Communication efficient distributed learning of neural networks in Big Data environments using Spark

1st Fouad Alkhoury
University of Bonn
Germany
alkhoury@cs.uni-bonn.de

2nd Dennis Wegener
Fraunhofer IAIS
Germany
dennis.wegener@iais.fraunhofer.de

3rd Karl-Heinz Sylla
Fraunhofer IAIS
Germany

4th Michael Mock
Fraunhofer IAIS
Germany

***Abstract*—Distributed (or federated) training of neural networks is an important approach to reduce the training time significantly. Previous experiments on communication efficient distributed learning have shown that model averaging, even if provably correct only in case of convex loss functions, is also working for the training of neural networks in some cases, however restricted to simple examples with relatively small standard data sets. In this paper, we investigate to what extent distributed communication efficient learning scales to huge data sets and complex, deep neural networks. We show how to integrate communication efficient distributed learning into the big data environment Spark and apply it to a complex real-world scenario, namely image segmentation on a large automotive data set (A2D2). We present evidence based results that the distributed approach scales successfully with increasing number of computing nodes in the case of fully convolutional networks.**

***Index Terms*—Federated Learning, Distributed Learning of Deep Neural Networks, Big Data Systems, Spark, Data Science Systems, Horizontal Scalability.**

I. Introduction

Neural networks turned out to be a key driver for a lot of use cases and applications in the area of machine learning and artificial intelligence. However, the training of neural networks is a complex and time consuming task. Approaches to distributed training have been developed to reduce the training time and to enable scenarios where the training process does not fit into a single computer, with federated learning being a subset of this with the special characteristic that the training data is not moved but only the weights of the models.

The process of distributed learning is mainly based on training local models on the distributed nodes and then averaging these models periodically to obtain a global model. However, for this kind of static averaging, a communication overhead is induced during the training, as it is needed to transfer the local models periodically over the network in order to compute the actual average.

In [1] it has been proposed to replace the static, periodic averaging with a dynamic averaging scheme, that synchronizes the local models during the training process only when needed. For linear models it has been shown that the prediction loss and communication can be minimized at the same time by dynamically synchronizing models in a way that the communication between the learners does not happen in the stable phases of a learning task. This approach was extended to kernelized online learners in [2]. By theoretical analysis and by an empirical experiment based on financial data it has been shown that this approach is successful. The approach was also extended to pattern learning based on Markov chains [3].

In [4] an approach for efficient decentralized learning of neural networks has been proposed which is also based on dynamic model averaging. Neural networks do not necessarily have a globally convex loss function. However, it has been empirically shown that the dynamical averaging does not only work for convex loss functions but also for non-convex cases. In detail, for the mnist dataset [5] the averaging showed to be successful.

In this work, we show how to integrate the approach for communication efficient distributed learning of neural networks into the big data framework Spark. We apply the approach to a complex real-world scenario based on a large scale real world dataset (A2D2) [6] and present results for successfully scaling the training of fully convolutional networks. We investigate empirically to which extent the communication overhead induced by the model synchronization actually contributes to the overall computation time and compute and evaluate a realistic speed-up factor. In a nutshell, it turned out that the distributed learning scaled well in Spark, with both, static and dynamic averaging. In a cluster of n GPU nodes, we achieved a speedup-factor that is proportional to $\frac{1}{n}$, e.g. in using 9 nodes, we reduced training time by the factor of 7.15.

II. Background and Related Work

As we aim at integrating the dynamic averaging method into a real Big Data system, we now first introduce the approach of applying model averaging in distributed learning. Then, we present the dynamic averaging method that we want to adapt to a the Big Data environment Spark. After that, we discuss related big data frameworks used for training deep neural networks.

This research has been funded by the Federal Ministry of Education and Research of Germany (project number 01IS19067).

A. Prior work in model averaging

Next, we first introduce the periodically averaging of models. After that, we present the main method used in this paper, namely the Communication efficient dynamic averaging method.

Sharing gradients between cluster nodes can take precious time and resources. This communication can be reduced by calculating gradients locally and communicating the sum of gradients periodically [7]. The method of averaging models periodically after computing local updates has positive effects. It keeps the privacy-sensitive data in local devices and trains a joint model. With this method, only the model parameters are sent without the need to exchange or centralize data samples or to communicate the learning algorithm.

However, this approach has some disadvantages. Either the nodes communicate so rarely that the models adapt too slowly to the changes or they communicate so frequently that they consume a big amount of unnecessary communication. Even if all models have already converged to an optimum, periodic averaging will require unnecessary communication.

The goal of communication efficient dynamic averaging is to reduce communication without losing predictive performance by investing the communication efficiently [4]. When local learners do not suffer loss, communication can be reduced; when they suffer large losses, an increased amount of communication is invested to improve their performances. This approach was achieved in several stages. First, the initial idea of communication efficient learning with linear models started with the first protocol for the distributed online prediction that aims to minimize online prediction loss and network communication at the same time [1]. The concept of this approach is to dynamically adjust the amount of communication performed depending on the hardness of the prediction problem.

The underlying idea is to perform model synchronizations only in system states that show a high variance among the local models, which indicates that synchronization would be most effective in terms of correcting the effect on future predictions. In addition to balancing the joint predictive performance, while not letting communication overhead deteriorate the responsiveness of the service. Then in 2016, this approach was extended to kernelized online learners that represent their models by a support vector expansion [2]. As a result, the protocol achieves similar service quality as any periodical communication protocol while communicating less by a factor depending on its loss.

After that, another extension of the approach to pattern learning was made in 2018 [3]. The main idea of extension is to design and implement an online, distributed and scalable pattern prediction system over massive streams of events, related to trajectories of moving objects. The approach combined a distributed online prediction protocol with an event forecasting method based on Markov chains. In this paper, we present the integration of communication efficient dynamic averaging method in real Big Data architecture using Spark.

B. Related work in distributed deep learning frameworks

There exists a variety of distributed deep learning frameworks (see [8] for an overview). Here, we focus on those which are most relevant for our work.

Spark is a unified analytics engine for large-scale data processing. It was developed at UC Berkeley in 2009 and has become the largest open source engine in Big Data [9]. It runs on memory (RAM) that makes the processing faster than on disk and faster than previous approaches to work with Big Data like MapReduce [10]. Spark provides high-level APIs in Java, Scala, Python and R. These APIs include a collection of operators for transforming data.

Sparknet is a framework for training deep networks in Spark [11]. In each iteration, the Spark master broadcasts the model parameters to the workers, then each worker runs Stochastic Gradient Descent on the model with its partition of data. Federated learning is done by data parallelism on partitioned data and the parameters are broadcast periodically. To test the scalability, an experiment was done to train the default Caffe model of AlexNet [12] on the ImageNet dataset [13]. The experiment ran on a cluster of 3,5, and 10 nodes. For comparison, another experiment ran Caffe on a single GPU and no communication overhead. The results showed that one GPU takes 55.6 hours to obtain an accuracy of 45%. While with 3,5, and 10 GPUs, SparkNet takes 22.9, 14.5, and 12.8 hours, giving speedups of 2.4, 3.8, and 4.4.

Intel Corporation BigDL is a distributed deep learning library for Apache Spark [14]. To study the scalability of the distributed training of BigDL, an ImageNet Inception-v1 model was trained using BigDL with various node counts. The results showed that the synchronization overheads represent a small fraction compared to the model computation time.

Horovod [15] is a distributed deep learning training framework for TensorFlow [16], Keras [17], PyTorch [18], and Apache MXNet [19]. In addition, it aims to scale a single-GPU training script to train across many GPUs in parallel [20]. Horovod uses the Message Passing Interface (MPI) to orchestrate single/multi-worker training in a High-Performance computing setup. In Horovod, each worker passes parameter updates to a neighboring worker in a ring topology.

The Federated learning approach of [21] is a collaborative machine learning method without centralized training data. This approach enables mobile phones to collaboratively learn a shared prediction model while keeping all the training data on the device. In fact, the device downloads the current model, improves it by learning from data on the same device. After that, it summarizes the changes as a small update. Then, this update is averaged with other user updates to improve the shared model.

To sum up, the referenced examples of learning algorithms update the parameters periodically either in a centralized approach, or continuously in a ring topology. This is the main difference to the algorithmic approach of efficient distributed training [4] we apply, where the parameters are synchronized conditionally if a threshold on divergence is passed.

III. Communication Efficient Distributed Training of Neural Networks in Spark

In this section we show how to integrate the approach for communication efficient distributed learning of neural networks into Spark. First, we briefly summarize how the approach of Dynamic Model Averaging [4] works. Next, we show how we managed the data distribution in Spark and explain the local learning that is performed on the distributed data partitions. Finally, we describe how the distributed training with dynamic model averaging works in Spark.

A. Communication Efficient Distributed Training

Our approach is based on the dynamic averaging method [4] explained in section II-A. In the following, we present some basics of this method that we utilize for our integration. We first define the local training procedure on one node, then we explain the entire distributed training on n nodes. The local procedure trains a model on a data partition and outputs a local model. Let us assume that a data point x is the input to the neural network that plays the role of the function $f_i(x)$ where $i \in [1, n]$. The output of $f_i(x)$ is the predicted value y, and we compute the loss by comparing it to the true value $\overline{y}$. The same learning procedure takes place on each node using a fixed structure of the neural network, i.e. all $f_i(x)$ have the same structure. However, the set of neural networks differentiate from each other by the learned weights. As a result, the output of the local training is a local model m_i on the node i.

To perform the static distributed learning, all learners start the training from the same model. To achieve that, the node which manages the control flow and the global state of the computation (master) broadcasts the initial global model to computing nodes. The training process starts on each computing node (worker) using an identical model and operates on a partition of data. After each iteration, these local models are synchronized by collecting the local models from the worker nodes and averaging them in the master node.

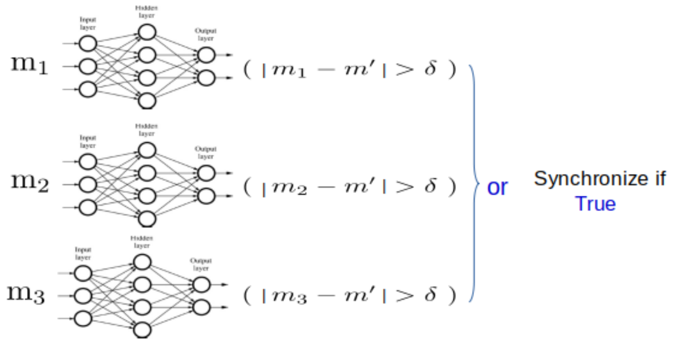

Fig. 1. This figure shows the condition required to synchronize the models. If one of the local models m_i diverges from the global model m' we need to synchronize.

In the dynamic averaging method, we check the divergence between the models trained locally and the global reference model after each iteration. If the difference between any local model m_i and the global model m' surpasses the divergence threshold $\delta \in R^+$ we need to synchronize the local models (see Fig. 1). The new averaged model will be distributed again as a global model to the worker nodes. Thus, using dynamic averaging reduces communication overhead needed, as the model synchronization is no longer performed after each iteration, but only when the models diverge significantly.

B. Conceptual Approach for Spark

In this section we present how we map the concept of distributed learning based on dynamic averaging onto the Spark framework. We use PySpark as Python interface to Spark [9], and PyTorch [18] to train the neural networks. Furthermore, we used Jupyter Notebooks [22] for the execution of the code. In order to apply the dynamic averaging method, we define User Defined Functions (UDF) to implement some necessary functions that do not exist in PySpark build-in functions. In the following we present the concept and implementation of the distributed learning process divided into 7 different steps. The approach is illustrated in Fig. 2 based on an image processing use case that we will later on use for the evaluation.

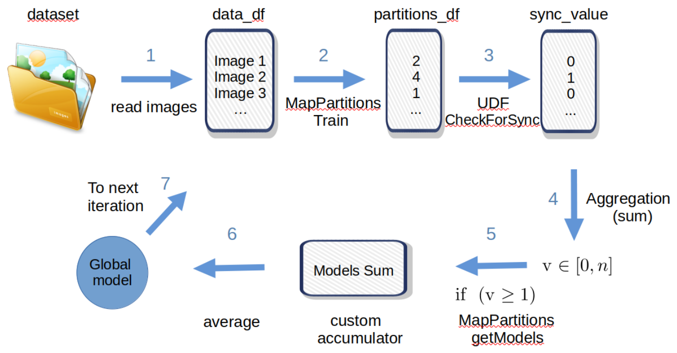

Fig. 2. This figure shows the workflow of the training process in 7 steps and the roles of the different Spark aspects such as UDF, the Accumulator and the Aggregation sum.

To start the learning experiment, we first configure Spark settings such as the master URL, driver and executor memory, number of workers and other important parameters. Then, we define the experiment's main variables such as the count of iterations and batch size. After that, we store the data images into a DataFrame as shown in **Step 1** in Fig. 2. The whole dataset is partitioned onto the worker nodes such that a single partition of data is assigned to a single process on a node and the nodes can process the data in parallel. Reading images and dividing them into approximately equal-sized partitions is managed by Spark. To initiate the training, the master node broadcasts the initial global model to the worker nodes.

As we need to run the same training process on each node, we use the function mapPartitionsWithIndex **(Step 2)**, which applies the training function to each partition of the data. Also, we need to track the index of the partition in order to continue the training from the same local model when there is no need for synchronization. Once the training iteration ends, the output model is saved on the local disk. Then, we check in **Step 3** whether the local models diverge from the global model or not (see also Fig. 1). We define a User Defined Function and call it checkForSync to check the divergence condition for each partition. It calculates the differences between the local and the global model and returns 1 if the difference surpasses the divergence threshold, or 0 otherwise. The boolean variables in Fig. 1 are expressed here as integers. The true variable is mapped to 1 and the false to 0. Then, in **Step 4** the returned values of the checkForSync UDF are aggregated to decide whether we should synchronize the models or continue the training without communication. We used here the Aggregation sum function to compute the sum of the returned integers from the previous step. Note that the logical OR in Fig. 1 is expressed here by the sum operation.

If there is no need to perform a synchronization, we move directly from **Step 4** to **Step 7**. Otherwise, we accumulate the models' weights in **Step 5** using the Accumulator variables in PySpark. The Accumulators are used to gather information and update counters across different executors. We used a primitive type accumulator to track the differences between local models and the global model. Also, we used a custom accumulator defined by the AccumulatorParam class to aggregate the network parameters in order to average them afterwards. In the driver program, we created an accumulator variable with an initial value of zero weights network. Then, the weights are aggregated on the driver side. To compute the average, each weight of the accumulated model is simply divided by the number of partitions, so we get the averaged global model (**Step 6**). The resulting model will be the new global model which is sent again to all worker nodes. Since our research is conducted to learn the model in a communication efficient way, each node will receive only one copy of the global model even if it contains several partitions. Thus, we also save communication while transferring models.

In the next rounds, we repeat the training process. The initial model on each node is either the global model, if the previous round ended with synchronization, or the training continues with the local model saved from the previous round. After the final round of training, a synchronization is done - even if the difference of the models does not surpass the threshold - to obtain the final global model for the experiment.

IV. Experimental Evaluation

In this section, we investigate the effectiveness of applying the approach of Communication Efficient Distributed Training of Neural Networks in Spark. We want to answer two basic questions: how effective is the communication saving of the approach, and does the approach scale horizontally in a complex scenario.

In order to apply our method to a real-world dataset, we used the Audi Autonomous Driving Dataset (A2D2) [6]. This dataset was published to support academic researchers working on autonomous driving. The dataset features 2D semantic segmentation and consists of 23 different driving scenes recorded from different views such as front center camera, side left camera, rear center camera,... etc. Each scene holding a series of frames. In our experiments, we took the subset of frames taken from front center camera. The total number of frames in this subset is 26591. Each pixel in a frame is given a label describing the type of element it represents, such as car or sky.

In the following we describe our procedure to evaluate the dynamic averaging method when training a Fully Convolutional Network (FCN) [23], which predicts the pixels' labels.

A. Effectiveness of the communication saving

First, we apply the static and dynamic method in a cluster of 9 nodes and check whether we achieve the same level of accuracy as in a reference training. Then we examine the amount of communication we could save using the dynamic approach and finally we investigate the improvement of the computation time.

1) Predictive Performance: For a reference of an acceptable predictive performance, we first ran the training of an FCN implementation [24] on a single node without using Spark with a split ratio (80%, 20%) of training and testing data. The experiment total time was 106h:35min and it took 28 epochs to get acceptable predictive performance measured by accuracy and mean Intersection over Union (mIoU). Then, we ran the distributed experiment on 9 nodes with data parallelism based on Spark. We trained for the same number of epochs and used the same split ratio as for the reference experiment. In this experiment, we applied static synchronization and averaged the local models after each epoch and updated the global model on each of the 9 nodes. The application of this method showed a performance close to the reference experiment. The pixel accuracy of the distributed experiment was 0.940 and the mean Intersection over Union (mIoU) was 0.455, compared to 0.945 and 0.483 for the local experiment. This distributed experiment took 15h: 06 min and thus executed 7.06 times faster than the single node reference experiment. Finally, we ran the dynamic version of our distributed experiment. In this experiment, the algorithm compares the each local model after each epoch of training with the last global model. If the difference between any of these local models and the global model surpasses the threshold δ, the synchronization takes place and the new global model is distributed to all cluster nodes. The difference between the two models is the sum of the absolute differences between the corresponding parameters. Based on a small series of experiments we set the value of the threshold, namely 20000, as a guessed medium value between a lower bound of values that always cause synchronization and an upper bound too high to cause synchronizations. This threshold is used in all experiments that perform the dynamic method for our scenario. Compared to the static method, the dynamic method performs with almost the same accuracy and mIoU using only 6 synchronizations during the entire training process with 28 epochs. Fig. 3 shows the comparison between the static and the dynamic synchronization.

As a cross check, we also ran an experiment that performed a synchronization only once after 28 epochs. This case is equivalent to choosing a high value of the divergence threshold. Postponing the synchronization to the end gives worse results compared to dynamic or static averaging, as displayed

TABLE I
THE TABLE SHOWS A COMPARISON BETWEEN THE SINGLE NODE EXPERIMENT AND THE THREE VARIANTS OF THE DISTRIBUTED EXPERIMENT.

	Single-node experiment	Static Sync.	Dynamic Sync.	Sync. once at the end
Count of syncs	-	28	6	1
Accuracy	0.945	0.940	0.942	0.924
mIoU	0.483	0.455	0.447	0.292
Total time	106 h: 35 min	15 h: 6 min	14 h: 55 min	14 h: 52 min

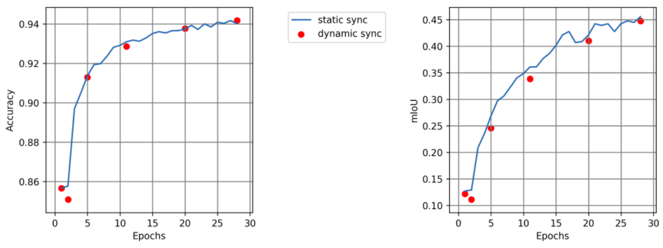

Fig. 3. This figure shows the accuracy comparison between the static and dynamic synchronization. The 6 red points represent the 6 synchronization events during training.

in Table I. By this, we show that intermediate synchronisations are necessary to obtain an acceptable predictive performance.

Summing up, the dynamic approach reduces the number of synchronizations compared to the static method, but still offers the same high quality of the predictive performance.

2) Network communication: In a distributed experiment using n nodes, we need to transmit $n-1$ local models per synchronization because in our setup the master node also acts as a worker and does the training on one partition. In the dynamic experiment, synchronization happened 6 times out of 28 epochs. Each synchronisation causes 3688 MB of network traffic. Thus, compared to the static synchronization experiment, the dynamic synchronisation avoids 81136 MB network traffic within the whole training.

This confirms the original hypothesis that we save on network communication and do not lose predictive power even with a complex large task. Fig. 4 shows the difference in network traffic between the static and the dynamic approach.

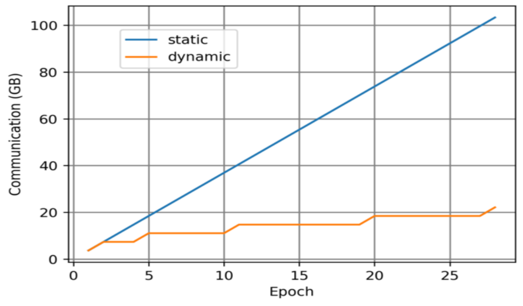

Fig. 4. The plot shows the difference in network communication between static and dynamic averaging methods.

3) Communication time: Considering the efficiency of the training process in terms of communication time, our experiments showed that the total time for the static averaging is 15h:06min, while it's 14h:55min for the dynamic averaging (see Table I). Contrary to our expectations, these results have not shown a significant difference (just 11 minutes) concerning the training time between the two averaging approaches.

As shown in Fig. 5, the synchronization time is minimal compared to training time. The process of model aggregation and averaging towards a new global model took only 32 seconds per round, which is around 1.7% of the training time itself. A synchronization happened 6 times in the dynamic experiment. Thus, the time saved is: $(28-6)\times 32$ sec $=$ 11 min : 44 sec. In terms of communication time, there is only a very low benefit of the dynamic approach.

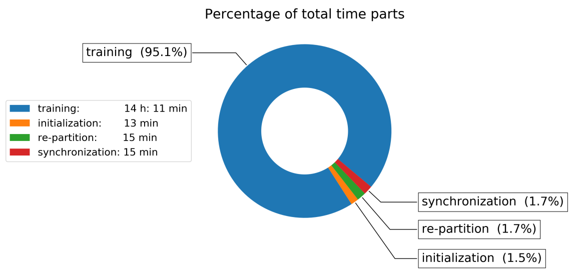

Fig. 5. The figure shows how the total time of the distributed learning experiment is divided. The synchronization part is minimal compared to the training.

A more detailed look at the synchronization times for experiments with 3 to 9 nodes shows that the synchronization has a base load of approx. 27 seconds. Model transmission from a worker to the master causes a network traffic of about 500MB. As the number of nodes increases, the synchronization time increases by less than a second per additional model. This is closely the transmission time of 500MB per model in the 10GBit network configured in our cluster.

B. Horizontal Scalability

In order to evaluate if our approach scales horizontally in a complex scenario, we run the distributed learning experiment on different numbers n of nodes (3 to 9). We calculate the speedup factor $S(n)$ by dividing the experiment time on a single node by the time needed using n nodes. Table II shows time and performance details of these experiments. Note that the column *'Distribution'* represents the time needed to partition and distribute the data before starting of the training *'Computation'* over a series of 28 epochs.

Amdahl's Law [25] states that if we apply n processes to a task that has a serial fraction σ, then the task will approach a speedup limit that is given by the following formula:

$$S(n)=\frac{n}{1+\sigma(n-1)} \quad (1)$$

TABLE II
THE TABLE SHOWS THE RESULTS OF APPLYING THE DYNAMIC AVERAGING METHOD ON MULTIPLE NUMBER OF WORKER NODES.

Nodes count	3	4	5	6	7	8	9
Distribution	1h:25m	1h:2m	51m	42m	36m	32m	29m
Computation	42h:16m	30h:56m	25h:18m	21h:21m	18h:05m	16h:01m	14h:11m
Serial Time	12m:4s	12m:6s	12m:26s	13m:7s	13m:12s	14m:6s	15m:20s
Total time	43h:53m	32h:10m	26h:21m	22h:16m	18h:55m	16h:46m	14h:55m
Accuracy	0.952	0.949	0.948	0.946	0.943	0.942	0.942
mIoU	0.477	0.471	0.467	0.461	0.458	0.452	0.447
Syncs count	8	8	8	7	7	6	6
Speedup factor	2.43	3.31	4.04	4.79	5.63	6.36	7.15

In our case, σ is the fraction of the time of the serial algorithmic parts performed by the master, e.g. the accumulating and averaging the model parameters, or of algorithmic sections that are equally performed on each worker node independent of the scale of distribution, e.g. the initialization of the local models and the execution of the checkForSync function. The value $(1-\sigma)$ is the fraction of training time performed by the data-parallel execution on the worker nodes.

An extension of Amdahl's Law, called the Universal Scalability Law [26] (USL), introduces an additional coefficient of performance that reflects delays due to communication between nodes. The USL is given by the following equation:

$$S(n) = \frac{\gamma n}{1 + \alpha(n-1) + \beta n(n-1)} \tag{2}$$

The coefficient γ represents the slope in the case of ideal parallelism, α defines the serial coefficient, and β represents additional delays. We fit the USL coefficients to our *Speedup* values from Table II using [27] and get the following equation:

$$S(n) = \frac{0.8266n}{1 + 0.00525(n-1) + 3.78 \times 10^{-16} n(n-1)} \tag{3}$$

The extreme small value of β corresponds to the marginal fraction of synchronization time as shown in Fig. 5. The single-node experiment took 106h:35min. Therefore, the time required on n nodes is given by the following equation:

$$T(n) = 106.58 \times S(n)^{-1} \tag{4}$$

As can be seen in Fig. 6, our results for nodes count $n \in [3, 9]$ fit the Universal Scalability Law almost exactly.

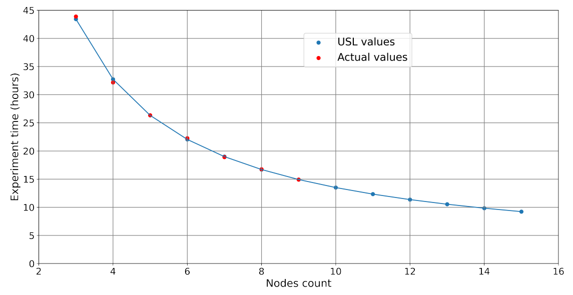

Fig. 6. The plot shows the relation between the total time and the number of nodes. The blue curve represents the time values computed by the Universal Scalability Law while the red points represent the actual time of our experiments.

C. *Experiment Wrapup*

We have investigated the two main questions about the effectiveness of the dynamic averaging method and the scalability. Our evidence-based results indicate that the distributed approach of Communication Efficient Distributed Learning performs successfully even on a large realistic task of training a fully convolutional network (FCN) [23]. Moreover, our results highlight the importance of doing dynamic averaging on intermediate steps. According to our expectations, we demonstrated empirically that we achieved high predictive performance with reduced network communication. Regarding communication time, it turned out that the synchronization time is minimal compared to the training time when we use a complex network and large dataset, even if we speed up training by data parallelism. Finally, we presented evidence-based results that the distributed approach scales successfully with an increasing number of computing nodes. Further results and detailed evaluations can be found in [28].

V. CONCLUSION

In this paper we have shown how to integrate communication efficient distributed learning of neural networks into the big data framework Spark. The integration is based on an existing method of dynamic model averaging which only synchronizes local models if they significantly diverge from the global model. Thus, it is a viable alternative with reduced communication compared to distributed learning frameworks, which are based on periodic averaging. By our approach, we were able for the first time to perform distributed learning of neural networks in a big data environment using Spark in a communication efficient way. In detail, we showed that the approach based on dynamic model averaging can achieve the same accuracy as with static periodic averaging. Communication is reduced, but for large models, the time needed for the synchronization of the models is very low compared to the duration of the whole training process. Lastly, we investigated to what extent the approach scales to huge datasets and complex deep neural networks. Our experiments on the real-world dataset A2D2 showed that our approach scales out successfully for fully convolutional networks. In detail, the speedup factor achieved for static and dynamic averaging on 9 nodes reduced the training time by a factor of 7.15. We believe that further work needs to be done to investigate on which deep networks we could apply the averaging method.

References

[1] M. Kamp, M. Boley, D. Keren, A. Schuster, and I. Sharfman, "Communication-efficient distributed online prediction by dynamic model synchronization," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECMLPKDD)*. Springer, 2014.

[2] M. Kamp, S. Bothe, M. Boley, and M. Mock, "Communication-efficient distributed online learning with kernels," in *Machine Learning and Knowledge Discovery in Databases*, P. Frasconi, N. Landwehr, G. Manco, and J. Vreeken, Eds. Springer International Publishing, 2016, pp. 805–819. [Online]. Available: http://michaelkamp.org/wp-content/uploads/2020/03/Paper467.pdf

[3] E. Qadah, M. Mock, E. Alevizos, and G. Fuchs, "A distributed online learning approach for pattern prediction over movement event streams with apache flink," in *Proceedings of the Workshops of the EDBT/ICDT 2018 Joint Conference (EDBT/ICDT 2018), Vienna, Austria, March 26, 2018*, ser. CEUR Workshop Proceedings, N. Augsten, Ed., vol. 2083. CEUR-WS.org, 2018, pp. 109–116. [Online]. Available: http://ceur-ws.org/Vol-2083/paper-17.pdf

[4] M. Kamp, L. Adilova, J. Sicking, F. Hüger, P. Schlicht, T. Wirtz, and S. Wrobel, "Efficient decentralized deep learning by dynamic model averaging," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2018. [Online]. Available: http://michaelkamp.org/wp-content/uploads/2018/07/commEffDeepLearning_extended.pdf

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] J. Geyer, Y. Kassahun, M. Mahmudi, X. Ricou, R. Durgesh, A. S. Chung, L. Hauswald, V. H. Pham, M. Mühlegg, S. Dorn, T. Fernandez, M. Jänicke, S. Mirashi, C. Savani, M. Sturm, O. Vorobiov, M. Oelker, S. Garreis, and P. Schuberth, "A2D2: audi autonomous driving dataset," *CoRR*, vol. abs/2004.06320, 2020. [Online]. Available: https://arxiv.org/abs/2004.06320

[7] J. Chen, R. Monga, S. Bengio, and R. Józefowicz, "Revisiting distributed synchronous SGD," *CoRR*, vol. abs/1604.00981, 2016. [Online]. Available: http://arxiv.org/abs/1604.00981

[8] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *CoRR*, vol. abs/1802.09941, 2018. [Online]. Available: http://arxiv.org/abs/1802.09941

[9] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. USA: USENIX Association, 2010, p. 10.

[10] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, p. 107–113, Jan. 2008. [Online]. Available: https://doi.org/10.1145/1327452.1327492

[11] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, "Sparknet: Training deep networks in spark," 2016.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "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., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014. [Online]. Available: http://arxiv.org/abs/1409.0575

[14] J. J. Dai, Y. Wang, X. Qiu, D. Ding, Y. Zhang, Y. Wang, X. Jia, C. L. Zhang, Y. Wan, Z. Li, J. Wang, S. Huang, Z. Wu, Y. Wang, Y. Yang, B. She, D. Shi, Q. Lu, K. Huang, and G. Song, "Bigdl: A distributed deep learning framework for big data," *CoRR*, vol. abs/1804.05839, 2018. [Online]. Available: http://arxiv.org/abs/1804.05839

[15] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[17] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[19] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015.

[20] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[21] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," 2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[22] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and J. development team, "Jupyter notebooks ? a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Scmidt, Eds. IOS Press, 2016, pp. 87–90. [Online]. Available: https://eprints.soton.ac.uk/403913/

[23] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1605.06211, 2016. [Online]. Available: http://arxiv.org/abs/1605.06211

[24] K. Wada, "pytorch-fcn: Pytorch implementation of fully convolutional networks," 2017. [Online]. Available: https://github.com/wkentaro/pytorch-fcn

[25] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: Association for Computing Machinery, 1967, p. 483–485. [Online]. Available: https://doi.org/10.1145/1465482.1465560

[26] N. J. Gunther, *Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[27] W. Wang, "pyusl: Universal scalability law in python," 2021. [Online]. Available: https://github.com/wip727/PyUSL

[28] F. Alkhoury, "Communication efficient distributed learning using spark," Master's thesis, University of Bonn, 2021.