DOI: 10.1051/itmconf/20171203030

DLTAP: A Network-efficient Scheduling Method for Distributed Deep Learning Workload in Containerized Cluster Environment

Wei QIAO[1,a], Ying LI[2,b] and Zhong-Hai WU[3,*]

[1]School of Software and Microelectronics, Peking University, Beijing, China
[2]School of Software and Microelectronics, Peking University, Beijing, China
[3]School of Software and Microelectronics, Peking University, Beijing, China
[a]qiaowei@pku.edu.cn, [b]li.ying@pku.edu.cn
**Corresponding author: wuzh@ss.pku.edu.cn*

Abstract: Deep neural networks (DNNs) have recently yielded strong results on a range of applications. Training these DNNs using a cluster of commodity machines is a promising approach since training is time consuming and compute-intensive. Furthermore, putting DNN tasks into containers of clusters would enable broader and easier deployment of DNN-based algorithms. Toward this end, this paper addresses the problem of scheduling DNN tasks in the containerized cluster environment. Efficiently scheduling data-parallel computation jobs like DNN over containerized clusters is critical for job performance, system throughput, and resource utilization. It becomes even more challenging with the complex workloads. We propose a scheduling method called Deep Learning Task Allocation Priority (DLTAP) which performs scheduling decisions in a distributed manner, and each of scheduling decisions takes aggregation degree of parameter sever task and worker task into account, in particularly, to reduce cross-node network transmission traffic and, correspondingly, decrease the DNN training time. We evaluate the DLTAP scheduling method using a state-of-the-art distributed DNN training framework on 3 benchmarks. The results show that the proposed method can averagely reduce 12% cross-node network traffic, and decrease the DNN training time even with the cluster of low-end servers.

1 Introduction

Large-scale deep learning, for instance, Deep Neutral Network (DNN), has driven advances with higher accuracy than traditional techniques in many different fields especially at image classification [1, 2], speech recognition [3] and text processing [4]. To train DNNs on a large-scale, researchers have exploited distributed deep learning systems [1, 5, 6, 7], such as DistBelief [1] and Tensorflow [7]. Compared to DistBelief, Tensorflow is more flexible, faster and easy to use. It introduces a new distributed programming paradigm that a DNN job is divided into a number of parameter server tasks (ps tasks) and worker tasks, which can be flexibly allocated in the cluster. It is different from previous parameter server concept [5] that there is no need for a centralized global parameter server to store all the parameters of a job. This flexible training approach brings some benefits, but on the other hand introduces new challenges: (a) how to provision resources for tasks in a timely and elastic way; and (b) how to schedule ps and worker tasks on the distributed environment to improve job performance.

With the cloud computing revolution brought by Docker [8], the advantages of containers in improving resource utilization and production efficiency have attracted much attention. The development of containerization technology has greatly accelerated the popularization and application of deep learning. To put deep learning task into container and make the container as the unit of management can enable broader and easier deployment of deep learning algorithms for building cloud-based distributed deep learning platform [9]. In addition, it enables higher levels of utilization and can transition from training to serving smoothly. However, container isolation brings a certain degree of performance loss, mainly due to the cost of network communication when updating the gradient between the ps task and the worker task.

In this paper, the issue of non-optimal network transmission for distributed deep learning workload in containerized cluster environment is put forward and solved by our network-efficient scheduling method called DLTAP. The main objective is to take aggregation degree of ps task and worker task into consideration, and the comparison demonstrates that it can greatly reduce the cross-node network transmission traffic caused by ps task and worker task. More importantly, in this way, we can also accelerate the training speed of DNNs.

The remainder of this paper is organized as follows. Section 2 describes the related works on container scheduler. Section 3 presents in detail the statement of problem and the proposed method. Section 4 presents the experimental results. Concluding remarks are covered in the last section.

2 Related Works

Container schedulers are aimed at scheduling a container to a most appropriate node. Docker Swarm [10] is simple to use due to the same usage with original Docker API engine, but it does not provide features to handle node failures which makes it not recommendable in a production environment. Mesos [11] is a kind of two-level scheduler [12] which uses a single active resource manager to offer compute resources to multiple parallel "scheduler frameworks". Choosing Mesos with Marathon [13] is an excellent combination if you already have a Mesos cluster. Kubernetes [14] is the third container-management system for automating deployment, scaling, and management of containerized applications influenced by Borg [9]. It aims to provide better ways of managing related, distributed components across varied infrastructure, while still benefiting from the improved utilization containers enable. Aurora [15] is a Borg-like scheduler that runs on top of Mesos, but it works for long running services. Tupperware [16] is a Borg-like system and it increases efficiency by reducing time spent on getting app to run in production. But only a few details have been disclosed.

However, all these schedulers treat each container as a "black box" and do not take into account characteristics of task running in the container that results in performance loss. In this paper, we differentiate containers by their running DNN tasks, i.e., ps task and worker task, and considering their network transmission traffic during scheduling.

3 Proposed Method

Problem Statement. We assume a relatively uncomplicated case, the cluster consists of four physical machines, including one as the master, remaining three are slaves, respectively recorded as s1, s2 and s3. We use a DNN based on standard data set MNIST [17]. The network structure is a modified version of LeNet5 [18] with 3 million parameters. In the case, the job is divided into two ps tasks, recorded as ps1, ps2, and three worker tasks, recorded as wk1, wk2, wk3. All parameters of our model are evenly distributed among ps tasks. Each worker task will send the appropriate gradient updates to the corresponding ps1 and ps2 after each mini-batch of input data trained, and receive the latest parameters they send back.

Considering two different scheduling scenarios. We assume that each host is load balanced due to other workloads. The first is that wk1 and wk3 are on s1, ps1 and ps2 are on s2, wk2 is on s3, as is shown in Fig. 1(a). In this scenario, a total of 6 cross-node network transmissions occur after each mini-batch of input data trained. The second scenario is that wk1, wk3 and ps1 are on s1, ps2 is on s2, wk2 is on s3, as is shown in Fig. 1(b). So a total of 4 cross-node and 2 inner-node network transmissions are generated after each mini-batch of input data trained. A significant difference is that the first scenario is 2 more times than the second scenario of cross-node network transmission after each mini-batch of input data trained. After massive experimental tests, we find that the network throughput decreased 12% and the training time reduced 15% in the second scenario compared with those in the first scenario. In Section 4 there will be a detailed comparative analysis. Here we call the first scheduling situation "the non-ideal" scheduling scenario, and the second scheduling situation the "ideal" scheduling scenario.

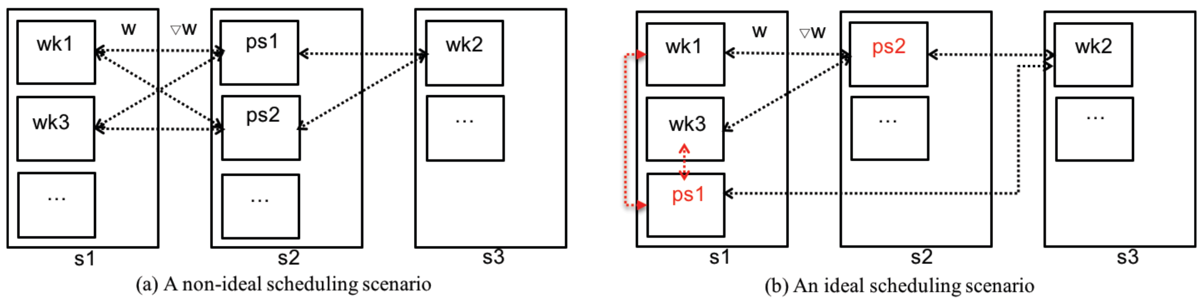

wk1
wk3
...
w
▽w
ps1
ps2
...
wk2
...
s1
s2
s3
(a) A non-ideal scheduling scenario
wk1
wk3
ps1
w
▽w
ps2
...
wk2
...
s1
s2
s3
(b) An ideal scheduling scenario

Figure 1. The two scheduling scenarios.

Proposed Method. The default Kubernetes scheduling method includes two steps. The first step is to filter out the nodes that do not meet certain requirements and the second step is ranking the remaining nodes to find a best fit for the pod. A pod is a group of one or more containers in Kubernetes. In our design, a pod has only one Docker container, and each ps task or worker task is put into a pod. In the ranking process, different priority methods take different dimensions into account. Each priority method is weighted by a positive number and the final score of each node is calculated by adding up all the weighted scores.

Despite the fact that there is a certain probability to produce non-ideal scheduling scenario by using the default scheduling method, we want to minimize this probability. To this end, we propose a new scheduling method DLTAP. The core idea is to improve the aggregation degree of ps tasks and worker tasks in one node. The DLTAP scheduling method is shown in Fig. 2.

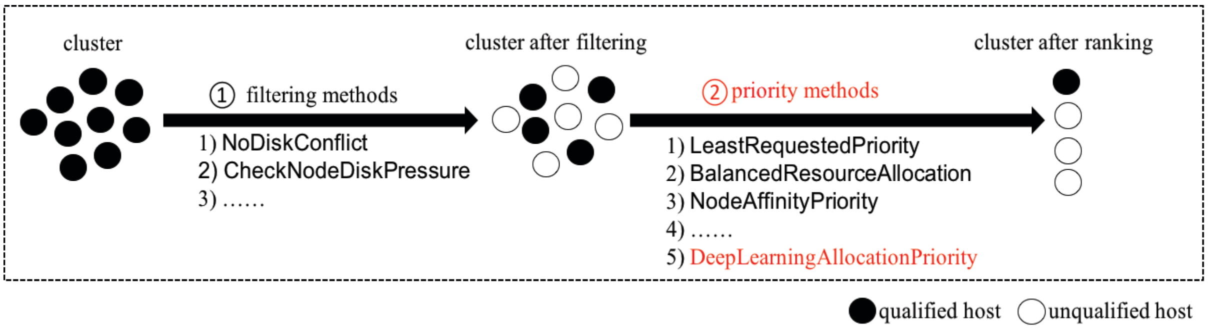


Figure 2. The DLTAP scheduling method.

Our method belongs to the ranking process and is divided into two steps. It is assumed that our training job consists of several ps tasks and worker tasks. Firstly, distribute each worker task according to the initial hybrid scoring model by Eq. 1. Secondly, distribute each ps task according to the new hybrid scoring model by Eq. 2. In the second step, the method calculates the current ps task aggregation degree for each remaining node by Eq. 3, and then, the result is added to the initial hybrid scoring model. Algorithm 1. outlines the ps task distributing process of DLTAP scheduling method.

$$finalScoreNodeA = \sum_{i=1}^{N}(weight_i \times defaultPriorityMethodScore_i). \quad (1)$$

$$finalScoreNodeA' = finalScoreNodeA + weightOfDLTAP \times priorityScoreOfDLTAP. \quad (2)$$

$$priorityScoreOfDLTAP = \frac{count(current\ job's\ worker\ tasks\ in\ current\ node)}{total\ number\ of\ current\ job's\ worker\ tasks)}. \quad (3)$$

After the scores of all nodes are calculated, the node with highest score is chosen as the host of the ps task. If there are more than one nodes with equal highest scores, a random one among them is chosen. The ps task distributing process completes until all of them are scheduled to the appropriate node.

Algorithm 1. The ps task distributing process of DLTAP.

Definition: *container* indicates the container waiting to be scheduled and there is a ps task in it.
nodes indicate all normal working hosts.
nodeNameToInfo indicates node level aggregated information.
hostPriorityList indicates the priority of scheduling to a particular host, higher priority is better.
hostPriorityList = *{host₁: score₁, host₂: score₂, ..., hostₙ, scoreₙ}*.
weightOfDLTAP indicates the weight of our method in the new hybrid scoring model.
Input: *container*, *nodes, nodeNameToInfo*
Output: *hostPriorityList*

```
1.  Initialize hostPriorityList.
2.  Get current running job id from container.
3.  for node in nodes:
4.      currentNodeWorkerTaskNumbers = 0
5.      for innerContainer in nodeNameToInfo[node.Name].getContainers():
6.          if innerContainer belongs to current job and innerContainer is a worker task:
7.              currentNodeWorkerTaskNumbers++.
8.      score = currentNodeWorkerTaskNumbers / the total number of job's worker tasks.
9.      hostPriorityList.add(node, score * weightOfDLTAP)
10. Return hostPriorityList.
```

4 Experiments Evaluation

To measure the DLTAP's performance, we have built a deep learning prototype platform based on open-sourced Tensorflow and Kubernetes, which has one master and 3 slaves with 2 GB of memory, 2 cores and 60 GB of storage. A common gigabit Ethernet switch connects each node. All slave nodes in the cluster are load balanced.

We performed our evaluations with three different models, including fully connected neural networks (FCNs) [19], convolutional neural networks (CNNs) [2] and recurrent neural networks (RNNs) [20] with Long Short-Term Memory (LSTM) [21]. For FCNs, a 4-layer, a 6-layer, and an 8-layer neural network (FCN-4, FCN-6, FCN-8) are constructed. For CNNs, we choose the classical LeNet5 and make some changes. A small size of LeNet5 (LeNet5-SS) with 0.3 million parameters and a big size of LeNet5 (LeNet5-BS) with 3 million parameters are constructed. For RNNs, we select 2 model with different size, with input length of 16 (LSTM-16) and 32 (LSTM-32) respectively. The detailed neural networks configuration can be found in Table 1.

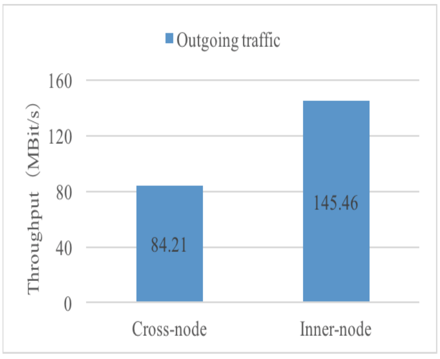
Table 1. The experimental setup of neural networks.

Model	Network	Layer	Parameters [million]
FCN	FCN-4	4	0.03
	FCN-6	6	0.3
	FCN-8	8	3
CNN	LeNet5-SS	6	0.3
	LeNet5-BS	6	3
RNN	LSTM-16	2	0.3
	LSTM-32	2	3

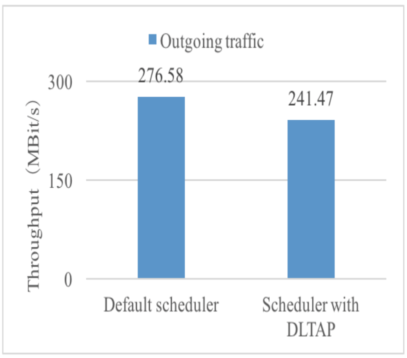
Performance on network throughput. Firstly, we use a FCN-8 job divided into a ps task and a worker task to test. We compare network throughput in the cross-node situation which is that ps task and worker task are in different node, and the inner-node situation which is that ps task and worker task are in same node. We can see from Fig. 3(a) that network throughput of the cross-node is less than the inner-node since inner-node transmission need not to pass the actual physical NIC.

Secondly, we train a FCN-8 job in the ideal scheduling scenario (defined above), we calculate the cross-node network throughput of outgoing traffic for each node associated with current job training. Then the value of each node are added up respectively. This job runs many times, and we take the average of multiple results as the cluster network throughput generated by the job in this scenario. The same experiments are also tested in the scenario of non-ideal scenario (defined above). The results are shown in Fig. 3(b). The results show that the cluster network throughput under the ideal scheduling scenario is less than that under the non-ideal scheduling scenario. Because our method reduces the network transmission of the parameters across the nodes.

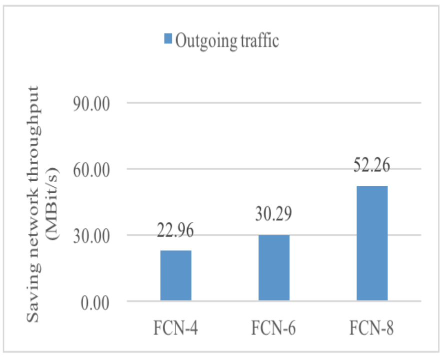
Finally, we use FCN-4, FCN-6, FCN-8 to run same tests respectively under two different scheduling scenarios. The results are shown in Fig. 3(c). We observe that the larger the model the more obvious the saving cluster network throughput. The method leads to a 12.11%, 14.39% and 18.89% decrease in cluster saving network traffic for FCN-4, FCN-6 and FCN-8 respectively.



(a) Cross-node and inner-node network throughput comparison.



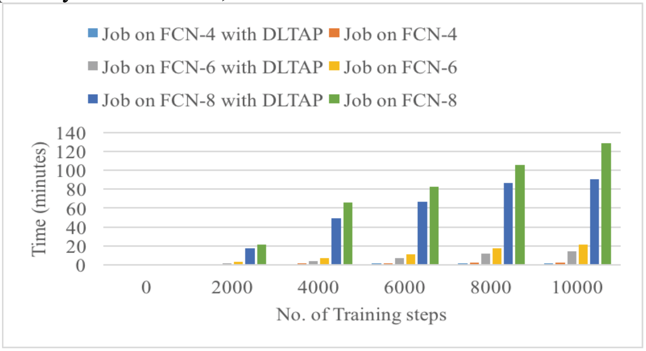
(b) Cluster cross-node network throughput comparison.



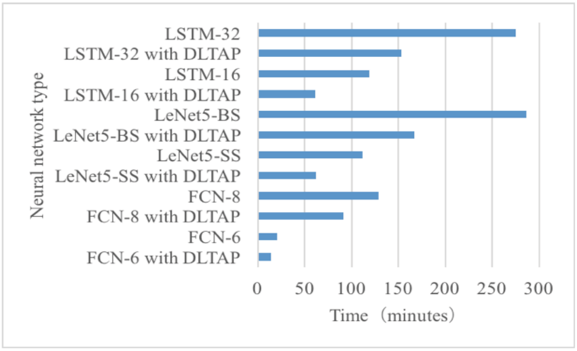
(c) Saving network throughput comparison.

Figure 3. Network throughput comparison.

Performance on training time. We use FCN-4, FCN-6, FCN-8 respectively to run multiple times in each of two scheduling scenarios. Fig. 4(a) illustrates the training time of different job instances when training steps vary from 0 to 10,000 with an increment of 2000. The results prove that the improved scheduler reduces the training time of the job. The reason is that worker tasks can update the gradients and get new parameters faster after each mini-batch of input data trained.



(a) Training time comparison on FCN-4,6,8.



(b) Training time comparison on different models.

Figure 4. Training time comparison.

Performance on different network model type. We use FCN-4, FCN-6, LeNet5-SS, LeNet5-BS, LSTM-16 and LSTM-32 respectively to test the sensitivity of different types of models to the DLTAP. Fig. 4(b) shows that, in the same training steps, the reduction percentage in training time of LSTM-32, LeNet5-BS and FCN-8 is nearly the same. But the training time of LSTM-32 and LeNet5-BS are far greater than the FCN-8's because their worker tasks need more computation to converge.

Summary

Our work points out a network transmission issue for deep learning workload in the containerized cluster environment. We propose DLTAP to alleviate this issue, in particularly, to decrease cross-node network transmission about parameters by improving the degree of aggregation for ps task and worker task, and confirm its effectiveness and performance by massive experiments. The main contribution is that we consider the types of task running in the container as a crucial scheduling factor. Experiments using a state-of-the-art distributed DNN framework demonstrate that our method can averagely reduce 12% cross-node network traffic, and decrease the DNN training time even on a cluster of 4 low-end servers. Moreover, we find that the larger the model the more significantly it effects. In light of these results, we believe that DLTAP can be used as a significant part of scheduling methods for distributed deep learning workload in containerized cluster environment.

In the future, as worker tasks are inherent compute-intensive, we will investigate and further improve the performance of scheduler by considering the factor of GPUs when perform scheduling decisions.

References

1. 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. In NIPS, 2012.
2. A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In NIPS, 2012.
3. G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Transactions on Audio, Speech, and Language Processing, 20(1), 2012.
4. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. The Journal of Machine Learning Research, 12:2493–2537, Nov. 2011.
5. Li M, Zhou L, Yang Z, et al. Parameter server for distributed machine learning[C]//Big Learning NIPS Workshop. 2013, 6: 2.
6. Chilimbi T, Suzue Y, Apacible J, et al. Project adam: Building an efficient and scalable deep learning training system[C]//11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014: 571-582.
7. Abadi M, Agarwal A, Barham P, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015[J]. Software available from tensorflow. org, 2015, 1.
8. Docker Project on https://www.docker.io/, 2014.
9. Verma A, Pedrosa L, Korupolu M, et al. Large-scale cluster management at Google with Borg[C]//Proceedings of the Tenth European Conference on Computer Systems. ACM, 2015: 18.
10. Docker Inc. Docker Swarm. 2015. url: https://github.com/docker/swarm (visited on 01/15/2016).
11. Hindman B, Konwinski A, Zaharia M, et al. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center[C]//NSDI. 2011, 11: 22-22.
12. Schwarzkopf M, Konwinski A, Abd-El-Malek M, et al. Omega: flexible, scalable schedulers for large compute clusters[C]//Proceedings of the 8th ACM European Conference on Computer Systems. ACM, 2013: 351-364.
13. Marathon on https://mesosphere.github.io/marathon/
14. Kubernetes on http://kubernetes.io, Aug. 2014.
15. Apache Aurora on http://aurora.incubator.apache.org/, 2014.
16. A. Narayanan. Tupperware: containerized deployment at Facebook. http://www.slideshare.net/dotCloud/tupperware-containerized-deployment-at-facebook, June 2014.
17. MNIST on http://yann.lecun.com/exdb/mnist/
18. LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. 1998. Gradient-based learning applied to document recognition. In Proceedings of the IEEE, 86(11):2278–2324, (Nov. 1998).
19. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to hand- written zip code recognition," Neural computation, vol. 1, no. 4, pp. 541–551, 1989.
20. W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," arXiv preprint arXiv:1409.2329, 2014.
21. S.HochreiterandJ. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.