CS744 Assignment-2 Report

Mohammed Danish Shaikh, Mohan Rao Divate Kodandarama, Shreeshrita Patnaik
{mshaikh4, divatekodand, spatnaik2}@wisc.edu
University of Wisconsin, Madison

1 Goals

1. To write simple Tensorflow [1] applications and make them run in a cluster.
2. To get a deeper understanding about Tensorflow's performance.

2 Experimental Setup

All experiments were performed on CloudLab [2] in a cluster containing three c220g5 nodes interconnected to each other. The configuration of each node is as listed in Table 1.

CPU	Two Intel Xeon Silver 4114 10-core CPUs at 2.20 GHz
RAM	192GB ECC DDR4-2666 Memory
Disk	One 1 TB 7200 RPM 6G SAS HDs
Disk	One Intel DC S3500 480 GB 6G SATA SSD
NIC	Dual-port Intel X520-DA2 10Gb NIC (PCIe v3.0, 8 lanes)
NIC	Onboard Intel i350 1Gb

Table 1: Cluster node configuration

These three nodes hereafter are referred to as N1, N2 and N3. Experiments were performed on Tensorflow cluster settings specified in Table 2.

M	**NWT**	**NPT**	**WTL**	**PTL**
Single	1	0	N1	-
Cluster	2	1	N1,N2	N1
Cluster2	3	1	N1,N2,N3	N1

Table 2: Tensorflow cluster settings

Where
M - Deployment mode
NWT - Number of worker tasks
NPT - Number of parameter tasks
WTL - Worker tasks location
PTL - Parameter tasks location

3 Logistic Regression

Logistic Regression is a discriminative algorithm for classification. The pseudo-code of the algorithm for k class classification is described in Algorithm 1.

3.1 Algorithm

Algorithm 1 Logistic Regression Algorithm

```
Require: Minimal misclassifications
Input: Training set - $S = \{(X, y) | X \in R^{m \times k}, y \in \{0, 1, ..., k\}\}$
Output: $W \in R^{m \times k}, b \in R^k \ni$ the cross entropy loss between y and $W^T x + b$ is minimized across all training examples.
  Initialization: $W, b \sim N(0, 1)$, Initialize parameter $W, b$ with random values drawn from standard normal distribution.
  while $step \leq NUM\_STEPS$ do
    For each each batch:
      *Forward Pass through the network and compute outputs as well as loss.
      *Compute gradients of loss with respect to all the parameters using    backpropagation algorithm
      *Upated the parameters as $W^{t+1} = W^t - \eta \partial W$
  end while
```

3.2 Dataset

We've used the MNIST [4] dataset(Modified National Institute of Standards and Technology database), which is a large database of handwritten digits. The database contains 60,000 training images and 10,000 testing images. Images are grayscale with dimensions of 28×28 each.

3.3 Synchronous SGD

In Synchronous SGD, the parameter servers wait for all workers to send their gradients, aggregate them, and send the updated parameters to all workers afterward, making sure that the actual algorithm is a true mini-batch stochastic gradient descent, where the actual batch size is the sum of all the mini-batch sizes of the workers.

3.4 Asynchronous SGD

In Asynchronous SGD, each worker processes a mini-batch of data and updates the parameters independently of the other ones, as follows:

1. It fetches from the parameter servers the most up-to-date parameters of the model needed to process the current mini-batch.

2. It then computes gradients of the loss with respect to these parameters.

3. Finally, these gradients are sent back to the parameter servers, which then updates the model accordingly.

3.5 Evaluation

We aim to evaluate the following:

1. Visualize the graphs produced by Tensorboard [6] for single mode of execution. (Section 3.5.1)

2. Performance characteristics of Synchronous SGD and Asynchronous SGD across deployment modes mentioned in Table 2. (Section 3.5.2 and 3.5.3)

3. Performance characteristics of Asynchronous SGD in comparison to Synchronous SGD. (Section 3.5.4)

4. Performance characteristics of Asynchronous and Synchronous SGD on varying batch sizes. (Section 3.5.5)

Unless otherwise stated, our evaluations should be interpreted with the following points in mind:

1. We have trained each worker in all our experiments for 6000 global steps.

2. CPU, Network and Memory statistics presented are the maximum of their corresponding occurrences during the experiments. Furthermore, these have only been reported for the node containing the parameter server job in the experiments.

3. Execution time of multiple workers is averaged in cluster experiments. We didn't find this to be a problem since all workers were taking around the same amount of time.

4. Our cluster experiments have one of the workers colocated with the parameter server job. This implies that the *loopback* and *external* networking interfaces would be utilized for the communication, requiring data to be collected across both components for analysis. We didn't find anything interesting in the loopback interface, hence we only include analysis for the external interface.

3.5.1 Single Mode

We implemented the Logistic Regression application and trained it using the single node mode in order to get better understanding of Tensorflow [8]. We visualized various summaries using Tensorboard, we present a select few of them here:

1. Computational graph constructed by Tensorflow (Figure 1).

2. Improvement of accuracy with the number of training epochs (Figure 2).

3. Decrease in loss with the number of training epochs (Figure 3).

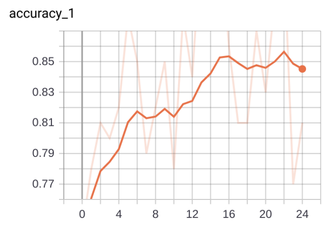

Figure 2: Accuracy vs Epoch

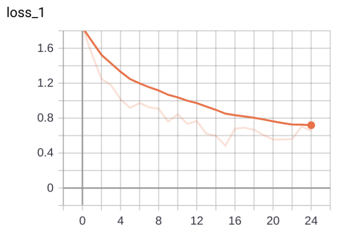

Figure 3: Loss vs Epoch

3.5.2 Asynchronous SGD

As shown in Table 3, increase in number of workers by one results in more workers asynchronously updating global steps, thereby improving average worker execution time.

Mode	Memory (MB)	Network (rxMB/s / txMB/s)	CPU Load	Execution Time (s)
Cluster	811	7.9/8.4	11	12.7
Cluster2	805	7.7/8.5	11	10

Table 3: Asynchronous SGD statistics

3.5.3 Synchronous SGD

As shown in Table 4, increase in number of workers by one resulted in a decrease in average worker execution time and increase in network data being received and transmitted by the parameter server. We attribute this to more coordination being required by the parameter server before aggregating the gradients in order to synchronize all the workers.

Mode	Memory (MB)	Network (rxMB/s / txMB/s)	CPU Load	Execution Time (s)
Cluster	2374	136/136	5	312
Cluster2	2241	155/155	5.28	282

Table 4: Synchronous SGD statistics

3.5.4 Asynchronous SGD vs Synchronous SGD

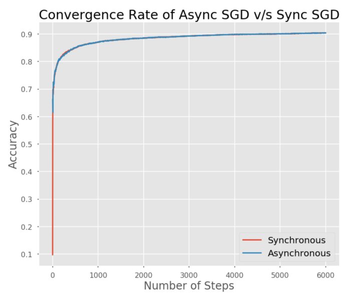

Figure 4

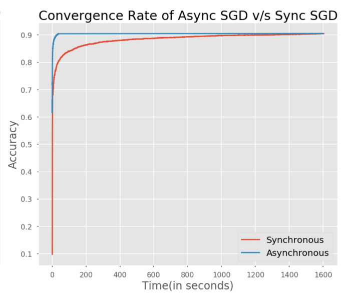

Figure 5

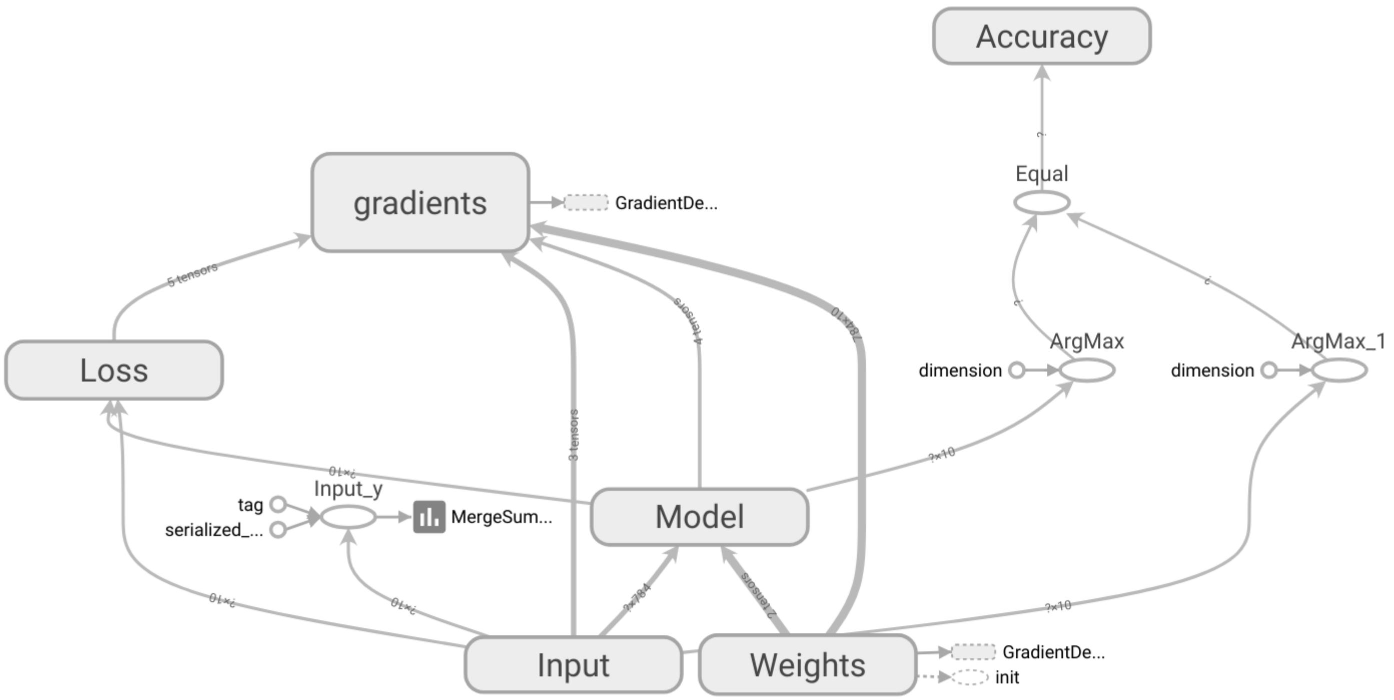

Figure 1: Computational Graph

As shown in the figure 5, Asynchronous SGD converges much faster than Synchronous SGD. Synchronous SGD takes about 1600 seconds to reach an accuracy of 90%, while Asynchronous SGD takes only 37 seconds to reach the same accuracy. However, the total number of steps taken by each algorithm is approximately the same as shown in 4. This difference in convergence rate is expected since, the parameter server in the Synchronous SGD has to wait until it has accumulated gradients from all the workers, essentially causing a barrier.

Resource utilization and performance metrics for both Synchronous SGD and Asynchronous SGD are summarized below:

1. In the Asynchronous case, each worker can pull the model from the parameter server, compute gradients and push the gradients to the parameter server independently. Hence CPU can be utilized efficiently as compared to Synchronous SGD, where in at every step, each worker has to wait until the parameter server has accumulated the gradients of the previous step from all the workers. This is evident in Figure 8.

2. Synchronous SGD used significantly more memory compared to Asynchronous SGD, since the parameter server has to store the intermediate gradients from all the workers until it has received the gradients form all the workers. This is demonstrated in the Figure 7

3. As shown in Figures 9 and 10, Synchronous SGD requires far more communication bandwidth than Asynchronous SGD since it has to co-ordinate the workers in every gradient update step.

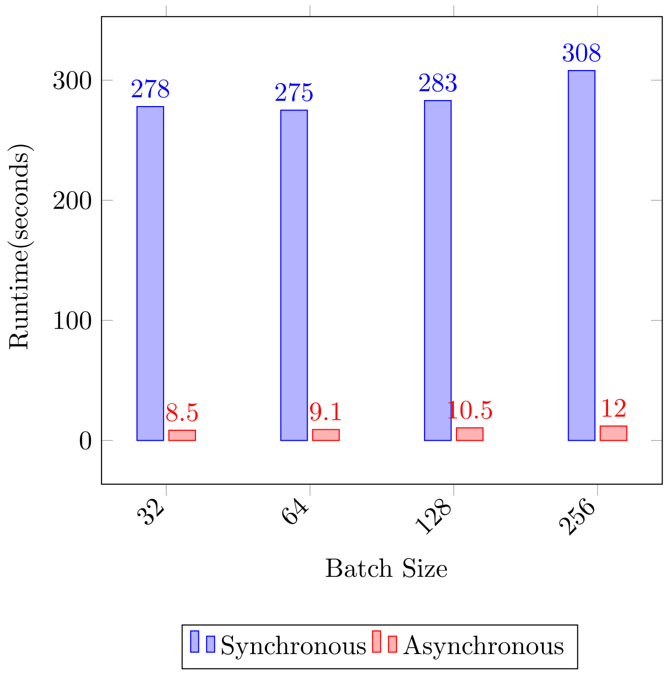

Figure 6: Runtime of Synchronous SGD and Asynchronous SGD for different Batch sizes

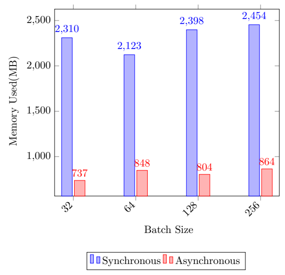

Figure 7: Memory used by Synchronous SGD vs Asynchronous SGD for different Batch sizes

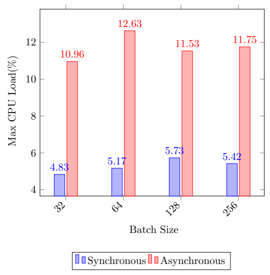

Figure 8: Max CPU load for Synchronous SGD and Asynchronous SGD for different Batch sizes

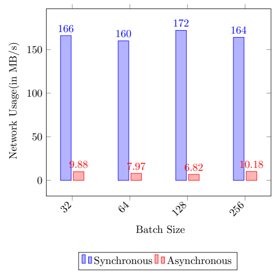

Figure 9: Receive Bandwidth for Synchronous SGD and Asynchronous SGD for different Batch sizes

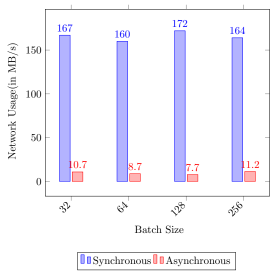

Figure 10: Transfer Bandwidth for Synchronous SGD and Asynchronous SGD for different Batch sizes

3.5.5 Varying Batch Sizes

The effects of batch size on system metrics are summarized below:

1. As shown in Figure 8, there is very little co-relation between Max CPU usage and batch size in both Asynchronous and Synchronous case.
2. As shown in the Figure 7, the memory usage increase with batch size. We suspect that the background processes are the reason for minor irregularities in the figure.
3. Execution time increases with increase in batch size as shown in Figure 6. This is because each algorithm is run for same number of epochs in all the settings of batch size and time taken to run each step increases with increase in batch size.
4. As shown in Figure 9 the network usage remains nearly same for all batch sizes. We suspect the background processes to be the reason for minor variations in the graph.

4 AlexNet

4.1 Description

AlexNet is an image classification neural network, where the input image(of 256x256 dimensions) could belong to one of thousand classes, and the output is a vector of a 1000 numbers. The i^{th} element of the output vector is the probability that the input image belongs to the i^{th} class. AlexNet consists of 5 Convolution Layers and 3 Fully Connected Layers. The first two Convolutional layers are followed by the Overlapping Max Pooling layers that we describe next. The third, fourth and fifth convolutional layers are connected directly. The fifth convolutional layer is followed by an Overlapping Max Pooling layer, the output of which goes into a series of two fully connected layers. The second fully connected layer feeds into a softmax classifier with 1000 class labels. ReLU nonlinearity is applied after all the convolution and fully connected layers.[9]

4.2 Aim of the Experiment

The assignment involved understanding Alexnet implementation and its execution on a distributed setup. For the purpose of the assignment, a library was provided to us and it consisted of the following components:

1. Datasets
2. Optimisers
3. Nets

The entry point was through a script named train.py, which took in arguments like batch size, batch numbers and the type of setup (refer Table 2). Based on the above runtime arguments, different functions were called to train the model on the images of the dataset, the labels provided, number of classes, batch numbers, batch sizes and the number of devices available to train the model on.

4.3 Algorithm

The algorithm for training the model for a single machine was already provided, our job was to train the model for clusters in synchronous mode. In the synchronous mode, we divided the input image tensor into n tensors, where n was the number of worker tasks available. Each worker then trained the model based on the split and its corresponding labels. The steps are given as follows:

1. *Global Steps* and *ModelBuilder* were initialized on the Parameter Server.
2. The optimizer *HybridMomentumOptimizer* was initialized, similar to the one used for the single machine model. It was initialized on Global Step 0 and the total number of samples.
3. The input image was split into multiple tensors, depending on number of worker tasks. The label set was also divided accordingly.
4. With the variable scope set on reuse, the function *alexnet_inference* was called on each worker task, with parameters that were the split, the corresponding label split, the number of classes, and the scope.
5. Alexnet_inference function then passed the image set through the layers of the model: (convolution layers, pooling layers and fully connected layers). Each of the layers updated the net. The activation function then used is SoftMax.

6. Loss was computed using labels.

7. The returned net, logits and loss were stored in a collection to be averaged later.

8. The gradients for each split were computed over the loss.

9. Each of the gradients were then back-propagated by the optimizer.

10. The session was run.

4.4 Evaluation

We aim to evaluate the following:

1. Visualize the graphs produced by Tensorboard for single and cluster mode of execution. (Figures 13, 14 and 15)

2. Monitor Loss and Performance characteristics for modes of execution mentioned in Table 5. (Section 4.4.1)

3. Monitor CPU, Memory and Network Utilization for modes of execution mentioned in Table 6. (Section 4.4.2)

4. Analyze the aforementioned characteristics across multiple batch sizes. (Section 4.4.3)

4.4.1 Loss and Performance

As shown in Table 5, we see that loss associated with the prediction of AlexNet algorithm trained on a single machine, is the least among all modes, and increases with increase in the number of worker tasks. In the Synchronous mode, since each worker works on its split of the image tensor, the quality of training is lower than when the training occurs on the complete image. On the other hand, the performance measures in terms of examples processed per second increases with the increase in number of worker tasks, which depicts the trade-off between accuracy and efficiency.

4.4.2 System Utilization

As shown in Table 6 and Figures 11 and 12, we see a decrease in CPU and memory utilization per node, with increase in number of worker tasks. This can be attributed to the distribution of computation in Synchronous Mode. For Network Utilization Statistics, the network read and write in single mode is negligible due to lack of data transfer or communication in the standalone architecture. For Cluster modes, we see an increase in network traffic with increase in the number of worker machines, which may be attributed to the distribution of computation and data.

4.4.3 Varying Batch Sizes

As seen in tables 5 and 6, and figures 11 and 12, we observe the following

1. Loss within a mode of operation, remains unchanged with change in batch sizes.

2. An increase in number of examples processed per second, indicating performance increases with decreasing batch numbers due to lower number of synchronizations needed.

3. CPU and Memory utilization increase with increase in batch size, due to increased computation in a single pass.

4. Network Read and Write reduce with increase in batch size due to lower communication overheads associated with fewer batches.

Mode	**Batch Size**	**Loss**	**E.g./s**	**s/Batch**
Single	64	1.96	27.6	2.329
Single	128	1.96	42.3	3.056
Single	256	1.96	43.5	5.681
Cluster	64	2.93	56.6	2.315
Cluster	128	2.93	71.1	3.462
Cluster	256	2.93	74.0	6.862
Cluster2	64	3.91	83.5	2.251
Cluster2	128	3.91	97.3	3.881
Cluster2	256	3.91	102.5	7.093

Table 5: Performance and Loss Characteristics of AlexNet

Mode	Size	CPU(%)	M(MB)	NR(rp/s	NW(tp/s)
Single	64	38.84	8842	183.17	3.61
Single	128	54.06	9540	148	3.59
Single	256	54.42	9932	176	150.40
Cluster	64	27.27	7794	98184	120601
Cluster	128	40.51	7271	66979	95581
Cluster	256	51.93	9473	33911	62023
Cluster2	64	17.13	5051	102409	229003
Cluster2	128	22.59	5760	69132	184953
Cluster2	256	30.94	7551	36034	123594

Table 6: System Utilization Characteristics of AlexNet

Where
M - Memory used in MB,
NR - Network Read in receive packets/second,
NW - Network Write in write packets/second

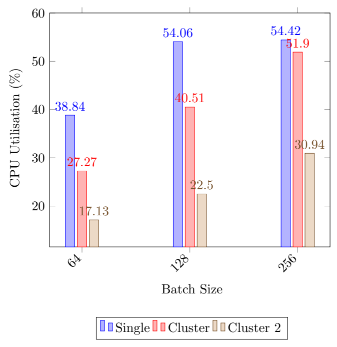

Figure 11: Max CPU load for Alexnet Classification on different modes with varying batch sizes

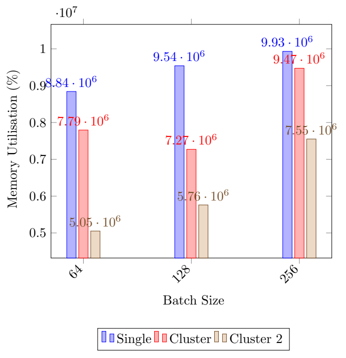

Figure 12: Max Memory Utilisation for Alexnet Classification on different modes with varying batch sizes

5 Conclusion

Our learnings from the project, helped reaffirm the objective and goals of the project. Through the project, we were able to setup the distributed version of Tensorflow, and correctly configure the environment. We were able to successfully understand the flowgraph based programming model of tensorflow. Further, we built, trained and analyzed the performance and resource utilization of a simple logistic regression model using this programming framework in both distributed and single device setup. We also implemented and analyzed a Deep Neural Network (AlexNet) used for Image classification. Furthermore, we also analyzed the performance and resource requirements of Synchronous SGD and Asynchronous SGD for the Logistic Regression model.

6 Acknowledgements

We would like to thank Prof. Akella for the interesting assignment, which served as a self-directed learning of Distributed TensorFlow concepts. We would also like to thank the TAs and our batchmates, for helping us with our queries and raising important observations.

7 Discussion

7.1 Pain points

Documentation for distributed tensorflow [7] was really hard to find. Even after finding appropriate documentation, it just seemed incomplete and was really hard to get started.

7.2 Precise Evaluation

We've used sar [5] for collecting and evaluating system metrics. Sar basically sets a cron job which collects system metrics at regular intervals. These metrics are stored in log files that can later be queried to evaluate past event occurrences and their effect on the system. Since this is a command-line based tool, this is not the best way to perform evaluations and has the following obvious flaws:

1. A person needs to be aware of the time frame of the occurrence of the event or be willing to scan through a huge list of logs for the system metric being evaluated.

2. A person needs to manually figure out trends (For example, memory usage is increasing) as opposed to intuitive way exposed by graph visualizations.

3. Co-relation of an event on multiple system metrics becomes difficult (For example, CPU load might increase due to a high volume of incoming packets)

4. Since cron jobs don't allow you to specify a granularity of less than a minute, multiple sar commands have to be put in the cron job in order to make the metric collection more granular.

We could have done a more precise evaluation by setting up a proper analytics and monitoring system like Grafana [3]. However, due to the time constraints, we didn't do this.

References

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[2] CloudLab Contributors. *CloudLab*. https://www.cloudlab.us/. Accessed 2019-04-19.

[3] Grafana Contributors. *Grafana*. https://grafana.com/. Accessed 2019-04-19.

[4] MNIST Contributors. *MNIST*. http://yann.lecun.com/exdb/mnist/index.html. Accessed 2019-04-19.

[5] sar Contributors. *Sar*. https://en.wikipedia.org/wiki/Sar_(Unix). Accessed 2019-04-19.

[6] Tensorboard Contributors. *Tensorboard*. https://www.tensorflow.org/guide/summaries_and_tensorboard. Accessed 2019-04-19.

[7] Tensorflow Contributors. *Distributed Tensorflow*. https://github.com/tensorflow/examples/blob/master/community/en/docs/deploy/distributed.md. Accessed 2019-04-19.

[8] Tensorflow Contributors. *Tensorflow*. https://www.tensorflow.org/guide/low_level_intro. Accessed 2019-04-19.

[9] Sunita Nayak. *Understanding AlexNet*. https://www.learnopencv.com/understanding-alexnet/. Accessed 2019-04-19.

Appendices

A TensorBoard Graphs for Alexnet

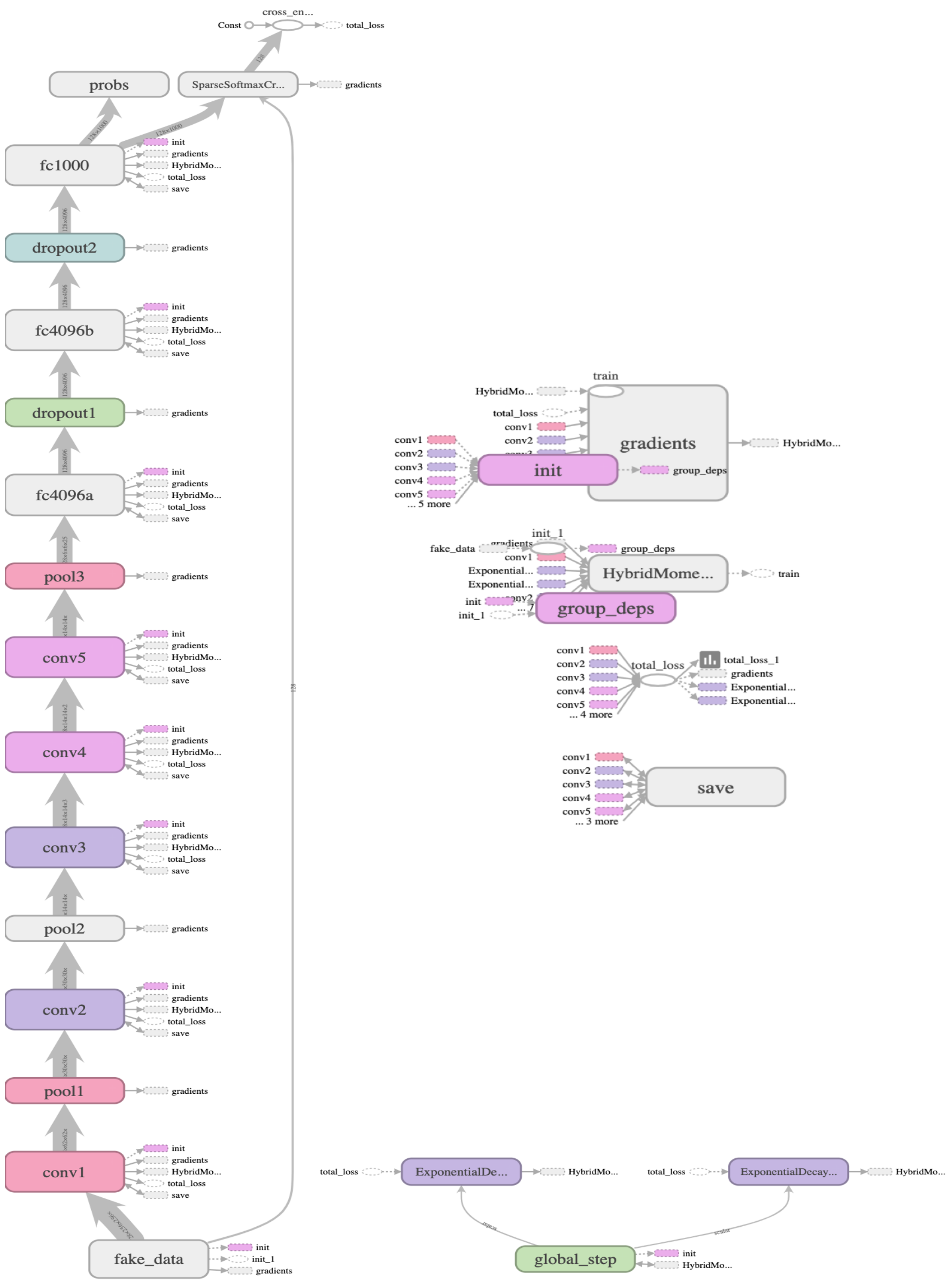

Figure 13: Computational Graph for AlexNet in Single Mode

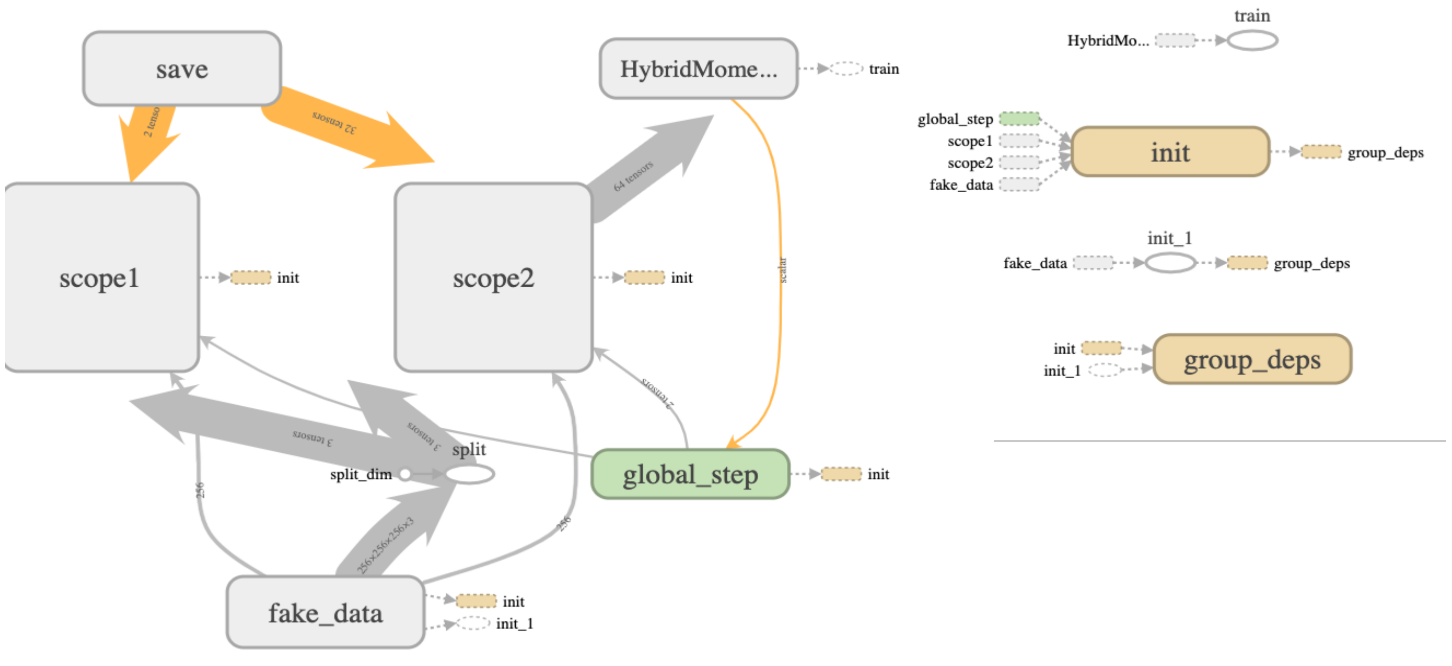

Figure 14: Computational Graph for AlexNet in Cluster Mode

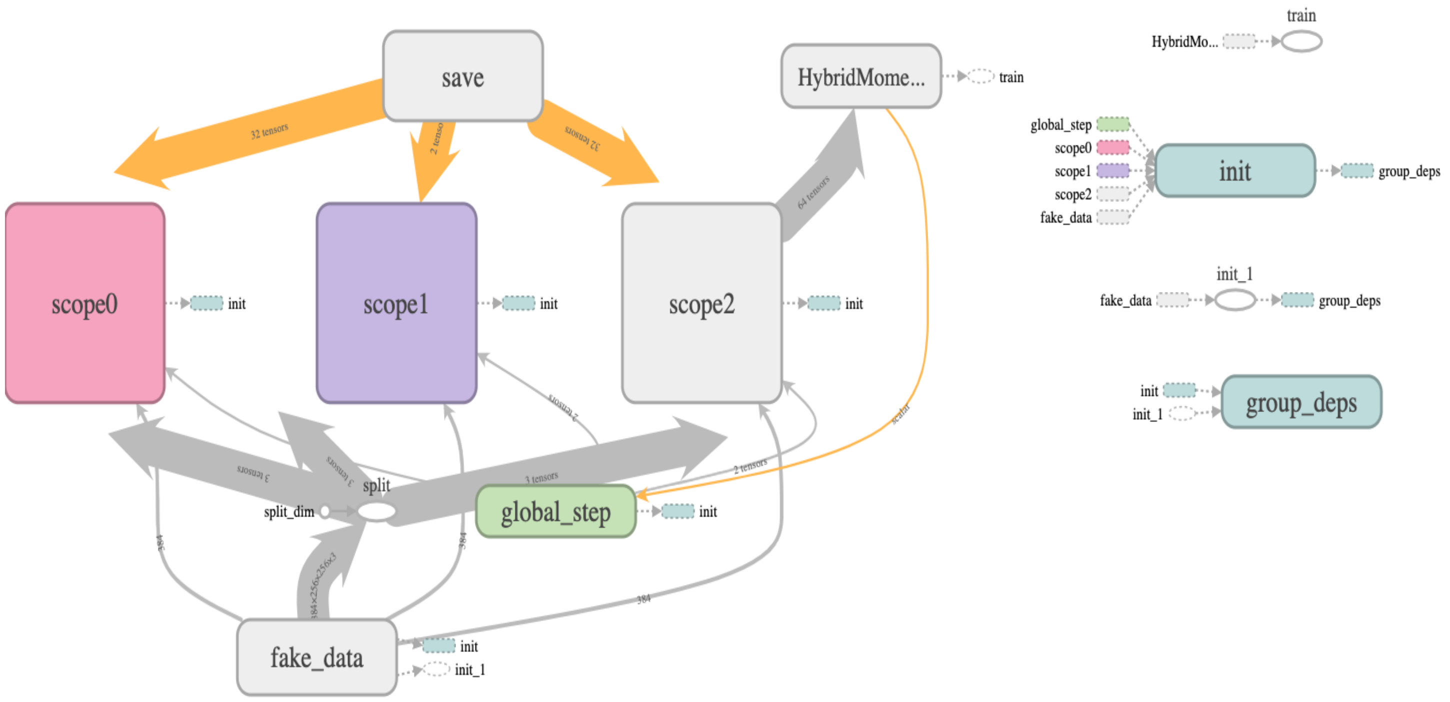

Figure 15: Computational Graph for AlexNet in Cluster2 Mode