in the system [ALBR18], and therefore the authors of these works propose to implement a relaxed round-based model, in which nodes interact once per round in perfect matchings. Their algorithms provide $O(1/\sqrt{Tn})$ convergence rates, under analytical assumptions.

Upon careful examination, we find that their analysis approach can be extended to the asynchronous gossip model we consider, by defining the "contact matrices" to correspond to pairwise interactions. However, this introduces two significant limitations. First, the analysis will not support *local gradient updates to models* nor *quantized communication*. If we remove these practical relaxations, our technique yields better bounds, as our potential analysis is specifically-tailored to this dynamic interaction model. Second, as we detail in Section 5.7, some of their technical conditions imply existence of global synchronization. For [ALBR18], as we detail in Section 5.7, their analysis would not guarantee any non-trivial speedup due to parallelization in the asynchronous gossip model.

[LDS20] provided a novel approach to analyze decentralized SGD with quantization and *limited asynchrony*: specifically, their algorithm requires *blocking* communication, i.e. nodes have to synchronize explicitly during interactions, but may see old versions of eachothers' models. More precisely, during each interaction, both parties are responsible for updating their local models, meaning that once node is woken up (we call it initiator node) and chooses interaction partner it has to block until the partner is woken up as well. In our case, initiator can update both its local model and the local model of its partner and proceed to the next step without blocking. [KLSJ20] use a similar update rule in the synchronous model. [ZY21] recently proposed a decentralized algorithm which is fully-asynchronous as long as node activation rates and message delays are bounded. As noted earlier, bounding activation rates does imply blocking; however, tolerating (bounded) message delays does improve over our approach of updating models using atomic writes. The setting further differs in that they assume that nodes compute full (non-stochastic) gradients, as well as that the loss function satisfies the PL condition.

In sum, we are the first to explicitly consider the *asynchronous gossip model*, and the impact of local updates, asynchrony, and quantization used *in conjunction* together with decentralized SGD. Our technique is new, and relies on a fine-grained analysis of individual interactions, and can yield improved bounds even in the case where $H = 1$. Further, our algorithm is the first to allow for both communication-compression and non-blocking communication.

5.7 Detailed Analytical Comparison

We compare our assumptions and the resulting bounds in more detail relative to [LZZL18], [ALBR18] and [LDS20]. We focus on these works since they are the only other papers which do not require explicit global synchronization in the form of rounds. (By contrast, e.g. [WJ21, KLB$^+$20] require that nodes synchronize in rounds, so that at every point in time every node has taken the same number of steps.)

5.7.1 Comparison with SGP

In [ALBR18], all nodes perform gradient steps at each iteration, in *synchronous* rounds, but averaging steps can be delayed by τ iterations. Unfortunately, the mixing time of their algorithm depends on the dimension d (more precisely, it contains a $\sqrt{d}$ factor). Moreover, it depends on the delay bound τ, and on Δ, defined as the number of iterations over which the interaction graph is well connected. Additionally, their analysis will not extend to the random interaction patterns required by the asynchronous gossip models. Practically, their analysis works for deterministic global interactions, but where nodes may see inconsistent versions of eachothers' models. As noted in [ALBR18], enforcing the τ bound inherently implies that the algorithm may have to block in case a slow node may cause this bound to be violated.

5.7.2 Comparison with AD-PSGD

[LZZL18] consider random interaction matrices and do not require the agents to perform the same number of gradient steps. Unlike our model, in their case more than two nodes can interact during the averaging step. Due to asynchronous model, like ours, [LZZL18] allow agents to have outdated views during the averaging step. We would like to emphasize that in their case outdated models *are assumed to come from the same step*.

More precisely, at every step t, there exists $\tau \leq \tau_{max}$ such that for every agent i, $\hat{X}_t^i = X_{t-\tau}^i$. As also noted by [ALBR18], enforcing this will require the usage of global barrier (or some alternate method of blocking while waiting for the nodes whose models are outdated by more than τ_{max} steps) once in every τ_{max} steps. Their implementation section suggests to explicitly implement synchronous pairings at every step.

In our case, at each step t and agent i, the delay τ_t^i is a random variable , such that $t - \tau_i$ is the last step node i was chosen as initiator. This implies naturally that $\hat{X}_t^i = X_{t-\tau}^i$, since $t - \tau_i$ is the last step node i updated its own model.

5.7.3 Comparison with Moniqua

[LDS20] consider a virtually identical model to AD-PSGD, but they also add quantization. The first difference between their work and ours is that we are using a random mixing matrix, thus we have to take the probability of models diverging (models diverge if the distance between them is larger then required by quantization algorithm) into account. Subsequently, this justifies our usage of [DGM$^+$21], since in this quantization method allows us to tolerate the larger distances between the models. This technical difference justifies the fact that our main bound has a non-trivial dependency on the second-moment bound M. As we showed, this dependency can be removed if we remove quantization. The second difference is that our interactions are one sided, that is, if i and j interact and i is initiator, i does not have to block while j is in compute.

5.7.4 Discussion

In summary, our algorithm is the first to explicitly consider the classic asynchronous gossip model [XB04], and show convergence bounds in its context. While AD-PSGD could be restated in this model, the corresponding bounds would be weaker than the ones we provide. To our knowledge we are the first to propose a fully non-blocking algorithm (assuming the access to atomic operations), which does not rely on an upper bound (probabilistic or deterministic) of $\tau_{\max}$ steps on the maximum delay between the nodes, and therefore remove the need for implementing global barrier-like communication to enforce $\tau_{\max}$. We would like to emphasize that we do not need to explicitly enforce an upper bound on the maximum delay, since it is inherently implied by our node activation model.

The price we pay for this added generality is that the rate given in Theorem 5.4.1 has a dependency on the second-moment bound. As we showed in Corollary 5.4.2, this requirement can be removed if communication is not quantized.

Bibliography

[AACH$^+$14] Dan Alistarh, James Aspnes, Keren Censor-Hillel, Seth Gilbert, and Rachid Guerraoui. Tight bounds for asynchronous renaming. *J. ACM*, 61(3):18:1–18:51, June 2014.

[AAD$^+$06] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed computing*, 18(4):235–253, 2006.

[AAF$^+$99] Yehuda Afek, Hagit Attiya, Arie Fouren, Gideon Stupp, and Dan Touitou. Long-lived renaming made adaptive. In *PODC*, pages 91–103, 1999.

[ABK$^+$18] Dan Alistarh, Trevor Brown, Justin Kopinsky, Jerry Z. Li, and Giorgi Nadiradze. Distributionally linearizable data structures. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, SPAA '18, pages 133–142, New York, NY, USA, 2018. ACM.

[ABKN18] Dan Alistarh, Trevor Brown, Justin Kopinsky, and Giorgi Nadiradze. Relaxed schedulers can efficiently parallelize iterative algorithms. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 377–386, New York, NY, USA, 2018. ACM.

[ABKU99] Yossi Azar, Andrei Z Broder, Anna R Karlin, and Eli Upfal. Balanced allocations. *SIAM journal on computing*, 29(1):180–200, 1999.

[ADSK18] Dan Alistarh, Christopher De Sa, and Nikola Konstantinov. The convergence of stochastic gradient descent in asynchronous shared memory. In *PODC*, pages 169–178, 2018.

[AGL$^+$17] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-efficient sgd via gradient quantization and encoding. In *NIPS*, pages 1709–1720, 2017.

[AH17] Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021*, 2017.

[AHJ$^+$18] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cédric Renggli. The convergence of sparsified gradient methods. In *NIPS*, pages 5977–5987, 2018.

[AKLN17] Dan Alistarh, Justin Kopinsky, Jerry Li, and Giorgi Nadiradze. The power of choice in priority scheduling. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 283–292. ACM, 2017.

[AKLS15] Dan Alistarh, Justin Kopinsky, Jerry Li, and Nir Shavit. The spraylist: A scalable relaxed priority queue. In *20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP 2015, San Francisco, CA, USA, 2015. ACM.

[AKY10] Yehuda Afek, Guy Korland, and Eitan Yanovsky. Quasi-linearizability: Relaxed consistency for improved concurrency. In *International Conference on Principles of Distributed Systems*, pages 395–410. Springer, 2010.

[ALBR18] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Michael Rabbat. Stochastic gradient push for distributed deep learning. *arXiv preprint arXiv:1811.10792*, 2018.

[ANK19] Dan Alistarh, Giorgi Nadiradze, and Nikita Koval. Efficiency guarantees for parallel incremental algorithms under relaxed schedulers. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 145–154. ACM, 2019.

[AW04] Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. J. W. & Sons, 2004.

[B$^+$15] Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[BCE$^+$12] Petra Berenbrink, Artur Czumaj, Matthias Englert, Tom Friedetzky, and Lars Nagel. Multiple-choice balanced allocation in (almost) parallel. In *APPROX-RANDOM*, pages 411–422. Springer, 2012.

[BCSV00] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: The heavily loaded case. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 745–754, New York, NY, USA, 2000. ACM.

[BFGS12] Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, and Julian Shun. Internally deterministic parallel algorithms can be fast. In J. Ramanujam and P. Sadayappan, editors, *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2012, New Orleans, LA, USA, February 25-29, 2012*, pages 181–192. ACM, 2012.

[BFH09] Petra Berenbrink, Tom Friedetzky, and Zengjian Hu. A new analytical method for parallel, diffusion-type load balancing. *J. Parallel Distrib. Comput.*, 69(1):54–61, January 2009.

[BFHM08] Petra Berenbrink, Tom Friedetzky, Zengjian Hu, and Russell Martin. On weighted balls-into-bins games. *Theor. Comput. Sci.*, 409(3):511–520, December 2008.

[BFK$^+$11] Dmitry Basin, Rui Fan, Idit Keidar, Ofer Kiselov, and Dmitri Perelman. CafÉ: Scalable task pools with adjustable fairness and contention. In *Proceedings of the 25th International Conference on Distributed Computing*, DISC'11, pages 475–488, Berlin, Heidelberg, 2011. Springer-Verlag.

[BFS12] Guy E Blelloch, Jeremy T Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 308–317. ACM, 2012.

[BGPS06] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE/ACM Trans. Netw.*, 14(SI):2508–2530, June 2006.

[BGSS16] Guy E Blelloch, Yan Gu, Julian Shun, and Yihan Sun. Parallelism in randomized incremental algorithms. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 467–478. ACM, 2016.

[BH14] Maciej Besta and Torsten Hoefler. Slim fly: A cost effective low-diameter network topology. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 348–359. IEEE, 2014.

[BJK$^+$96] Robert D Blumofe, Christopher F Joerg, Bradley C Kuszmaul, Charles E Leiserson, Keith H Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. *Journal of parallel and distributed computing*, 37(1):55–69, 1996.

[BL99] Robert D Blumofe and Charles E Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM (JACM)*, 46(5):720–748, 1999.

[Ble17] Guy E Blelloch. Some sequential algorithms are almost always parallel. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA*, pages 24–26, 2017.

[BT89] Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

[CDR15] Sorathan Chaturapruek, John C Duchi, and Christopher Ré. Asynchronous stochastic convex optimization: the noise is in the noise and sgd don't care. In *Advances in Neural Information Processing Systems*, pages 1531–1539, 2015.

[CF90] Neil Calkin and Alan Frieze. Probabilistic analysis of a parallel algorithm for finding maximal independent sets. *Random Structures & Algorithms*, 1(1):39–50, 1990.

[CRT87] Don Coppersmith, Prabhakar Raghavan, and Martin Tompa. Parallel graph algorithms that are efficient on average. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 260–269. IEEE, 1987.

[CSAK14] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *OSDI*, volume 14, pages 571–582, 2014.

[DBS17] Laxman Dhulipala, Guy Blelloch, and Julian Shun. Julienne: A framework for parallel graph algorithms using work-efficient bucketing. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '17, pages 293–304, New York, NY, USA, 2017. ACM.

[DBS21] Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. *ACM Trans. Parallel Comput.*, 8(1), apr 2021.

[DCM$^+$12] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[DGJ09] Camil Demetrescu, Andrew V Goldberg, and David S Johnson. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74. American Mathematical Soc., 2009.

[DGM$^+$21] Peter Davies, Vijaykrishna Gurunathan, Niusha Moshrefi, Saleh Ashkboos Ashkboos, and Dan Alistarh. Distributed variance reduction with optimal communication. In *Proceedings of the International Conference on Learning Representations, ICLR 2021. Full version available at arXiv:2002.09268*, 2021.

[Dij59] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DLM13] Dave Dice, Yossi Lev, and Mark Moir. Scalable statistics counters. In *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada*, pages 43–52, 2013.

[DP92] N. Deo and S. Prasad. Parallel heap: An optimal parallel priority queue. *The Journal of Supercomputing*, 6(1):87–98, March 1992.

[DSS06] Dave Dice, Ori Shalev, and Nir Shavit. Transactional locking ii. In *International Symposium on Distributed Computing*, pages 194–208. Springer, 2006.

[EHS12] Faith Ellen, Danny Hendler, and Nir Shavit. On the inherent sequentiality of concurrent objects. *SIAM J. Comput.*, 41(3):519–536, 2012.

[FN18] Manuela Fischer and Andreas Noever. Tight analysis of parallel randomized greedy mis. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2152–2160. SIAM, 2018.

[GDG$^+$17] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[GL13] Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[GLG$^+$12] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In Chandu Thekkath and Amin Vahdat, editors, *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 17–30. USENIX Association, 2012.

[GM96] Bhaskar Ghosh and S. Muthukrishnan. Dynamic load balancing by random matchings. *J. Comput. Syst. Sci.*, 53(3):357–370, December 1996.

[GNW16] George Giakkoupis, Yasamin Nazari, and Philipp Woelfel. How asynchrony affects rumor spreading time. In *35th ACM Symposium on Principles of Distributed Computing (PODC 2016)*, Chicago, United States, July 2016.

[HCC$^+$13] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *NIPS*, pages 1223–1231, 2013.

[HHH$^+$16] Andreas Haas, Thomas A. Henzinger, Andreas Holzer, Christoph M. Kirsch, Michael Lippautz, Hannes Payer, A. Sezgin, Ana Sokolova, and Helmut Veith. Local linearizability for concurrent container-type data structures. In *CONCUR*, 2016.

[HKP$^+$13] Thomas A. Henzinger, Christoph M. Kirsch, Hannes Payer, Ali Sezgin, and Ana Sokolova. Quantitative relaxation of concurrent data structures. *SIGPLAN Not.*, 48(1):317–328, January 2013.

[HLH$^+$13] Andreas Haas, Michael Lippautz, Thomas A. Henzinger, Hannes Payer, Ana Sokolova, Christoph M. Kirsch, and Ali Sezgin. Distributed queues in shared memory: multicore performance and scalability through quantitative relaxation. In Hubertus Franke, Alexander Heinecke, Krishna V. Palem, and Eli Upfal, editors, *Computing Frontiers Conference, CF'13, Ischia, Italy, May 14 - 16, 2013*, pages 17:1–17:9. ACM, 2013.

[HW90] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, July 1990.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IS15] Shams Imam and Vivek Sarkar. Load balancing prioritized tasks via work-stealing. In *European Conference on Parallel Processing*, pages 222–234. Springer, 2015.

[JRJ09] Björn Johansson, Maben Rabi, and Mikael Johansson. A randomized incremental subgradient method for distributed optimization in networked systems. *SIAM Journal on Optimization*, 20(3):1157–1170, 2009.

[JSY$^+$16] Mark C Jeffrey, Suvinay Subramanian, Cong Yan, Joel Emer, and Daniel Sanchez. Unlocking ordered parallelism with the swarm architecture. *IEEE Micro*, 36(3):105–117, 2016.

[JWG$^+$19] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, and Gennady Pekhimenko. Priority-based parameter propagation for distributed dnn training. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 132–145, 2019.

[KDG03] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 482–491. IEEE, 2003.

[KDSA08] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *2008 International Symposium on Computer Architecture*, pages 77–88. IEEE, 2008.

[KLB$^+$20] Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian U. Stich. A unified theory of decentralized sgd with changing topology and local updates. In *ICML*, pages 5381–5393, 2020.

[KLSJ20] Anastasia Koloskova, Tao Lin, Sebastian U Stich, and Martin Jaggi. Decentralized deep learning with arbitrary communication compression. In *International Conference on Learning Representations*, 2020.

[KRSJ19] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. Error feedback fixes SignSGD and other gradient compression schemes. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3252–3261. PMLR, 09–15 Jun 2019.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KZ93] R. M. Karp and Y. Zhang. Parallel algorithms for backtrack search and branch-and-bound. *Journal of the ACM*, 40(3):765–789, 1993.

[LDS20] Yucheng Lu and Christopher De Sa. Moniqua: Modulo quantized communication in decentralized SGD. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6415–6425. PMLR, 13–18 Jul 2020.

[LDS21] Yucheng Lu and Christopher De Sa. Optimal complexity in decentralized training. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7111–7123. PMLR, 18–24 Jul 2021.

[LHLL15] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[LHM$^+$18] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *ICLR, Poster*, 2018.

[Li14] Mu Li. Scaling distributed machine learning with the parameter server. In *Proceedings of the 2014 International Conference on Big Data Science and Computing*, BigDataScience '14, New York, NY, USA, 2014. Association for Computing Machinery.

[LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[LNDS20] Yucheng Lu, Jack Nash, and Christopher De Sa. Mixml: A unified analysis of weakly consistent parallel learning. *arXiv preprint arXiv:2005.06706*, 2020.

[LNP15] Andrew Lenharth, Donald Nguyen, and Keshav Pingali. Priority queues are not good concurrent priority schedulers. In *European Conference on Parallel Processing*, pages 209–221. Springer, 2015.

[LPLJ16] Rémi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. Asaga: asynchronous parallel saga. *arXiv preprint arXiv:1606.04809*, 2016.

[LS21] Dimitrios Los and Thomas Sauerwald. Balanced allocations with incomplete information: The power of two queries. *CoRR*, abs/2107.03916, 2021.

[LSPJ18] Tao Lin, Sebastian U Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't use large mini-batches, use local sgd. *arXiv preprint arXiv:1808.07217*, 2018.

[LW16] Christoph Lenzen and Roger Wattenhofer. Tight bounds for parallel randomized load balancing. *Distrib. Comput.*, 29(2):127–142, April 2016.

[LZZ$^+$17] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[LZZL18] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052. PMLR, 2018.

[Mit96] Michael David Mitzenmacher. *The Power of Two Random Choices in Randomized Load Balancing*. PhD thesis, PhD thesis, Graduate Division of the University of California at Berkley, 1996.

[Mit00] Michael Mitzenmacher. How useful is old information? *IEEE Transactions on Parallel and Distributed Systems*, 11(1):6–20, 2000.

[MPP$^+$15] Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. 07 2015.

[MS96] Maged M. Michael and Michael L. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 267–275, 1996.

[MS03] Ulrich Meyer and Peter Sanders. δ-stepping: a parallelizable shortest path algorithm. *Journal of Algorithms*, 49(1):114–152, 2003.

[NLP13] Donald Nguyen, Andrew Lenharth, and Keshav Pingali. A lightweight infrastructure for graph analytics. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 456–471, New York, NY, USA, 2013. ACM.

[NMC$^+$21] Giorgi Nadiradze, Ilia Markov, Bapi Chatterjee, Vyacheslav Kungurtsev, and Dan Alistarh. Elastic consistency: A practical consistency model for distributed stochastic gradient descent. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9037–9045, May 2021.

[NNvD$^+$18] Lam M. Nguyen, Phuong Ha Nguyen, Marten van Dijk, Peter Richtárik, Katya Scheinberg, and Martin Takác. SGD and hogwild! convergence without the bounded gradients assumption. In *ICML*, pages 3747–3755, 2018.

[NO09] Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48, 2009.

[NSD$^+$21] Giorgi Nadiradze, Amirmojtaba Sabour, Peter Davies, Shigang Li, and Dan Alistarh. Asynchronous decentralized SGD with quantized and local updates. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[PTW15] Yuval Peres, Kunal Talwar, and Udi Wieder. Graphical balanced allocations and the $1 +$ beta-choice process. *Random Struct. Algorithms*, 47(4):760–775, December 2015.

[PZC$^+$19] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A generic communication scheduler for distributed dnn training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 16–29, 2019.

[QAZX19] Aurick Qiao, Bryon Aragam, Bingjing Zhang, and Eric P. Xing. Fault tolerance in iterative-convergent machine learning. In *ICML*, pages 5220–5230, 2019.

[RAT] Adones Rukundo, Aras Atalar, and Philippas Tsigas. 2d-stack: A scalable lock-free stack design that continuously relaxes.

[RM51] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[RMS01] Andrea W Richa, M Mitzenmacher, and R Sitaraman. The power of two random choices: A survey of techniques and results. *Combinatorial Optimization*, 9:255–304, 2001.

[RRWN11] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, pages 693–701, 2011.

[RSD15] Hamza Rihani, Peter Sanders, and Roman Dementiev. Brief announcement: Multiqueues: Simple relaxed concurrent priority queues. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 80–82, New York, NY, USA, 2015. ACM.

[San98] P. Sanders. Randomized priority queues for fast parallel access. *Journal Parallel and Distributed Computing, Special Issue on Parallel and Distributed Data Structures*, 49:86–97, 1998.

[SBFG13] Julian Shun, Guy E. Blelloch, Jeremy T. Fineman, and Phillip B. Gibbons. Reducing contention through priority updates. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '13, pages 152–163, New York, NY, USA, 2013. ACM.

[SCJ18] Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. In *NIPS*, pages 4452–4463, 2018.

[SFJY14] F. Seide, H. Fu, L. G. Jasha, and D. Yu. 1-bit stochastic gradient descent and application to data-parallel distributed training of speech dnns. *Interspeech*, 2014.

[SGB$^+$14] Julian Shun, Yan Gu, Guy E Blelloch, Jeremy T Fineman, and Phillip B Gibbons. Sequential random permutation, list contraction and tree contraction are highly parallel. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 431–448. SIAM, 2014.

[SHM$^+$16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[SL00] Nir Shavit and Itay Lotan. Skiplist-based concurrent priority queues. In *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, pages 263–268. IEEE, 2000.

[SS14] Ohad Shamir and Nathan Srebro. Distributed stochastic optimization and learning. In *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 850–857. IEEE, 2014.

[Sti19] Sebastian U. Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2019.

[Str15] Nikko Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[SW16] Konstantinos Sagonas and Kjell Winblad. The contention avoiding concurrent priority queue. In *International Workshop on Languages and Compilers for Parallel Computing*, pages 314–330. Springer, 2016.

[SW17] Konstantinos Sagonas and Kjell Winblad. A contention adapting approach to concurrent ordered sets. *Journal of Parallel and Distributed Computing*, 2017.

[SZOR15] C. M. De Sa, C. Zhang, K. Olukotun, and C. Re. Taming the wild: A unified analysis of hogwild-style algorihms. In *Advances in Neural Information Processing Systems*, 2015.

[Tsi84] John Nikolas Tsitsiklis. Problems in decentralized decision making and computation. Technical report, Massachusetts Inst of Tech Cambridge Lab for Information and Decision Systems, 1984.

[TW07] Kunal Talwar and Udi Wieder. Balanced allocations: The weighted case. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 256–265, New York, NY, USA, 2007. ACM.

[TZG$^+$18] Hanlin Tang, Ce Zhang, Shaoduo Gan, Tong Zhang, and Ji Liu. Decentralization meets quantization. *arXiv preprint arXiv:1803.06443*, 2018.

[WGTT15] Martin Wimmer, Jakob Gruber, Jesper Träff, and Philippas Tsigas. The lock-free k-lsm relaxed priority queue. *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP*, 2015, 03 2015.

[WJ21] Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of local-update sgd algorithms. *Journal of Machine Learning Research*, 22(213):1–50, 2021.

[WSB$^+$06] Tim S Woodall, Galen M Shipman, George Bosilca, Richard L Graham, and Arthur B Maccabe. High performance rdma protocols in hpc. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 76–85. Springer, 2006.

[WWLZ18] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, pages 1306–1316, 2018.

[WWS$^+$18] Blake E Woodworth, Jialei Wang, Adam Smith, Brendan McMahan, and Nati Srebro. Graph oracle models, lower bounds, and gaps for parallel stochastic optimization. In *Advances in neural information processing systems*, pages 8496–8506, 2018.

[XB04] Lin Xiao and Stephen Boyd. Fast linear iterations for distributed averaging. *Systems & Control Letters*, 53(1):65–78, 2004.

[XHD$^+$15] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2):49–67, 2015.

[ZCL15] Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. In *Advances in neural information processing systems*, pages 685–693, 2015.

[ZY21] Jiaqi Zhang and Keyou You. Fully asynchronous distributed optimization with linear convergence in directed networks, 2021.