2017 16th IEEE International Conference on Machine Learning and Applications

A Hierarchical, bulk-synchronous stochastic gradient descent algorithm for deep-learning applications on GPU clusters

Guojing Cong, Onkar Bhardwaj
IBM TJ Watson Research Center
1101 Kitchawan Road, Yorktown Heights, NY, 10598
{gcong,obhardw}@us.ibm.com,

***Abstract*—The training data and models are becoming increasingly large in many deep-learning applications. Large-scale distributed processing is employed to accelerate training. Increasing the number of learners in synchronous and asynchronous stochastic gradient descent presents challenges to convergence and communication performance.**

We present our hierarchical, bulk-synchronous stochastic gradient algorithm that effectively balances execution time and accuracy for training in deep-learning applications on GPU clusters. It achieves much better convergence and execution time at scale in comparison to asynchronous stochastic gradient descent implementations. When deployed on a cluster of 128 GPUs, our implementation achieves up to 56 times speedups over the sequential stochastic gradient descent with similar test accuracy for our target application.

I. Introduction

Many current deep-learning applications with large inputs take a very long time to train. Efficient parallelization at scale is critical to accelerating long-running machine learning applications.

Properties of stochastic gradient descent algorithms including parallel and distributed variants have been established (e.g., see [8], [1], [6], [7], [3], [9], [4], [5]). The convergence rates of SGD for non-convex and convex problems are $\mathrm{O}\left(1/\sqrt{S}\right)$ and $\mathrm{O}(1/S)$, respectively, with S being the number of processed samples. Synchronous SGD with p learners has the convergence rate of $\mathrm{O}\left(1/\sqrt{pK}\right)$ for non-convex objectives, with K being the number of samples processed by each learner [3]. Hogwild!, a lockfree implementation of ASGD, is shown to converge for strongly convex problems with theoretical linear speedup over SGD [7]. *Downpour* [2], another influential ASGD implementation with resilience against machine failures, also converges for strongly convex problems. With relatively mild assumptions of the network and the objective function, Lian *et al.* [5] show that ASGD converges and achieves asymptotic linear speedup over SGD for non-convex problems if the gradient staleness is bounded by the number of learners.

We propose a new parallelization of stochastic gradient descent that performs better in terms of both execution time and accuracy at scale than ASGD implementations for our target application. We call this algorithm hierarchical, bulk-synchronous SGD (*HierSGD*). As *HierSGD* is bulk-synchronous, it allows for sparse gradient aggregation among learners to effectively minimize the communication overhead. Instead of a parameter server, the learners in *HierSGD* communicate their learned gradients with each other at regular intervals through global reductions. Since the communication interval T is explicit, the staleness of gradients can be precisely controlled. The communication time is amortized among the data samples processed within each interval and becomes negligible if T is large enough. Compared to asynchronous updates to a parameter server, global reduction minimizes the amount of data transported in the system.

We evaluate the practical efficiency of *HierSGD* for a deep learning application using three different neural network models on a 128-GPU cluster. Our experiments demonstrate the superior performance of *HierSGD* over two popular ASGD implementations: *Downpour* [2] and *EAMSGD* [10]. In *EAMSGD*, global gradient aggregation among learners simulates an elastic force that links the parameters they compute with a center variable stored by the parameter server.

On our target platform, with a moderate number of learners, for example, 8 and 16 learners distributed on 2 and 4 nodes respectively, *HierSGD* achieves similar practical efficiency as *Downpour* and *EAMSGD*. With more than 32 learners, *HierSGD* significantly reduces the communication overhead compared to *Downpour* and *EAMSGD* while achieving much better accuracy. With 128 learners, *HierSGD* is up to 12 times faster than *Downpour* and *EAMSGD*, and achieves much better accuracy with the same amount of data samples processed. *HierSGD* achieves better accuracy than SGD with 8 or 16 GPUs for some models. With 128 GPUs, it is up to 56 times faster than the sequential SGD while achieving similar test accuracy.

II. Hierarchical, bulk-synchronous SGD

In addressing the challenges faced by ASGD at scale, we propose a hierarchical, bulk-synchronous SGD algorithm (*HierSGD*). *HierSGD* does not employ a parameter server for gradient aggregation. Instead, it adopts a bulk-synchronous parallel paradigm, and the learners aggregate their accumulated gradients through reduction after T minibatches have been processed. Compared with a parameter se

0-7695-6321-X/17/31.00 ©2017 IEEE
DOI 10.1109/ICMLA.2017.00-56

has two advantages on HPC system. The first is that fast reduction does not demand high bisectional bandwidth as would a parameter server. The second is that the staleness of gradients are explicitly controlled, thus the stochastic and inconsistency associated with a parameter server are avoided.

HierSGD is recursive, and constructs a hierarchy of learners. At the lowest level (level 1), the learners are sequential SGD solvers. These learners are divided into groups. Within a group, they form a distributed learner (level-2 learner) by aggregating gradients in a bulk-synchronous fashion. Level-3 learners are constructed by aggregating gradients learned by level-2 learners. Higher-level learners can be constructed in a similar fashion. For a level-h learner, the level-$(h-1)$ learners used in the construction are its children. Alg. 1 gives a formal description of *HierSGD*.

In Alg. 1, first a l-level hierarchy is built on top of the SGD solver set $\mathcal{P}$. At each level h ($2 \leq h \leq l$), $\mathcal{G}_h[i]$ is the i^{th} level-h learner with parameter $x_{h,i}$ and learning rate $\gamma_{h,i}$. The function *number-of-children* returns the number of level-$(h-1)$ learners used to construct $\mathcal{G}_h[i]$. The t-th child of $\mathcal{G}_h[i]$ is its t-th level-$(h-1)$ learner. The gradient aggregation interval is $\mathcal{T}_h$. At the highest level l, $|\mathcal{G}_l|$=1. That is, there is only 1 l-level learner.

Algorithm 1 *HierSGD* ($\mathcal{S}$: data samples, $\mathcal{P}$: set of SGD learners, l: number of levels)

$\mathcal{G}$, $\mathcal{T}$, x, $\gamma \leftarrow$ build-hier ($\mathcal{P}$, l)
for k=1, K' **do**
  hier-iter ($\mathcal{S}$, $\mathcal{G}$, $\mathcal{T}$, l, 1)
end for

Algorithm 2 *Hier-iter* ($\mathcal{S}$, $\mathcal{G}$, $\mathcal{T}$, h, i)

for t=1, number-of-children($\mathcal{G}_h[i]$) in parallel **do**
  $\mathcal{G}_{h-1}[n] := t-$th child of $\mathcal{G}_h[i]$
  $x_{h-1,n} \leftarrow x_{h,i}$, $g_n^s \leftarrow 0$, $j \leftarrow 0$
  while $j < \mathcal{T}_h$ **do**
    if $h = 2$ **then**
      g_n compute gradient g for a randomly sampled minibatch from $\mathcal{S}$ using $x_{1,t}$
      $x_{1,t} \leftarrow x_{1,t} - \gamma_{1,t} * g_n$
    else
      $g_n \leftarrow$ hier-iter ($\mathcal{S}$, $\mathcal{G}$, $\mathcal{T}$, $h-1$, n)
    end if
    $g_n^s \leftarrow g_n^s + g_i$, $j \leftarrow j+1$
  end while
end for
$g^s \leftarrow \sum_{n=1}^{|\mathcal{G}_h[i]|} g_n^s$, $x_{h,i} \leftarrow x_{h,i} - \gamma_{h,i} * g^s$
return g^s

Hier-iter, described in Alg. 2, at level h runs $\mathcal{T}_h$ steps of gradient descent with the children of $\mathcal{G}_h[i]$, level-$(h-1)$ learners. If $h = 2$, regular SGD is used. The learner $\mathcal{G}_h[i]$ updates its local parameter $x_{h,i}$ using the aggregated gradient g^s from all children.

To show the convergence of *HierSGD*, we first consider the convergence behavior of *HierSGD* with $l = 2$ (*HierSGD*$_2$). A non-recursive version of *HierSGD*$_2$ in SPMD style is given in Alg. 3. In Alg. 3, p is the number of SGD learners, and id is the SGD learner ID ranging from 1 to p. Collective *broadcast* replicates the parameter x_2 at level 2 to all SGD learners. After *allreduce*(g_s), each learner gets the aggregated gradient stored in g_s.

Algorithm 3 *HierSGD*$_2$ ($\mathcal{S}$, T, p, id, γ, γ_2, K)

$g_s \leftarrow 0$
if $id = 1$ **then**
  initialize parameter x_2
end if
for $i \leq K$ **do**
  $x_1 \leftarrow$ *broadcast*(x_2)
  for j=1, T **do**
    compute gradient g for a random minibatch sampled from $\mathcal{S}$
    $x_1 \leftarrow x_1 - \gamma * g$, $g_s \leftarrow g_s + g$
  end for
  $g_s \leftarrow$ *allreduce*(g_s), $x_2 \leftarrow x_2 - \gamma_2 g_s$, $g_s \leftarrow 0$
end for

HierSGD$_2$ is a distributed learner constructed directly with SGD. It maintains its own parameter x_2 using a learning rate γ_2. Each SGD solver has its own local parameter x_1, and uses a learning rate γ. T is the aggregation interval, M is the minibatch size, and K is the total number of global gradient aggregations.

Learning rate γ is the step size for local updates within an aggregation interval, while γ_p is the step size for global aggregation. Each learner accumulates gradients learned within an interval into g_s. Global aggregation aggregates g_s from all learners through *allreduce*. The parameter x_2 is initialized by *HierSGD*$_2$.

The asymptotic convergence guarantee of *HierSGD*$_2$ is omitted due to limited space.

Since *HierSGD*$_2$ is itself a learner with convergence rate $\mathrm{O}\left(1/\sqrt{S}\right)$, by induction on the hierarchy level, it can be shown that each learner at level $h > 2$ converges, and ultimately *HierSGD* converges.

When deployed on the target platform, *HierSGD* can adapt to the architecture, and balance convergene and communication overhead through the right hierarchy of learners. The average expected gradient norm $\bar{R}_K$ for *HierSGD*$_2$ as convergence guarantee decreases as T decreases, yet on a large GPU cluster, frequent aggregation (small T) incurs large communication overhead. Since there is a natural hiearchy in the organization of computing resources in most computer systems, more frequent aggregations can be applied to a group of learners that have large communication bandwidth among them, and less frequent aggregations can be applied to learners that have relatively small communication bandwidth among them. As a result, communication overhead and convergence can both be improved.

III. Performance evaluation

We evaluate the performance of *HierSGD* on our target platform. We first study the scaling of *HierSGD*$_2$ in terms of execution time and accuracy in comparison to ASGD implementations. We then evaluate the impact of deeper hierarchies on the performance of *HierSGD* at scale on the 128-GPU cluster.

In order to compare with ASGD implementations, unless noted other wise, we run *HierSGD* for 600 epochs with 8, 16, 32, 64, and 128 GPUs. Note that the sequential SGD runs for 300 epochs.

A. HierSGD$_2$ *scaling*

With *HierSGD*$_2$, each GPU in the cluster runs a sequential SGD learner. At interval T, they aggregate accumulated gradients and compute new parameters. To balance communication with computation, different T is used for *vgg*, *nin*, and *convnet*, as they have different network size and structure. We choose T=16, 4, and 4 for *vgg*, *nin*, and *convnet*, respectively.

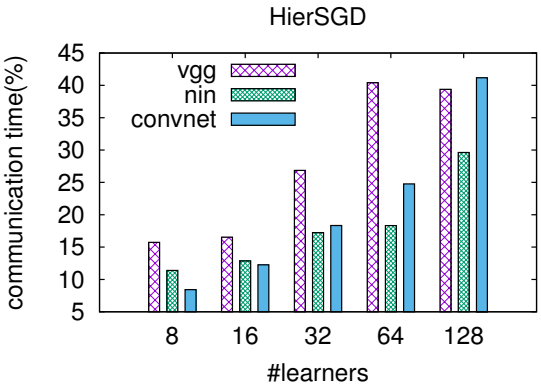

Fig. 1: Communication scaling

We collect the percentage of epoch time spent on communication with 8, 16, 32, 64, and 128 learners. Fig. 1 shows the scaling of communication performance for *HierSGD*$_2$. At 8 and 16 learners, the communication time is below 20%. Even at 128 learners, it is still below 45% for all models. In fact, the communication overhead for *nin* is consistently below 30%. The communication overhead is largest for *vgg* except at 128 learners. At 128 learners, the communication overhead for *convnet* is the largest among the three models.

With regard to accuracy, the other metric for measuring practical efficiency, *HierSGD*$_2$ performs quite well with 8, 16, and 32 learners. Fig. 2 shows the accuracy gap between *HierSGD*$_2$ and the sequential SGD. With 8 learners, the gap is negative for all three models. This means *HierSGD*$_2$ actually achieves better accuracy than the sequential SGD. With 16 learners, *HierSGD*$_2$ and SGD achieve similar accuracy for *vgg*. *HierSGD*$_2$ achieves slightly better accuracy than SGD for *nin*. For *convnet*, *HierSGD*$_2$ achieves significantly better (by 1.83%) accuracy than SGD. *HierSGD*$_2$ and SGD achieve comparable accuracies with 32 learners. As the number of

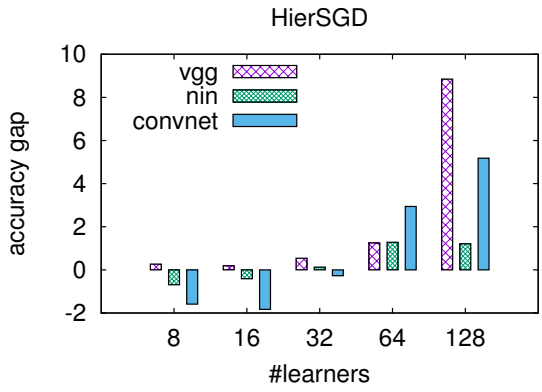

Fig. 2: Accuracy gap

learners reaches 64 and 128, significant accuracy degradation, up to 8.8% and 4.2%, is observed for *vgg* and *convnet*, respectively. Interestingly, the accuracy degradation for *nin* is still within 1.3% with 128 learners.

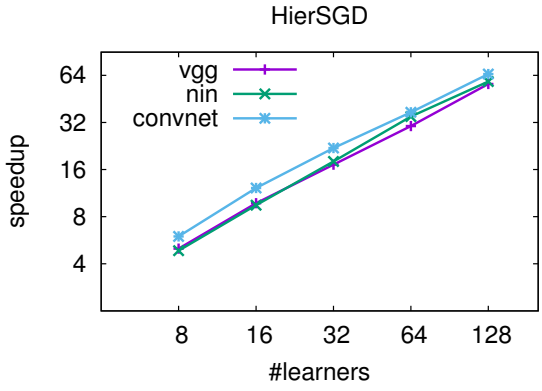

Fig. 3: Epoch time speedup – in log-log plot

Fig. 3 shows the epoch time speedup of *HierSGD*$_2$ against SGD with 8, 16, 32, 64, and 128 learners. With 8 learners, the speedups for *vgg*, *nin*, and *convnet* are 4.9, 4.8, and 5.9, respectively. With 128 learners, the speedups for *vgg*, *nin*, and *convnet* are 56.3, 58.3, and 65.1, respectively. Since twice as many epochs are run for *HierSGD*$_2$ in comparison to SGD, the wallclock time speedups are half of these numbers.

In machine learning applications, there are scenarios where one percentage point is considered as a big difference in accuracy. In other cases the speed to train the model is relatively more important. The aggregation interval in T can be used to compromise between accuracy and training speed.

Finally, we evaluate the speedup and accuracy gap achieved with *HierSGD*$_3$ against the sequential implementation. In this experiment, both *HierSGD*$_3$ and the sequential SGD run for 300 epochs. For *vgg*, *nin*, and *convnet*, we experiment with *HierSGD*$_3$ with $(T_1, T_2) = (4, 16)$, $(2, 4)$ and $(4, 16)$ respectively. The results are shown in Tab. I.

Model	*vgg*	*nin*	*convnet*
Impl	*HierSGD*$_3$ (4, 16)	*HierSGD*$_3$ (2, 4)	*HierSGD*$_3$ (4, 16)
Speedup	46.87	55.55	21.05
Gap	2.06%	1.06%	3.26%

TABLE I: Speedup and accuracy gap after 300 epochs

In Tab. I, the speedups achieved for *vgg*, *nin*, and *convnet* are 46.87, 55.55, and 21.05, respectively, and the accuracy gaps are 2.06%, 1.06%, and 3.26%, respectively.

IV. Conclusion and future work

Deep learning applications that analyze large inputs of texts, speeches, images, and videos can take a long time to train with stochastic gradient descent. Efficient parallelization at scale is needed to reduce the execution time and also retain the accuracy for the trained model. The scaling of synchronous SGD is limited by the minibatch size. Although in theory ASGD converges with asymptotic linear speedup over SGD, in practice ASGD at scale faces serious challenges: large communication overhead and high sample complexity to reach convergence. In our experiments, two popular ASGD implementations, *Downpour* and *EAMSGD* perform poorly, in terms of both execution time and accuracy, at 128 GPUs on our target platform.

We propose a bulk-synchronous, distributed SGD algorithm, *HierSGD*, that balances the communication overhead and convergence. *HierSGD* constructs a hierarchy of learners with SGD solvers at the bottom level. The explicit gradient aggregation interval T in *HierSGD*$_2$ amortizes the communication cost, and gradient aggregation through collective *allreduce* is more efficient than the complete bipartite communication pattern between learners and sharded paramater servers in ASGD. In our experiments, with 64 and 128 GPUs, *HierSGD*$_2$ is up to 10 and 16 times faster than *Downpour* and *EAMSGD*, respectively. *HierSGD*$_2$ also achieves much better accuracy than the ASGD implementations.

HierSGD with deeper hierarchies can further improve the convergence and reduce the communication overhead. *HierSGD* can be deployed on to the target platform to benefit from the architectural features. With 128 GPUs, in our implementation, *HierSGD*$_3$ constructs 32 *HierSGD*$_2$ learners, and each *HierSGD*$_2$ learner uses 4 GPUs in the Minsky node. Our experiments show that *HierSGD*$_3$ has the fastest practical convergence speed for the *vgg*, *nin*, and *convnet* models. Compared with the sequential implementation, *HierSGD*$_3$ achieves up to 56 times speedup with similar accuracy.

In our future work we will continue to evaluate other schemes for building a hierarchy of learners. For example, depending on the network topology, more levels of hierarchies may be introduced. In our current implementation the hierarchy forms a tree structure. Other structures may be beneficial for different types of communication networks. We will also evaluate *HierSGD* for deep-learning applications with larger inputs.

References

[1] L. Bottou. Online learning and stochastic approximations, 1998.

[2] J. Dean, G. Corrado, R. Monga, and et al. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1223–1231. Curran Associates, Inc., 2012.

[3] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(Jan):165–202, 2012.

[4] S. Ghadimi and G. Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[5] X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[6] 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.

[7] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[8] H. Robbins and D. Siegmund. A convergence theorem for non negative almost supermartingales and some applications. In *Herbert Robbins Selected Papers*, pages 111–135. Springer, 1985.

[9] O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *ICML (1)*, pages 71–79, 2013.

[10] S. Zhang, A. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 685–693, 2015.