Dike: Revisiting Resource Management for Distributed Deep Learning

Erci Xu
Ohio State University
xu.1556@osu.edu

Mohit Saxena
saxena.mohit@gmail.com

Feng Qin
Ohio State University
qin.34@osu.edu

ABSTRACT

The recent adoption of deep learning for diverse applications has required scaling infrastructures both horizontally and vertically. As a result, efficient resource management for distributed deep learning (DDL) frameworks is becoming increasingly important. However, existing techniques for scaling DDL applications rely on general-purpose resource managers originally designed for data intensive applications. In contrast, DDL applications present unique challenges for resource management as compared to traditional big data frameworks, such as a different master-slave communication paradigm, deeper ML models that are more computationally and network bound than I/O, and use of heterogeneous resources (GPUs, TPUs, and variable memory). In addition, most DDL frameworks require data scientists to manually configure the task placement and resource assignment to execute DDL models.

In this paper, we present Dike, an application scheduler framework that transparently makes scheduling decisions for placement and resource assignment to DDL workers and parameter servers, based on the unique characteristics of the DDL model (number and type of parameters and neural network layers), node heterogeneity (CPU/GPU ratios), and input dataset. We have implemented Dike as a resource manager for DDL jobs in Tensorflow on top of Apache Mesos. We show that Dike significantly outperforms both manual and static assignment of resource offers to Tensorflow tasks, and achieves at least 95% of the optimal throughput for different DDL models such as ResNet, Inception.

INTRODUCTION

Today, distributed deep learning (DDL) is the widely used in different areas ranging from image classification to speech recognition [11, 12]. Various open source frameworks for DDL such as Tensorflow, MXnet, Azure Machine Learning [1, 2] are being offered as services by cloud providers or deployed by users in private clusters using resource containers. As a result, efficient resource management [8, 10] for DDL frameworks is critically important.

Resource management in major DDL frameworks is still evolving and does not account for the unique characteristics of the machine learning jobs. For example, the data scientist has to address the following four questions while deploying the DDL model: (1) How many DDL tasks need to be launched? (2) How much resource allocated for each task? (3) What is the role or functionality of each task? (4) Which physical node to use for launching each task? As these questions are specific to the requirements of the DDL model and have multiple possible answers, users need to iteratively try out different deployment plans, which requires considerable manual tuning. Moreover, even if a good solution is obtained, it may be no longer suitable if users change the cluster configurations, use different models, or train on another dataset.

Previously, answering similar questions in other data-intensive analytic frameworks [4] is often straightforward due to a clear guideline of mitigating bottleneck [9]. As a result, most big data frameworks provide resource assignment by allocating sufficient memory to each task and reduce frequent disk I/O (addressing questions - 1 and 2). For task placement, locality has been the foremost priority to minimize network transfers (addressing question - 3/4).

Unfortunately, simply applying these principles to resource management for DDL frameworks would not always lead to a good solution. This is attributed to three unique features of DDL: deeper ML model pipelines that are both more computationally and network-bound than disk-bound, master-slave communication paradigm between parameter-servers and workers, and the use of heterogenous resources for DDL such as GPUs and variable memory.

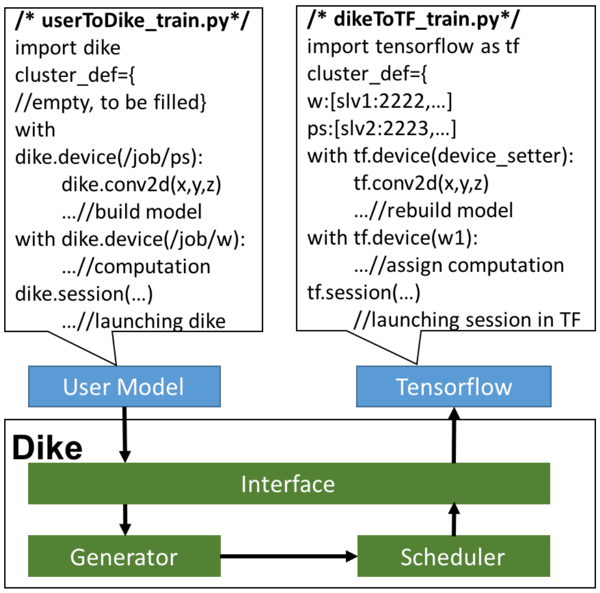
First, most DDL models use a computation-intensive procedure called gradient descent. Iteratively computing to minimize the loss function in gradient descent is the key bottleneck rather than disk I/O[3]. Similarly, the DL pipelines usually comprise of several layers of highly computationally-intensive operations and backward propagation of parameters. As a result, allocating resources to tasks in DDL based on memory requirements alone may lead to allocate more than enough memory for workers and subsequently waste of computing resources.

Second, DDL introduces a new role for classic distributed master-slave communication - parameter server (PS) [7]. This affects the task placement as a slave node can either serve as a worker executing the DL models or a PS that maintains a synchronized view of the DL model parameters across different workers. Accordingly, resources assigned to a PS have very different requirements than that of a worker. Similarly, a PS needs to be placed in a location so as to minimize the network overhead for synchronizing the parameters with the workers for which it is responsible.

Third, DDL clusters are usually equipped with very heterogenous hardware such as differnet compute power - GPU, TPU, and different provisioned memory [1]. As a result, selecting the right compute and memory offers to provision for the different DL model operators becomes critical. For example, GPUs offer high parallelism and concurrency levels well suited for computation on workers. Most data analytic frameworks run on homogeneous configurations. However, simply binding the GPU nodes to workers and CPUs to PS, may not result in the optimal placement, as different workers need to be co-located with different PS.

In existing systems [1, 2], most of the resource assignment and task placement is still done by the data scientist by selecting the physical nodes or resource containers for executing different workers and PS. For example, Tensorflow requires the data scientists writing the DL model to determine the total number of workers and PS, location of PS/worker tasks, and their CPU/GPU and memory configurations. In contrast, resource management frameworks based on Apache Mesos [5, 6] use coarse-grained levels for resource assignment and scheduling strategies not aware of the DL model characteristics. In addition, data scientists still need to determine the total number of tasks and resources for each of them. As a result, we find that their task assignment is not optimal in most cases.

In this paper, we present Dike, an online scheduler framework for DDL as shown in Figure 1. Data scientists only need to write their DL models, which are similar to the Tensorflow models, however, without requiring any resource management details. Dike creates a new wrapper *Context* over Tensorflow context to intercept and capture runtime information, including model details and cluster configuration, from the DDL framework. Dike then passes this information, as shown in Figure 2 to its *Generator* component to produce multiple candidate resource assignment plans. These plans are finally evaluated by Dike's *Scheduler* based on a cost model, which decides the best plan and its placement for each task. We show that Dike achieves at least 95% of the optimal performance for distributed DDL workloads and automates most of the cluster resource management. In addition, Dike provides 81% speed-up as compared to manual configuration employed by data scientists on a 30-instance cluster.

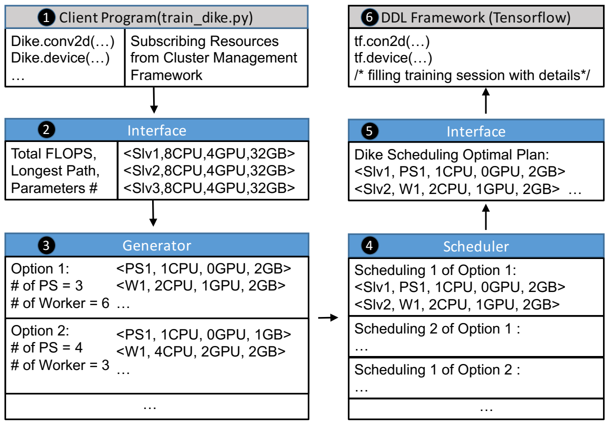


```
/* userToDike_train.py*/
import dike
cluster_def={
//empty, to be filled}
with
dike.device(/job/ps):
        dike.conv2d(x,y,z)
        ...//build model
with dike.device(/job/w):
        ...//computation
dike.session(...)
        ...//launching dike
```

```
/* dikeToTF_train.py*/
import tensorflow as tf
cluster_def={
w:[slv1:2222,...]
ps:[slv2:2223,...]
with tf.device(device_setter):
        tf.conv2d(x,y,z)
        ...//rebuild model
with tf.device(w1):
        ...//assign computation
tf.session(...)
        //launching session in TF
```

Figure 1: Architecture of Dike.

Future work of Dike includes: supporting various DDL frameworks, such as MxNet and CNTK, adapting to more types of neural networks models, and optimizing resource management algorithms for large scale cluster. We aim at building Dike as a platform where it supports various DDL framework and connects other backends.



1 Client Program(train_dike.py)
Dike.conv2d(...)
Dike.device(...)
...
Subscribing Resources from Cluster Management Framework
2 Interface
Total FLOPS, Longest Path, Parameters #
<Slv1,8CPU,4GPU,32GB>
<Slv2,8CPU,4GPU,32GB>
<Slv3,8CPU,4GPU,32GB>
3 Generator
Option 1:
# of PS = 3
# of Worker = 6
<PS1, 1CPU, 0GPU, 2GB>
<W1, 2CPU, 1GPU, 2GB>
...
Option 2:
# of PS = 4
# of Worker = 3
<PS1, 1CPU, 0GPU, 1GB>
<W1, 4CPU, 2GPU, 2GB>
...
...
6 DDL Framework (Tensorflow)
tf.con2d(...)
tf.device(...)
/* filling training session with details*/
5 Interface
Dike Scheduling Optimal Plan:
<Slv1, PS1, 1CPU, 0GPU, 2GB>
<Slv2, W1, 2CPU, 1GPU, 2GB> ...
4 Scheduler
Scheduling 1 of Option 1:
<Slv1, PS1, 1CPU, 0GPU, 2GB>
<Slv2, W1, 2CPU, 1GPU, 2GB>
Scheduling 2 of Option 1 :
...
Scheduling 1 of Option 2 :
...
...

Figure 2: Detailed Process of Dike

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 *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, GA, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[2] 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). http://arxiv.org/abs/1512.01274

[3] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. 2013. Deep learning with COTS HPC systems. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, Sanjoy Dasgupta and David Mcallester (Eds.), Vol. 28. JMLR Workshop and Conference Proceedings, 1337–1345. http://jmlr.org/proceedings/papers/v28/coates13.pdf

[4] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[5] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11)*. USENIX Association, Berkeley, CA, USA, 295–308. http://dl.acm.org/citation.cfm?id=1972457.1972488

[6] Douban Inc. 2017. TFMesos. https://github.com/douban/tfmesos. (2017).

[7] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 583–598. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu

[8] Dirk Merkel. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 2014, 239, Article 2 (March 2014). http://dl.acm.org/citation.cfm?id=2600239.2600241

[9] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making Sense of Performance in Data Analytics Frameworks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 293–307. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/ousterhout

[10] David K. Rensin. 2015. *Kubernetes - Scheduling the Future at Cloud Scale.* 1005 Gravenstein Highway North Sebastopol, CA 95472. All pages. http://www.oreilly.com/webops-perf/free/kubernetes.csp

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. *CoRR* abs/1409.3215 (2014). http://arxiv.org/abs/1409.3215

[12] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the Inception Architecture for Computer Vision. *CoRR* abs/1512.00567 (2015). http://arxiv.org/abs/1512.00567