CrossBow: Scaling Deep Learning on Multi-GPU Servers

Alexandros Koliousis[†], Pijika Watcharapichat[†], Matthias Weidlich[*], Paolo Costa[‡], Peter Pietzuch[†]

[†]Imperial College London [‡]Microsoft Research [*]Humboldt-Universität zu Berlin

ABSTRACT

With the widespread availability of servers with 4 or more GPUs, scalability in terms of the number of GPUs in a server when training deep learning models becomes a paramount concern. Systems such as TensorFlow and MXNet train using synchronous stochastic gradient descent—an input batch is partitioned across the GPUs, each computing a partial gradient. The gradients are then combined to update the model parameters before proceeding to the next batch. For many deep learning models, this introduces a scalability challenge: to keep multiple GPUs fully utilised, the batch size must be sufficiently large, but a large batch size slows down model convergence due to the less frequent model updates, thus prolonging the time to reach a desired level of accuracy.

This paper introduces CrossBow, a new single-server multi-GPU deep learning system that avoids the above trade-off. CrossBow trains multiple model replicas concurrently on each GPU, thereby avoiding under-utilisation of GPUs even when the preferred batch size is small. For this, CrossBow must (i) decide on an appropriate number of model replicas per GPU and (ii) employ an efficient and scalable synchronisation scheme within and across GPUs. CrossBow automatically tunes the number of replicas per GPU at runtime to maximise training throughput for a given batch size. We designed a novel synchronisation scheme that eliminates dependencies among model replicas, enabling high throughput and scalability. Our experiments show that CrossBow outperforms TensorFlow on a 4-GPU server by 2.5× with ResNet-32.

1 BATCH SIZES IN DEEP-LEARNING SYSTEMS

Large-scale deep learning models are prominent in many application fields, including computer vision [5, 9, 12], speech recognition [14], and natural language processing [13]. Models are trained using labelled data samples to make accurate predictions. When training with stochastic gradient descent (SGD), model parameters are iteratively refined based on the how well they can predict the label of each sample. A model update, termed a *gradient*, is typically computed over a *batch* of samples. The goal of training is to reach a desired level of accuracy as fast as possible.

Training can effectively exploit the parallelism of a GPU, and multi-GPU servers have become widely available: public cloud providers offer VMs with up to 16 GPUs [2], and a 10-GPU server costs less than $40,000. Deep learning systems such as TensorFlow [1], MXNet [3], CNTK [10], and Caffe [6] must therefore scale well with multiple GPUs in a single server. As the number and capability of GPUs in a single server increases, this becomes increasingly challenging because systems must fully utilise the parallelism of all GPUs during training without introducing bottlenecks.

Current systems [1, 3] must use large batch sizes in order to scale to multiple GPUs. They commonly employ synchronous stochastic gradient descent (S-SGD) [8] to parallelise model training on a multi-GPU server. Assuming a batch size b across k GPUs, parallel training with S-SGD involves two types of computational tasks for *learning* and *synchronisation*: learning tasks on each GPU first compute a partial gradient using b/k samples; synchronisation tasks then update the model with aggregate values before the next batch is considered. To scale S-SGD with the number of GPUs k, the batch size b must increase as well. When b increases, however, eventually statistical efficiency [15] decreases due to the less frequent model updates, resulting in a higher time-to-accuracy.

Systems attempt to compensate for this effect during training: they may proportionally increase the learning rate [4], or they may adapt the batch size during training [11]. These techniques, however, do not avoid the fundamental issue that current systems *couple* the batch size b with the number of GPUs k, preventing the training of models with small batch using many GPUs. *We ask the question how to design for a deep-learning system that supports the efficient training with small batch sizes while scaling to many GPUs.*

2 CROSSBOW: MULTIPLE REPLICAS PER GPU

Our design for a new deep-learning system, CrossBow, exploits the idea to train multiple model replicas concurrently on each GPU, thus fully utilising all GPU resources even with small batch sizes. For this, CrossBow overcomes two challenges: (i) it automatically selects the optimal number of model replicas per GPU (§2.1); and (ii) it frequently synchronises a large number of model replicas within and across GPUs without limiting concurrency (§2.2).

2.1 Selecting the number of model replicas

When training multiple model replicas on a single GPU, the number of replicas must be chosen carefully for a given batch size: when training too few replicas, the GPU is under-utilised, wasting resources; choosing too many means that learning tasks are sequentialised partially on the GPU, leading to a slow-down. In addition, the synchronisation of many replicas reduces statistical efficiency, particularly for large models. Finding a sweet-spot is challenging because it depends on the features of the model (e.g. its parameters), the GPU (e.g. the number of cores), and the software stack used for training (e.g. the specific employed kernels).

CrossBow estimates the best number of replicas per GPU based on the observed training throughput, i.e. the data volume of processed batches per second. CrossBow thus detects under- and over-utilisation of the GPU during training: it observes a decrease in throughput and adapts the number of model replicas accordingly.

CrossBow starts with a single replica per GPU, m=1, and increases m as long as a significant increase in throughput is observed. The increase must be above a given threshold (empirically set to 10%) to compensate for the potential loss in statistical efficiency due to the larger number of replicas. In practice, CrossBow discovers the best number of replicas to use after a few seconds.

2.2 Synchronising model replicas

Deep learning systems such as TensorFlow and MXNet adopt synchronous SGD (S-SGD) to coordinate model replicas. It requires all replicas to synchronise at the same time, creating a global barrier between successive learning tasks. This approach is ill-suited for CrossBow for two reasons: (i) since CrossBow trains multiple replicas per GPU, the overall number of replicas in the system is significantly higher, typically by a factor of 2 or more. This drastically increases the chances of *stragglers*, which would slow down the rest of the replicas due to the global barrier. The effect would be even more pronounced if different generations of GPUs (with different performance characteristics) were used; and (ii) the cost of synchronising *within* a single GPU is significantly lower than *across* GPUs. By requiring a global synchronisation step, S-SGD does not exploit this difference.

To address these shortcomings, we develop a new synchronisation technique that eliminates any global barrier and reduces dependencies among tasks to increase concurrency and, hence, training throughput. Our approach works as follows: each GPU stores a reference copy of the model locally. After a model replica has been updated by all its assigned learning tasks, the CrossBow scheduler issues a synchronisation task that computes the weighted difference between the model replica and the local reference model and applies it to the replica. Once all model replicas across all GPUs have completed their assigned learning tasks, all differences are aggregated and sent to one GPU where a new global model is generated and then copied to all other GPUs. This ensures that all reference models are consistent. As done by *elastic averaging SGD* (EA-SGD) [16], CrossBow uses a moving average to update the values of models, which yields faster convergence.

A key property of our approach is that all replica synchronisation tasks are independent of each other. Once a replica has synchronised with its local reference model, it is ready to accept the next learning task *without waiting for other synchronisation tasks to complete*. Dependencies between successive synchronisation tasks are set on a per replica basis. This fine-grained concurrency control has several benefits: (i) by avoiding a global synchronisation step, it allows CrossBow to scale to a large number of GPUs and model replicas; (ii) it is not impacted by stragglers (e.g. due to heterogeneous hardware). CrossBow's task scheduler maintains a pool of model replicas and assigns learning tasks opportunistically as a new model replica becomes available. This ensures that model replicas executing on a faster GPU receive more tasks; and (iii) by decoupling intra- and inter-device synchronisation, it is possible to optimise performance further by taking advantage of locality. In our current implementation, we first aggregate the differences for all model replicas hosted on the same GPU before propagating them to the GPU responsible for generating the new model, thus reducing GPU-to-GPU transfers.

TensorFlow and Caffe2 [4] support several inter-GPU synchronisation mechanisms to aggregate gradients and broadcast models. While they overlap data movement with gradient computation, as each operation is handled by distinct parts of the GPU runtime (the copy and compute engines), CrossBow enables efficient multiplexing of tasks on the compute engine. DimmWitted [15] explores the problem of model replica synchronisation on multi-core CPUs.

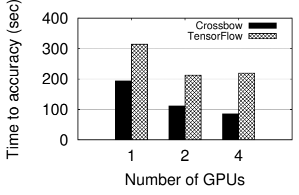

Figure 1: Scalability

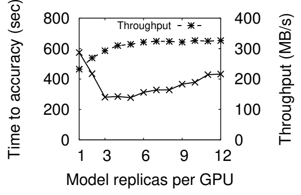

Figure 2: Model replicas

Each NUMA node has its own replica shared by the cores within that node, which can be seen as having multiple replicas per GPU. Within a node, DimmWitted builds an asynchronous update model amongst cores to boost performance, relying on the hardware to ensure data coherence. On GPUs, however, uncontrolled concurrency leads to data corruption. CrossBow's model achieves high utilisation while remaining synchronous.

3 EVALUATION

We evaluate CrossBow's ability to scale on a 4-GPU server, compared to TensorFlow v1.4 [1], and the impact of varying the number of model replicas per GPU. The server has two Intel Xeon E5-2640 v4 2.40 GHz CPUs (20 CPU cores) with 64 GB of RAM and four NVIDIA Titan X (Pascal) GPUs with 12 GB of RAM connected via PCIe 3.0 (×16). CrossBow's implementation consists of 8.9K LOCs in C with a Java dataflow front-end (24.8K LOCs). The GPUs use the NVIDIA driver 375.26 with the CUDA 8.0 runtime; the dataflow operator implementations rely on NVIDIA cuBLAS and cuDNN 5.1.

First, we measure the time-to-accuracy of CrossBow and TensorFlow as we vary the number of GPUs. We use ResNet-32, a residual network with ~120 layers [5], and train it to classify images from the CIFAR-10 dataset [7]. We set the target test accuracy to 80% because, without data augmentation, this is where test accuracy plateaus. We use the same configuration for the hyper-parameters for both systems apart from the batch size b.

The results in Fig. 1 show that (i) CrossBow outperforms TensorFlow (by up to 2.5×) due to its ability to train multiple replicas per GPU and (ii) it exhibits better scalability due to its efficient replica synchronisation (§2.2). With 1 GPU, TensorFlow achieves the shortest time-to-accuracy with b=128; the best batch size for CrossBow is the same, but it trains two model replicas concurrently, thus boosting convergence speed. With 2 GPUs, the best batch size for TensorFlow increases to b=512 (i.e. 256 per GPU); CrossBow scales linearly with b=64 and 3 replicas per GPU. With 4 GPUs, both systems suffer from the synchronisation overhead, but the impact on CrossBow is lower.

Next we explore CrossBow's approach for selecting the number of model replicas per GPU. To fit a large number of replicas, we use ResNet-20 with a target accuracy of 88% trained on 3 GPUs. Fig. 2 shows the time-to-accuracy and the throughput as we increase the number of model replicas per GPU. With more model replicas, eventually the GPU resources are saturated and the throughput plateaus. This coincides with values of m that yield the shortest time-to-accuracy, which is what CrossBow exploits for tuning m. This approach works because, while EA-SGD permits model replicas to diverge, CrossBow schedules synchronisation tasks after a model replica was updated a few times. We have also experimented with other ResNet networks with similar results.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Berkeley, CA, USA, 265–283.

[2] Amazon EC2 Instance Types (P2). (2017). https://aws.amazon.com/ec2/instance-types/ Last accessed on January 6, 2017.

[3] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *CoRR* abs/1512.01274 (2015). arXiv:1512.01274 http://arxiv.org/abs/1512.01274

[4] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR* abs/1706.02677 (2017). arXiv:1706.02677

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385

[6] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the ACM International Conference on Multimedia, MM '14, Orlando, FL, USA, November 03 - 07, 2014*, Kien A. Hua, Yong Rui, Ralf Steinmetz, Alan Hanjalic, Apostol Natsev, and Wenwu Zhu (Eds.). ACM, 675–678. https://doi.org/10.1145/2647868.2654889

[7] Alex Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images*. Master's thesis. Department of Computing Science, University of Toronto. www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf.

[8] Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. *CoRR* abs/1404.5997 (2014). arXiv:1404.5997 http://arxiv.org/abs/1404.5997

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105.

[10] Frank Seide and Amit Agarwal. 2016. CNTK: Microsoft's Open-Source Deep-Learning Toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 2135. https://doi.org/10.1145/2939672.2945397

[11] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. 2017. Don't Decay the Learning Rate, Increase the Batch Size. *CoRR* abs/1711.00489 (2017). arXiv:1711.00489

[12] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. *CoRR* abs/1409.4842 (2014).

[13] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). arXiv:1609.08144

[14] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. 2016. The Microsoft 2016 Conversational Speech Recognition System. *CoRR* abs/1609.03528 (2016). arXiv:1609.03528

[15] Ce Zhang and Christopher Ré. 2014. DimmWitted: A Study of Main-memory Statistical Analytics. *Proc. VLDB Endow.* 7, 12 (Aug. 2014), 1283–1294. https://doi.org/10.14778/2732977.2733001

[16] Sixin Zhang, Anna Choromanska, and Yann LeCun. 2015. Deep Learning with Elastic Averaging SGD. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15)*. MIT Press, Cambridge, MA, USA, 685–693.