A Comprehensive Linear Speedup Analysis for Asynchronous Stochastic Parallel Optimization from Zeroth-Order to First-Order

Authored by:

Ji Liu
Cho-Jui Hsieh
Xiangru Lian
Yijun Huang
Huan Zhang

Abstract

Asynchronous parallel optimization received substantial successes and extensive attention recently. One of core theoretical questions is how much speedup (or benefit) the asynchronous parallelization can bring to us. This paper provides a comprehensive and generic analysis to study the speedup property for a broad range of asynchronous parallel stochastic algorithms from the zeroth order to the first order methods. Our result recovers or improves existing analysis on special cases, provides more insights for understanding the asynchronous parallel behaviors, and suggests a novel asynchronous parallel zeroth order method for the first time. Our experiments provide novel applications of the proposed asynchronous parallel zeroth order method on hyper parameter tuning and model blending problems.

1 Paper Body

Asynchronous parallel optimization received substantial successes and extensive attention recently, for example, [5, 25, 31, 33, 34, 37]. It has been used to solve various machine learning problems, such as deep learning [4, 7, 26, 36], matrix completion [25, 28, 34], SVM [15], linear systems [3, 21], PCA [10], and linear programming [32]. Its main advantage over the synchronous parallel optimization is avoiding the synchronization cost, so it minimizes the system overheads and maximizes the efficiency of all computation workers. One of core theoretical questions is how much speedup (or benefit) the asynchronous parallelization can bring to us, that is, how much time can we save by employing more computation resources? More precisely, people are interested in the running time speedup

(RTS) with T workers: RTS(T) =

running time using a single worker . running time using T workers

Since in the asynchronous parallelism all workers keep busy, RTS can be measured roughly by the computational complexity speedup (CCS) with T workers1 CCS(T) =

total computational complexity using a single worker ? T. total computational complexity using T workers

In this paper, we are mainly interested in the conditions to ensure the linear speedup property. More specifically, what is the upper bound on T to ensure CCS(T) = ?(T)? Existing studies on special cases, such as asynchronous stochastic gradient descent (ASGD) and asynchronous stochastic coordinate descent (ASCD), have revealed some clues for what factors can 1

For simplicity, we assume that the communication cost is not dominant throughout this paper.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

Table 1: Asynchronous parallel algorithms. ?I? and ?C? in ?model? stand for inconsistent and consistent read model respectively, which will be explained later. ?Base alg.? is short for base algorithm. Asyn. alg. ASGD [25] ASGD [1] ASGD[11] ASGD [18] ASGD [18] ARK [21] ASCD [20] ASCD [20] ASCD [19] ASCD [3] ASCD [3] ASCD [15] ASZD ASGD ASGD ASCD

problem type upper bound of T model smooth, strongly convex O(N 1/4) C smooth, convex O(K 1/4 min{? 3/2 , ? 1/2 }) C composite, convex O(K 1/4 ? 1/2) C smooth, nonconvex O(N 1/4 K 1/2 ?) I smooth, nonconvex O(K 1/2 ?) C Ax = b O(N) C smooth, convex, unconstrained O(N 1/2) C smooth, convex, constrained O(N 1/4) C composite, convex O(N 1/4) I T 1 T x Ax ? b x O(N) C 2 1 T x Ax ? bT x O(N 1/2) I 2 T 1/2 1 T x Ax ? b x , constrained O(N) I 2 zeroth order ? 3/2 1/2 2 SGD & SCD smooth, nonconvex O(?N + KN ?) I 1/2 ? 2) SGD smooth, nonconvex O(N 3/2 + KN I ? SGD smooth, nonconvex O(K? 2 + 1) C SCD smooth, nonconvex O(N 3/4) I base alg. SGD SGD SGD SGD SGD SGD SCD SCD SCD SCD SCD SCD

affect the upper bound of T . For example, Agarwal and Duchi [1] showed the upper bound depends on the variance of the stochastic gradient in ASGD; Niu et al. [25] showed that the upper bound depends on the data sparsity and the dimension of the problem in ASGD; and Avron et al. [3], Liu and Wright [19] found that the upper bound depends on the problem dimension as well as the diagonal dominance of the Hessian matrix of the objective. However, it still lacks a comprehensive and generic analysis to comprehend all pieces and show how these factors jointly affect the speedup property. This paper provides a comprehensive and generic analysis to study the speedup property for a broad range of asynchronous parallel stochastic algorithms from the zeroth order to the first order methods. To avoid unnecessary complication and cover practical problems and algorithms, we consider the following nonconvex stochastic optimization problem: minx?RN f (x) := E? (F (x; ?)), (1) where ? ? ? is a random variable, and both F (?; ?) : RN ? R and f (?) : RN ? R are smooth but not necessarily convex functions. This objective function covers a

large scope of machine learning problems including deep learning. F (?; ?)?s are called component functions in this paper. The most common specification is that ? is an index set of all training samples ? = {1, 2, ? ? ? , n} and F (x; ?) is the loss function with respect to the training sample indexed by ?. We highlight the main contributions of this paper in the following: ? We provide a generic analysis for convergence and speedup, which covers many existing algorithms including ASCD, ASGD (implementation on parameter server), ASGD (implementation on multicore systems), and others as its special cases. ? Our generic analysis can recover or improve the existing results on special cases. ? Our generic analysis suggests a novel asynchronous stochastic zeroth-order gradient descent (ASZD) algorithm and provides the analysis for its convergence rate and speedup property. To the best of our knowledge, this is the first asynchronous parallel zeroth order algorithm. ? The experiment includes a novel application of the proposed ASZD method on model blending and hyper parameter tuning for big data optimization. 1.1 Related Works We first review first-order asynchronous parallel stochastic algorithms. Table 1 summarizes existing linear speedup results for asynchronous parallel optimization algorithms mostly related to this paper. The last block of Table 1 shows the results in this paper. Reddi et al. [29] proved the convergence of asynchronous variance reduced stochastic gradient (SVRG) method and its speedup in sparse setting. Mania et al. [22] provides a general perspective (or starting point) to analyze for asynchronous stochastic algorithms, including H OGWILD !, asynchronous SCD and asynchronous sparse SVRG. The fundamental difference in our work lies on that we apply different analysis and our result can be 2

directly applied to various special cases, while theirs cannot. In addition, there is a line of research studying the asynchronous ADMM type methods, which is not in the scope of this paper. We encourage readers to refer to recent literatures, for example, Hong [14], Zhang and Kwok [35]. We end this section by reviewing the zeroth-order stochastic methods. We use N to denote the dimension of the problem, K to denote the iteration number, and ? to the ?variance of stochastic gradient. Nesterov and Spokoiny [24] proved a convergence rate of O(N/ K) for zeroth-order SGD applied ? to convex optimization. Based on [24], Ghadimi and Lan [12] proved a convergence rate of O(N/K) rate for ? zeroth-order SGD on nonconvex smooth problems. Jamieson et al. [16] shows a lower bound O(1/ K) ? for any zeroth-order method with inaccurate evaluation. Duchi et al. [9] proved a O(N 1/4 /K + 1/ K) rate for zeroth order SGD on convex objectives but with some? very different assumptions compared to our paper. Agarwal et al. [2] proved a regret of O(poly(N) K) for zeroth-order bandit algorithm on convex objectives. For more comprehensive review of asynchronous algorithms, please refer to the long version of this paper on arXiv:1606.00498. 1.2 Notation ? ei ? RN denotes the ith natural unit basis vector. ? E(?) means taking the expectation with respect to all random variables, while Ea (?) denotes the expectation with respect to a random variable a. ? ?f (x) ? RN is the gradient of f (x) with respect to x. Let S be a subset of {1, ? ? ? , N }. ?S f (x) ? RN is the projection of ?f (x) onto the index set S, that is, setting components of ?f (x) outside of S to be zero. We use ?i f (x) ?

RN to denote ?{i} f (x) for short. ? f ? denotes the optimal objective value in (1).

2

Algorithm

Algorithm

1

Generic

Asynchronous

Stochastic

We illustrate the asynchronous parallelism Algorithm (GASA) by assuming a centralized network: a cen- Require: x0 , K, Y, (?1 , ?2 , . . . , ?N), {?k }k=0,...,K?1 ? tral node and multiple child nodes (work?k is the step length for k th iteration ers). The central node maintains the opti- Ensure: {x }K k mization variable x. It could be a param- 1: for k = 0, k=0 . . . , K ? 1 do eter server if implemented on a computer 2: Randomly select a component function index ?k cluster [17]; it could be a shared memory and a set of coordinate indices Sk , where —Sk — = Y ; if implemented on a multicore machine. 3: xk+1 = xk ? ?k GSk (? xk ; ?k); Given a base algorithm A, all child nodes 4: end for run algorithm A independently and concurrently: read x from the central node (we call the result of this read x ?, and it is mathematically defined later in (4)), calculate locally using the x ?, and modify x on the central node. There is no need to synchronize child nodes. Therefore, all child nodes stay busy and consequently their efficiency gets maximized. In other words, we have CCS(T) ? RTS(T). Note that due to the asynchronous parallel mechanism the variable x in the central node is not updated exactly following the protocol of Algorithm A, since when a child node returns its computation result, the x in the central node might have been changed by other child nodes. Thus a new analysis is required. A fundamental question would be under what conditions a linear speedup can be guaranteed. In other words, under what conditions CCS(T) = ?(T) or equivalently RTS(T) = ?(T)? To provide a comprehensive analysis, we consider a generic algorithm A ? the zeroth order hybrid of SCD and SGD: iteratively sample a component function2 indexed by ? and a coordinate block S?{1, 2, ? ? ? , N }, where —S— = Y for some constant Y and update x with x ? x ? ?GS (x; ?)

(2) ?1

where GS (x; ?) is an approximation to the block coordinate stochastic gradient N Y ?S F (x; ?): ? (3) GS (x; ?) := i?S 2YN?i (F (x + ?i ei ; ?) ? F (x ? ?i ei ; ?))ei , S ? {1, 2, . . . , N }. th In the definition of GS (x; ?), ?i is the approximation parameter for the i coordinate. (?1 , ?2 , . . . , ?N) is predefined in practice. We only use the function value (the zeroth order information) to estimate GS (x; ?). It is easy to see that the closer to 0 the ?i ?s are, the closer GS (x; ?) and N Y ?1 ?S f (x; ?) will be. In particular, lim?i ?0,?i GS (x; ?) = N Y ?1 ?S f (x; ?). 2

The algorithm and theoretical analysis followed can be easily extended to the minibatch version.

3

Applying the asynchronous parallelism, we propose a generic asynchronous stochastic algorithm in Algorithm 1. This algorithm essentially characterizes how the value of x is updated in the central node. ?k is the predefined steplength (or learning rate). K is the total number of iterations (note that this iteration number is counted by the the central node, that is, any update on x no matter from which child node will increase this counter.) As we mentioned, the key difference of the asynchronous algorithm from the protocol of Algo?k may be not equal to xk . In asynchronous parallelism, there are two rithm A in Eq. (2) is that x different ways to model the value of x ?k : ? Consistent read: x ?k is some early existed state of x in the central node, that is, x ?k = xk??k for some ?k ? 0. This happens if reading x and writing x on the central node by any child node are atomic operations, for instance, the implementation on a parameter server [17]. ? Inconsistent read: x ?k could be more complicated when the atomic read on x cannot be guaranteed, which could happen, for example, in the implementation on the multi-core system. It means that while one child is reading x in the central node, other child nodes may be performing modifications on x at the same time. Therefore, different coordinates of x read by any child node may have different ages. In other words, x ?k may not be any existed state of x in the central node. Readers who want to learn more details about consistent read and inconsistent read can refer to [3, 18, 19]. To cover both cases, we note that x ?k can be represented in the following generic form: ? x ?k = xk ? j?J(k) (xj+1 ? xj), (4) where J(k) ? {k?1, k?2, . . . , k?T } is a subset of the indices of early iterations, and T is the upper bound for staleness. This expression is also considered in [3, 18, 19, 27]. Note that the practical value of T is usually proportional to the number of involved nodes (or workers). Therefore, the total number of workers and the upper bound of the staleness are treated as the same in the following discussion and this notation T is abused for simplicity.

3

Theoretical Analysis

Before we show the main results of this paper, let us first make some global assumptions commonly used for the analysis of stochastic algorithms.3 Bounded Variance of Stochastic Gradient E? (??F (x; ?) ? ?f (x)?2) ? ? 2 , ?x. Lipschitzian Gradient The gradient of both the objective and its component functions are Lipschitzian:4 max{??f (x) ? ?f (y)?, ??F (x; ?) ? ?F (y; ?)?} ? L?x ? y? ?x, ?y, ??.

(5)

Under the Lipschitzian gradient assumption, define two more constants Ls and Lmax . Let s be any positive integer bounded by N . Define Ls to be the minimal constant satisfying the following ? inequality: ??, ?x, ?i ei ?S ? {1, 2, ..., N } with —S— ? s for any z = i?S we have: max {??f (x) ? ?f (x + z)? , ??F (x; ?) ? ?F (x + z; ?)?} ? Ls ?z? Define L(i) for i ? {1, 2, . . . , N } as the minimum constant that satisfies: max{??i f (x) ? ?i f (x + ?ei)?, ??i F (x; ?) ? ?i F (x + ?ei ; ?)?} ? L(i) —?—. ??, ?x.

(6)

Define Lmax := maxi?{1,...,N } L(i) . It can be seen that Lmax ? Ls ?

L. Independence All random variables ?k , Sk for k = 0, 1, ? ? ? , K are independent to each other. Bounded Age Let T be the global bound for delay: J(k)?{k ? 1, . . . , k ? T }, ?k, so —J(k)— ? T . We define the following global quantities for short notations:) ((? ?) 2 N 3/2 2 2 2 := L /N, ? 4 + 4 T Y + Y T / N LT /(L2Y N), ? := ? 1 i=1 (i) i ?2 := Y /((f (x0) ? f ?)LY N),

?3 := (K(N ? + ? 2)?2 + 4)L2Y /L2T .

(7)

Next we show our main result in the following theorem: 3 Some underlying assumptions such as reading and writing a float number are omitted here. As pointed in [25], these behaviors are guaranteed by most modern architectures. 4 Note that the Lipschitz assumption on the component function F (x; ?)?s can be eliminated when it comes to first order methods (i.e., ? ? 0) in our following theorems.

4

Theorem 1 (Generic Convergence Rate for GASA). Choose the steplength ?k to be a constant ? in Algorithm 1 (?) ? ?k?1 = ? ?1 = 2LY N Y ?1 ?12 /(K(N ? + ? 2)?2 + ?1) + K(N ? + ? 2)?2 , ?k) (? ? N and suppose the age T is bounded by T ? 2Y 1/2 1 + 4Y ?1/2 N 1/2 ?3 ? 1 . We have the following convergence rate: ?K

k=0

E??f (xk)?2 1 20 ? + K K?2 K?2

(

L2T L2Y

?) ? ? 1 + 4Y ?1/2 N 1/2 ?3 ? 1 ? + 11 N ? + ? 2 K?2 + N ?. N Y ?1 (8)

Roughly speaking, the first term on the RHS of (8) is related to SCD; the second term is related to ?stochastic? gradient descent; and the last term is due to the zeroth-order approximation. Although this result looks complicated (or may be less elegant), it is capable to capture many important subtle structures, which can be seen by the subsequent discussion. We will show how to recover and improve existing results as well as prove the convergence for new algorithms using Theorem 1. To make the results more interpretable, we use the big-O notation to avoid explicitly writing down all the constant factors, including all L?s, f (x0), and f ? in the following corollaries. 3.1 Asynchronous Stochastic Coordinate Descent (ASCD) We apply Theorem 1 to study the asynchronous SCD algorithm by taking Y = 1 and ? = 0. Sk = {ik } only contains a single randomly sampled coordinate, and ? = 0 (or equivalently ?i = 0, ?i). The essential updating rule on x is xk+1 = xk ? ?k ?ik f (? xk). Corollary 2 (ASCD). Let ? = 0, ? = 0, and Y = 1 in Algorithm 1 and Theorem 1. If ? O(N 3/4),

(9)

) 2 E??f (x)? /K ? O(N/K). k k=0

(10)

T the following convergence rate holds: (? K

The proved convergence rate O(N/K) is consistent with the existing analysis of SCD [30] or ASCD for smooth optimization [20]. However, our requirement in (9) to ensure the linear speedup property is better than the one in [20], by

improving it from T ? O(N 1/2) to T ? O(N 3/4). Mania et al. [22] analyzed ASCD for strongly convex objectives and proved a linear speedup smaller than O(N 1/6), which is also more restrictive than ours. 3.2 Asynchronous Stochastic Gradient Descent (ASGD) ASGD has been widely used to solve deep learning [7, 26, 36], NLP [4, 13], and many other important machine learning problems [25]. There are two typical implementations of ASGD. The first type is to implement on the computer cluster with a parameter sever [1, 17]. The parameter server serves as the central node. It can ensure the atomic read or write of the whole vector x and leads to the following updating rule for x (setting Y = N and ?i = 0, ?i in Algorithm 1): xk+1 = xk ? ?k ?F (? xk ; ?k).

(11)

Note that a single iteration is defined as modifying the whole vector. The other type is to implement on a single computer with multiple cores. In this case, the central node corresponds to the shared memory. Multiple cores (or threads) can access it simultaneously. However, in this model atomic read and write of x cannot be guaranteed. Therefore, for the purpose of analysis, each update on a single coordinate accounts for an iteration. It turns out to be the following updating rule (setting Sk = {ik }, that is, Y = 1, and ?i = 0, ?i in Algorithm 1): xk+1 = xk ? ?k ?ik F (? xk ; ?k).

(12)

Readers can refer to [3, 18, 25] for more details and illustrations for these two implementations. Corollary 3 (ASGD in (11)). Let ? = 0 (or ?i = 0, ?i equivalently) and Y = N in Algorithm 1 and Theorem 1. If (?) T ? O K? 2 + 1 , (13) then the following convergence rate holds: (?) (?) K 2 + 1/K . (14) E??f (x)? /K ? O ?/ K k k=0 5

First note that the convergence rate in (14) is tight since it is consistent with the serial (nonparallel) version of SGD [23]. We compare this linear speedup property indicated by (13) with results in [1], [11], and [18]. ?To ensure such rate, Agarwal and Duchi [1] need T to be bounded by T ? O(K 1/4 min{? 3/2 , ?}), which is inferior to our result in (13). Feyzmahdavian et al. [11] need T to be bounded by ? 1/2 K 1/4 to achieve the same rate, which is also inferior to our result. Our requirement is consistent with the one in [18]. To the best of our knowledge, it is the best result so far. Corollary 4 (ASGD in (12)). Let ? = 0 (or equivalently, ?i = 0, ?i) and Y = 1 in Algorithm 1 and Theorem 1. If (?) T ?O N 3/2 + KN 1/2 ? 2 , (15) then the following convergence rate holds) (? (?) K 2 E??f (x)? /K ? O N/K? + N/K . k k=0

(16)

The additional factor N in (16) (comparing to (14)) arises from the different way of counting the iteration. This additional factor also appears in [25] and [18]. We first compare our result with [18], ? which requires T to be bounded by O(KN 1/2 ? 2). We can see that our requirement in (16) allows a larger value for T , especially when ? is small such that N 3/2 dominates KN 1/2 ? 2 . Next we compare with [25], which assumes that the objective function is strongly convex. Although this is sort of comparing ?apple? with ?orange?, it is still meaningful if one believes that the strong convexity would not affect the linear speedup property, which is implied by [22]. In [25], the linear speedup

is guaranteed if T ? O(N 1/4) under the assumption that the sparsity of the stochastic gradient is bounded by O(1). In comparison, we do not require the assumption of sparsity for stochastic gradient and have a better dependence on N . Moreover, beyond the improvement over existing analysis in [22] and [18], our analysis provides some interesting insights for asynchronous parallelism. Niu et al. [25] essentially suggests a large problem dimension N is beneficial to the linear speedup, while Lian et al. [18] and many others (for example, Agarwal and Duchi [1], Feyzmahdavian et al. [11]) suggest that a large stochastic variance ? (this often implies the number of samples is large) is beneficial to the linear speedup. Our analysis shows the combo effect of N and ? and shows how they improve the linear speedup jointly. 3.3 Asynchronous Stochastic Zeroth-order Descent (ASZD) We end this section by applying Theorem 1 to generate a novel asynchronous zeroth-order stochastic descent algorithm, by setting the block size Y = 1 (or equivalently Sk = {ik }) in GSk (? xk ; ?k) GSk (? xk ; ?k) = G{ik } (? xk ; ?k) = (F (? xk + ?ik eik ; ?k) ? F (? xk ? ?ik eik ; ?k))/(2?ik)eik .

(17)

To the best of our knowledge, this is the first asynchronous algorithm for zeroth-order optimization. Corollary 5 (ASZD). Set Y = 1 and all ?i ?s to be a constant ? in Algorithm 1. Suppose that ? satisfies (? {? ? }) ? ? O 1/ K + min ?(N K)?1/4 , ?/ N , (18) and T satisfies T ?O

(?

) N 3/2 + KN 1/2 ? 2 .

We have the following convergence rate (?) ( ) ? K 2 E??f (x)? /K ? O N/K + N/K? . k k=0

(19)

(20)

We firstly note that the convergence rate in (20) is consistent with the rate for the serial (nonparallel) zeroth-order stochastic gradient method in [12]. Then we evaluate this result from two perspectives. First, we consider T = 1, which leads to the serial (non-parallel) zeroth-order stochastic descent. Our result implies a better dependence on ?, comparing with [12].5 To obtain such convergence rate 5

Acute readers may notice that our way in (17) to estimate the stochastic gradient is different from the one used in [12]. Our method only estimates a single coordinate gradient of a sampled component function, while Ghadimi and Lan [12] estimate the whole gradient of the sampled component function. Our estimation is more accurate but less aggressive. The proved convergence rate actually improves a small constant in [12].

6

(?) in (20), Ghadimi and Lan [12] require ? ? O 1/(N K) , while our requirement in (18) is much less restrictive. An important insight in our requirement is to suggest the dependence on the variance ?: if the variance ? is large, ? is allowed to be a much larger value. This insight meets the common sense: a large variance means that the stochastic gradient may largely deviate from the true gradient, so we are allowed to choose a large ? to obtain a less

exact estimation for the stochastic gradient without affecting the convergence rate. From the practical view of point, it always tends to choose a large value for ?. Recall the zeroth-order method uses the function difference at two different points (e.g., x + ?ei and x ? ?ei) to estimate the differential. In a practical system (e.g., a concrete control system), there usually exists some system noise while querying the function values. If two points are too close (in other words ? is too small), the obtained function difference is dominated by noise and does not really reflect the function differential. Second, we consider the case T ? 1, which leads to the asynchronous zeroth-order stochastic descent. To the best of our knowledge, this is the first such algorithm. The upper bound for T in (19) essentially indicates the requirement for the linear speedup property. The linear speedup property here also shows that even if K? 2 is much smaller than 1, we still have O(N 3/4) linear speedup, which shows a fundamental understanding of asynchronous stochastic algorithms that N and ? can improve the linear speedup jointly.

4

Experiment

Since the ASCD and various ASGDs have been extensively validated in recent papers. We conduct two experiments to validate the proposed ASZD on in this section. The first part applies ASZD to estimate the parameters for a synthetic black box system. The second part applies ASZD to the model combination for Yahoo Music Recommendation Competition. 4.1 Parameter Optimization for A Black Box We use a deep neural network to simulate a black box system. The optimization variables are the weights associated with a neural network. We choose 5 layers (400/100/50/20/10 nodes) for the neural network with 46380 weights (or parameters) totally. The weights are randomly generated from i.i.d. Gaussian distribution. The output vector is constructed by applying the network to the input vector plus some Gaussian random noise. We use this network to generate 463800 samples. These synthetic samples are used to optimize the weights for the black box. (We pretend not to know the structure and weights of this neural network because it is a black box.) To optimize (estimate) the parameters for this black box, we apply the proposed ASZD method. The experiment is conducted on the machine (Intel Xeon architecture), which has 4 sockets and 10 cores for each socket. We run Algorithm 1 on various numbers of cores from 1 to 32 and the steplength is chosen as ? = 0.1, which is based on the best performance of Algorithm 1 running on 1 core to achieve the precision 10?1 for the objective value. Table 2: CCR and RTS of ASZD for different # of threads (synthetic data). thr-# 1

4

8

12

16

20

24

28

32

CCS 1 3.87 7.91 9.97 14.74 17.86 21.76 26.44 30.86 RTS 1 3.32 6.74 8.48 12.49 15.08 18.52 22.49 26.12

The speedup is reported in Table 2. We observe that the iteration speedup is almost linear while the running time speedup is slightly worse than the iteration speedup. We also draw Figure 1 (see the supplement) to show the curve of the objective value against the number of iterations and running time respectively. 4.2 Asynchronous Parallel Model Combination for Yahoo Music Recommendation Competition In KDD-Cup 2011, teams were challenged to predict user ratings in music given the Yahoo! Music data set [8]. The evaluation criterion is the Root Mean Squared Error (RMSE) of the test data set: ?? RMSE = ?ui)2 /—T1 —, (21) (u,i)?T1 (rui ? r where (u, i) ? T1 are all user ratings in Track 1 test data set (6,005,940 ratings), rui is the true rating for user u and item i, and r?ui is the predicted rating. The winning team from NTU created more than 200 models using different machine learning algorithms [6], including Matrix Factorization, k-NN, Restricted Boltzmann Machines, etc. They blend these models using Neural Network and Binned Linear Regression on the validation data set (4,003,960 ratings) to create a model ensemble to achieve better RMSE. 7

We were able to obtain the predicted ratings of N = 237 individual models on the KDD-Cup test data set from the NTU KDD-Cup team, which is a matrix X with 6,005,940 rows (corresponding to the 6,005,940 test data set samples) and 237 columns. Each element Xij indicates the j-th model?s predicted rating on the i-th Yahoo! Music test data sample. In our experiments, we try to linearly blend the 237 models using information from the test data set. Thus, our variable to optimize is a vector x ? RN as coefficients of the predicted ratings for each model. To ensure that our linear blending does not over-fit, we further split X randomly into two equal parts, calling them the ?validation? set (denoted as A ? Rn?N) for model blending and the true test set. We define our objective function as RMSE2 of the blended output on the validation set: f (x) = ?Ax ? r?2 /n where r is the corresponding true ratings in the validation set and Ax is the predicted ratings after blending. We assume that we cannot see the entries of r directly, and thus cannot compute the gradient of f (x). In our experiment, we treat f (x) as a blackbox, and the only information we can get from it is its value given a model blending coefficients x. This is similar to submitting a model for KDD-Cup and obtain a leader-board RMSE of the test set; we do not know the actual values of the test set. Then, we apply our ASZD algorithm to minimize f (x) with zero-order information only. Table 3: Comparing RMSEs on test data set with KDD-Cup winner teams NTU (1st) Commendo (2nd) InnerPeace (3rd) Our result RMSE

21.0004

21.0545

21.2335

21.1241

We implement our algorithm using Julia on a 10-core Xeon E7-4680 machine an run our algorithm for the same number of iterations, with different number of threads, and measured the running time speedup (RTS) in Figure 4 (see sup-

plement). Similar to our experiment on neural network blackbox, our algorithm has a almost linear speedup. For completeness, Figure 2 in supplement shows the square root of objective function value (RMSE) against the number of iterations and running time. After about 150 seconds, our algorithm running with 10 threads achieves a RMSE of 21.1241 on our test set. Our results are comparable to KDD-Cup winners, as shown in Table 3. Since our goal is to show the performance of our algorithm, we assume we can ?submit? our solution x for unlimited times, which is unreal in a real contest like KDD-Cup. However, even with very few iterations, our algorithm does converge fast to a reasonable small RMSE, as shown in Figure 3.

5

Conclusion

In this paper, we provide a generic linear speedup analysis for the zeroth-order and first-order asynchronous parallel algorithms. Our generic analysis can recover or improve the existing results on special cases, such as ASCD, ASGD (parameter implementation), ASGD (multicore implementation). Our generic analysis also suggests a novel ASZD algorithm with guaranteed convergence rate and speedup property. To the best of our knowledge, this is the first asynchronous parallel zeroth order algorithm. The experiment includes a novel application of the proposed ASZD method on model blending and hyper parameter tuning for big data optimization.

Acknowledgements This project is in part supported by the NSF grant CNS-1548078. We especially thank Chen-Tse Tsai for providing the code and data for the Yahoo Music Competition.

2 References

[1] A. Agarwal and J. C. Duchi. Distributed delayed stochastic optimization. NIPS, 2011. [2] A. Agarwal, D. P. Foster, D. J. Hsu, S. M. Kakade, and A. Rakhlin. Stochastic convex optimization with bandit feedback. In NIPS, pages 1035?1043, 2011. [3] H. Avron, A. Druinsky, and A. Gupta. Revisiting asynchronous linear solvers: Provable convergence rate through randomization. Journal of the ACM (JACM), 62(6):51, 2015. [4] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. The Journal of Machine Learning Research, 3:1137?1155, 2003. [5] S. Chaturapruek, J. C. Duchi, and C. R?. Asynchronous stochastic convex optimization: the noise is in the noise and SGD don?t care. In NIPS, pages 1531?1539, 2015. [6] P.-L. Chen, C.-T. Tsai, Y.-N. Chen, K.-C. Chou, C.-L. Li, C.-H. Tsai, K.-W. Wu, Y.-C. Chou, C.-Y. Li, W.-S. Lin, et al. A linear ensemble of individual and blended models for music rating prediction. In KDDCup, 2012. [7] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. NIPS, 2012.

8

[8] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The Yahoo! Music dataset and KDD-Cup?11. In KDDCup, pages 8?18, 2012. [9] J. C. Duchi,

P. L. Bartlett, and M. J. Wainwright. Randomized smoothing for stochastic optimization. SIAM Journal on Optimization, 22(2):674?701, 2012. [10] J. Fellus, D. Picard, and P. H. Gosselin. Asynchronous gossip principal components analysis. Neurocomputing, 2015. doi: 10.1016/j.neucom.2014.11.076. [11] H. R. Feyzmahdavian, A. Aytekin, and M. Johansson. An asynchronous mini-batch algorithm for regularized stochastic optimization. arXiv, 2015. [12] S. Ghadimi and G. Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. SIAM Journal on Optimization, 23(4):2341?2368, 2013. [13] K. Gimpel, D. Das, and N. A. Smith. Distributed asynchronous online learning for natural language processing. In CoNLL, pages 213?222, 2010. [14] M. Hong. A distributed, asynchronous and incremental algorithm for nonconvex optimization: An ADMM based approach. arXiv:1412.6058, 2014. [15] C. Hsieh, H. Yu, and I. S. Dhillon. PASSCoDe: Parallel ASynchronous Stochastic dual Coordinate Descent. In ICML, pages 2370?2379, 2015. [16] K. G. Jamieson, R. Nowak, and B. Recht. Query complexity of derivative-free optimization. In NIPS, 2012. [17] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. OSDI, 2014. [18] X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In NIPS, pages 2719?2727, 2015. [19] J. Liu and S. J. Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. arXiv:1403.3862, 2014. [20] J. Liu, S. J. Wright, C. R?, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. ICML, 2014. [21] J. Liu, S. J. Wright, and S. Sridhar. An asynchronous parallel randomized kaczmarz algorithm. arXiv, 2014. [22] H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M. I. Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. arXiv:1507.06970, 2015. [23] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. SIAM Journal on Optimization, 19(4):1574?1609, 2009. [24] Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. Foundations of Computational Mathematics, pages 1?40, 2011. [25] F. Niu, B. Recht, C. Re, and S. Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. NIPS, 2011. [26] T. Paine, H. Jin, J. Yang, Z. Lin, and T. Huang. Gpu asynchronous stochastic gradient descent to speed up neural network training. NIPS, 2013. [27] Z. Peng, Y. Xu, M. Yan, and W. Yin. Arock: an algorithmic framework for asynchronous parallel coordinate updates. arXiv, 2015. [28] F. Petroni and L. Querzoni. Gasgd: stochastic gradient descent for distributed asynchronous matrix completion via graph partitioning. ACM Conference on Recommender systems, 2014. [29] S. J. Reddi, A. Hefny, S. Sra, B. P?czos, and A. J. Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In NIPS, pages 2629?2637, 2015. [30] P. Richt?rik and M. Tak??c. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. Mathematical Programming, 144(1-2):1?38, 2014. [31] C. D. Sa, C. Zhang, K. Olukotun, C. R?, and C. R?. Taming the wild: A unified analysis of hogwild-style algorithms. In NIPS, pages 2674?2682, 2015. [32] S.

Sridhar, S. Wright, C. Re, J. Liu, V. Bittorf, and C. Zhang. An approximate, efficient LP solver for lp rounding. NIPS, 2013. [33] J. Wei, W. Dai, A. Kumar, X. Zheng, Q. Ho, and E. P. Xing. Consistent bounded-asynchronous parameter servers for distributed ml. arXiv:1312.7869, 2013. [34] H. Yun, H.-F. Yu, C.-J. Hsieh, S. Vishwanathan, and I. Dhillon. Nomad: Non-locking, stochastic multimachine algorithm for asynchronous and decentralized matrix completion. arXiv:1312.0193, 2013. [35] R. Zhang and J. Kwok. Asynchronous distributed ADMM for consensus optimization. ICML, 2014. [36] S. Zhang, A. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. arXiv, 2014. [37] S. Zhao and W. Li. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In AAAI, pages 2379?2385, 2016.

9