Performance Optimization on Model Synchronization in Parallel Stochastic Gradient Descent Based SVM

1st Vibhatha Abeykoon
Intelligent Systems Engineering
Indiana University
Bloomington, USA
vlabeyko@iu.edu

2nd Geoffrey Fox
Intelligent Systems Engineering
Indiana University
Bloomington, USA
gcf@iu.edu

3rd Minje Kim
Intelligent Systems Engineering
Indiana University
Bloomington, USA
minje@iu.edu

***Abstract*—Understanding the bottlenecks in implementing stochastic gradient descent (SGD)-based distributed support vector machines (SVM) algorithm is important when it comes to training larger data sets. The communication time to do the model synchronization across the parallel processes is the main bottleneck that causes inefficiency in the training process. In order to produce an efficient distributed model, the communication time in training model synchronization has to be as minimum as possible while retaining a high testing accuracy. The effect from model synchronization frequency over the convergence of the algorithm and accuracy of the generated model must be well understood to design an efficient distributed model. In this research, we identify the bottlenecks in model synchronization in parallel stochastic gradient descent (PSGD)-based SVM algorithm with respect to the training model synchronization frequency (MSF). Our research shows that by optimizing the MSF in the data sets that we used, a reduction of 98% in communication time can be gained (16x - 24x speed up) with respect to high-frequency model synchronization. The training model optimization discussed in this paper guarantees a higher accuracy than the sequential algorithm along with faster convergence with MSF optimization.**

***Index Terms*—model synchronization, sgd, svm, distributed communication optimization, scaling svm**

I. Introduction

Support vector machines (SVM) are an important classification algorithm in the supervised machine learning domain. In training SVM for larger data sets, the most important thing is to identify the bottlenecks in training. The main reason is that the time to train an SVM is computationally higher when it comes to dealing with high volume data with higher dimensions. Distributed version of SVM is an effective solution to this problem. In scaling distributed support vector machines algorithm, the most important thing is to determine the bottlenecks in scaling the algorithm. Number of processes and limitation of resources in the distributed environment [1] is vital to determine optimized performance. In scaling the algorithm across the cluster resources, there are two types of overheads that has to be dealt with. The major challenge is avoiding the communication overhead in synchronizing distributed models which causes a lag in performance. The next challenge is to identify the core algorithm used in the SVM to minimize the computation overhead. In referring to the computation overhead, there are many versions of SVM algorithm which has provided various optimization to improve the performance in the sequential algorithm [2], [3], [4], [5], [16]. The realization of a faster computational model will enhance the computation model and reduce the computation overhead in nodes in the distributed cluster. In realizing the communication overhead, depending on the limited memory requirements and computing resources in the cluster, the distributed algorithm has to be optimized [11], [12], [14]. In order to provide a highly efficient training model, the developed distributed algorithm must be communication-efficient and computation-efficient. Apart from that, the configurations in communication model has to be optimized to obtain higher accuracy along with an efficient training model. In distributed SVM, the training model synchronization across the distributed nodes is a very important fact to gain higher accuracy and efficiency in training. In this research, we thoroughly look into the communication overhead caused by the model synchronization against the frequency of synchronization. In an optimum and efficient distributed model, the communication cost should be relatively lesser than the computation cost. With the faster execution, maintaining a high accurate predictive model along with the convergence of the algorithm with respect to scaling must be thoroughly understood. Through out this research, we analyze the effects of faster execution over accuracy of the predictive model along with minimizing the bottlenecks in scaling the SVM algorithm in distributed environments. In this paper, we discuss how to guarantee the higher testing accuracy and faster convergence by frequent model synchronization along with minimizing communication overhead caused by frequent model synchronization in the distributed paradigm.

In section II we discuss the related work done on SVM, in section III, the mathematical aspects of the SVM training model is discussed. In section IV, the nature of the traditional sequential algorithm and the effect by the frequent model synchronization in parallel algorithm will be discussed along with simulating that effect on the sequential version of the parallel algorithm. In section V the conducted experiments

and results will be explained with respect to the methodologies discussed in section IV. The conclusions and future work of the current research is discussed in section VI.

II. Related Work

Support Vector Machines (SVM) by Cortes and Vapnik [2] can be considered as one of the earliest methodologies used in the supervised learning-based classification. There are couple of sequential implementations like DC-SVM [3], LibSVM [4] and Sequential Minimal Optimization (SMO) [5], [6], [7], [8] which can be considered as most prominent sequential implementations to solve the SVM problem. Building a SVM model becomes computationally expensive depending on the number of data points and dimension of a data point in the data set. For a data set having few hundreds of Mega Bytes can cause memory bound issues when the algorithm has to compute a kernel matrix of size $n \times n$ where n is the number of data points in the data set. To overcome this problem there have been many studies done considering random samples via bootstrap techniques [9], described in SVM ensemble. But the performance improvement on nature of execution for very large data sets has not been elaborated for bigger data sets in these studies. In LibSVM, DC-SVM and most of the SVM-based implementations, the core algorithm used is the SMO algorithm which is computationally expensive. The parallel implementations done on SMO-based SVM by Keerthi et al in [10] can be recognized as one of the earliest work on this problem. But the SMO itself is a computationally intensive model due to the high overhead in optimizing Lagrangian multipliers in an iterative way. Apart from parallel SMO, there have been matrix approximation methods that has been used to work on the memory-based overhead in the traditional SMO algorithm [11], [12]. The matrix factorization methods and SMO-based algorithms are still computationally intensive and it doesn't provide a pleasingly parallel model.

In the recent research, to avoid this problem involved with model parallelism overhead, gradient descent-based optimization [13] has been widely used. The main reason for the overhead in model parallelism in traditional SVM algorithm is due to solving the quadratic objective function using a linear equation system using Lagrangian multipliers. In this regard, SGD-based approaches are an alternative solution because the computation of stochastic gradient step is much faster than solving a set of linear equations. Instead of solving a linear equation system, the problem can be solved by minimizing the objective function using a traditional SGD-based approach and estimating the weights that satisfy the minimal objective function. The SGD-based approaches have been widely used in pPackSVM [14] and fast feature extracting SVM approaches [15]. Pegasos [16] is another prominent SGD-based SVM optimization done with an adaptive decreasing learning rate, which provides a guaranteed convergence in lesser number of epochs.

III. Background

In our research we focus on linear kernel-based binary classification on three different data sets. The Epsilon [17] dataset contains 400,000 samples with 2,000 features; Ijcnn1 [18] dataset contains 35,000 samples with 22 features; Webspam [19] contains 350,000 samples with 254 features. In referring to the mathematical background associated with SGD-based SVM, in the sample space of S with n samples, x_i refers to a d-dimensional feature vector and y_i is the label of the i^{th} data point as shown in (1).

$$S = \{x_i, y_i\}$$
$$\text{where } i = [1, 2, 3, ..., n],\ x \in R^d,\ y_i \in [+1, -1] \quad (1)$$

In the SGD approach the objective is to minimize the objective function in (2) with the constraint on the optimization defined in (3):

$$J^t = \min_{w \in R^d} \frac{1}{2} \|w\|^2 + C \sum_{x,y \in S} g(w; (x, y)) \quad (2)$$

$$g(w; (x, y)) = \max(0, 1 - y\langle w|x\rangle) \quad (3)$$

Depending on the value of the constraint function, the weight update will be done as in (4) with the learning rate α by considering (5) as the derivative depending on the value obtained for the expression in (6). C in (2) refers to a tuning hyper parameter. w in (2) refers to weight vector.

$$w = w - \alpha \nabla J^t,\ \alpha = \frac{1}{1+t} \quad (4)$$

$$\nabla J^t = \begin{cases} w & \text{if } \max(0, 1 - y\langle w|x\rangle) = 0 \\ w - Cx_i y_i & \text{Otherwise} \end{cases} \quad (5)$$

$$y\langle w|x\rangle = y_i w^T x_i \quad (6)$$

In the experiments conducted in this paper, we use a learning rate α which is decaying with the epoch number t. The communication overhead in frequent model synchronization has not been discussed with respect to the convergence of the algorithm on lower objective function value and higher cross validation accuracy. In this paper we analyze how the MSF affects the faster convergence and faster execution of the distributed SGD-based SVM algorithm.

IV. Methodology

Our objective is to analyze the effect from model synchronization on faster convergence and to see how model synchronization communication overhead can be optimized to run the training model in an efficient way. In this regard, we modified the original sequential algorithm to get the same effect caused by the parallel model synchronizing algorithm to verify the accuracy of the distributed model we built. The model synchronization resembles synchronizing the training weight vector in all parallel machines by averaging over the sum of a local weight vector in each machine. Model synchronization in the distributed mode considers each

machine in the system as a single block of a sequential algorithm, where each machine updates the model per each data point and do a model synchronization after each machine has calculated the corresponding model. This is the atomic-level model synchronization that can take place in the frequent model synchronization. It is clear that there can be a model synchronization overhead caused by frequent synchronization due to inter-process communication and this effect will be addressed in section V. In this paper, we refer to a term called model synchronization frequency (MSF) which refers to the number of data points used to calculate the model before synchronizing it with other processes in the distributed training mode. Note that if data points used for model synchronization is unity ($= 1$), it is considered as a higher MSF as we will be synchronizing the models after each data point in each process is done with calculating the model. If the data points used per synchronization is a large number L, it means there will be a model synchronization happening after calculating weight for L data points in each process which implies that the MSF is low. First, we focus on determining the accuracy of the distributed model synchronizing algorithm that we introduce with respect to the sequential version of the distributed computation model without the communication implementation. Then we focus on implementing the distributed version with configurable MSF value to observe the convergence of the algorithm with respect to higher cross-validation accuracy (without over fitting) and lower value of the objective function.

A. Standard Sequential Algorithm

In the standard sequential version of SGD-based SVM algorithm in Algorithm 1, the weights are initialized with a Gaussian distribution and the training process is done for T iterations. The value for T is decided by prior experiments, where both cross-validation accuracy and the value of the objective function are considered in a such a way that both values come to a stage where their oscillations are in a minimum level. The reason for picking a constant T is for the convenience in timing comparisons and to see how each tuning parameter affects the convergence in the experiments conducted with methods in IV-B and IV-C.

Algorithm 1 Sequential Stochastic Gradient Descent SVM

```
INPUT : [x, y] ∈ S, w ∈ R^d
OUTPUT : w ∈ R^d
procedure SGD(S, w)
    for t = 0 to T do
        for i = 0 to n do
            if (g(w; (x, y)) == 0) then
                ∇J^t = w
            else
                ∇J^t = w − C x_i y_i
            w = w − α∇J^t
    return w
```

B. Model Synchronizing Sequential Algorithm

The model synchronizing sequential algorithm (MSSGD) in Algorithm 2 is important because we are altering the order of updating the weights in the parallel mode with respect to the standard SGD algorithm. Hence, it is important to see the effect from the sequential version of the same algorithm is as same as the parallel version of the algorithm. This way we can identify the accuracy of the implemented algorithm.

In the sequential version of the model synchronizing algorithm, we shuffle the data before creating the small blocks of data resembled S_k in algorithm 2. Each block resembles a one chunk of data which will be processed in the parallel mode in a single process before doing the model synchronization. The block size of unity resembles to the standard SGD-based algorithm. For each data point in each block the weight vector is same unlike in the standard SGD algorithm. In parallel mode if the parallelism is k, and the model synchronization frequency is one, the initial weights of each element processed in each processor is same, implying the sequential version of the algorithm must have the same quality, that is the intuitive idea behind the different block sizes chosen in the sequential algorithm. But, in the sequential algorithm, the model synchronization does not mean that it is doing any communication over the network. Instead, it merely represents the idea that each block will have the same initial weight for each element in a given block. If the block size related to model synchronization frequency in distributed mode is b_1 the corresponding sequential algorithm have a block size of $b_2 = kb_1$.

Algorithm 2 Sequential Model Synchronizing Algorithm

```
INPUT : S ∈ [S_1, S_2..., S_b], w ∈ R^d
OUTPUT : w ∈ R^d
procedure MSSGD(S, w)
    for k = 1 to b do
        procedure SGD(S_k, w)
return w
```

C. Distributed Model Synchronizing Algorithm

In the model synchronizing distributed algorithm in Algorithm 3, the training data set is loaded in a way that equal amount of data is loaded to each machine in order to balance the load among processes. Load balancing is compulsory in order to reduce the process waiting time, which is caused when a subset of processes or a process takes a longer time to complete the computation than the rest. Each machine loads the data and shuffles before the training process. The shuffled data is then processed as blocks in the training process. When the block size defined in the training process is s_b, each machine will run the sequential MSSGD SVM algorithm on s_b number of data points in each process (a block is processed) and the model synchronization is done after processing each block with s_b data points in each of the processes. For each block there will be its own local model, which then communicates with each of the machines using

TABLE I
DATASETS

DataSet	Training Data (60%,80%)	Cross-Validation Data (60%,80%)	Testing Data (60%,80%)	Sparsity	Features
Ijcnn1	21000 ,28000	7000,3500	7000,3500	40.91	22
Webspam	210000,280000	70000,35000	70000,35000	99.9	254
Epsilon	240000,320000	80000,40000	80000,40000	44.9	2000

`MPI_AllReduce`. The average of this is used as the global model after each model synchronization. The global model then becomes the initial weights for the next block. The T training iterations are conducted to reach a convergence.

Algorithm 3 Distributed Model Synchronizing Algorithm

$INPUT : S \in [S_1, S_2..., S_b], w \in R^d, |(j-i)| * K = b$
$OUTPUT : w \in R^d$
procedure DMSSGD(S, w)
    In Parallel in K Machines $[S_1, ..., S_b] \subset S$
    $w_{local} = w$
    for $m = i$ to j **do**
        procedure MSGD(S_m, w_{local})
        $w_{global} =$ **`MPI_AllReduce`**(w_{local})
        $w = w_{global}/K$
return w

V. EXPERIMENTS

In the experiments, we focused on three main sections to analyze how to optimize the training process such that we gain faster execution with higher accuracy. In the first set of experiments, we see how the model synchronization variation affects the convergence of the sequential algorithm and analyze how that information can be used to optimize the distributed version of the algorithm. In the second set of experiments, we change the model synchronization frequency (MSF) and parallelism and observe how convergence can be obtained. In the third set of experiments, we analyze the execution time variation with respect to the variation of MSF for different parallelisms. From these experiments we analyze how an optimized training model can be obtained to guarantee faster execution and higher testing accuracy [20].

Ijcnn1, Webspam and Epsilon are the datasets which are used in the experiments. In Table I, the nature of the data sets is shown. The experiments were conducted in Intel(R) Xeon(R) distributed cluster hosted in Future Systems. For the experiments we use single node core level parallelism for smaller data sets and for the largest data set we use node level parallelism with Infiniband support in OpenMPI. Throughout the experiments we kept the hyper-parameter $C = 1$ in (2) and the learning rate as an adaptive diminishing function as in (4).

A. Model Synchronization Effect on Sequential Algorithm Convergence

The objective of these sequential experiments is to analyze how the variation of model synchronization effect could affect

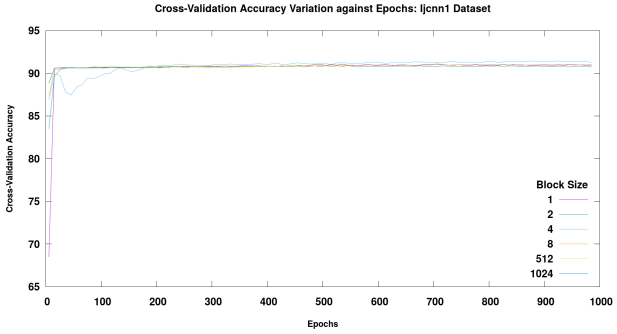


Fig. 1. Cross Validation Accuracy Variation in Sequential Algorithm for Ijcnn1 Dataset : Block Size = [1,2,4,8,512,1024]

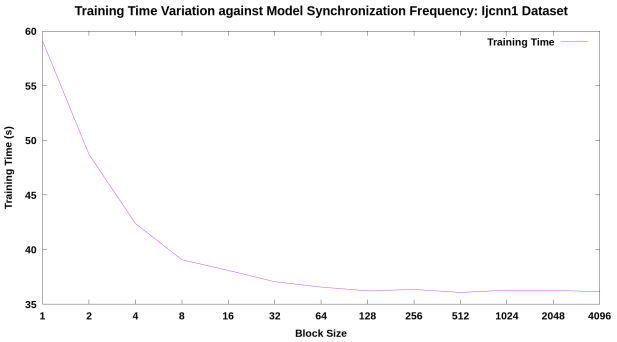


Fig. 2. Training Time Variation with Variable Block Sizes on Ijcnn1 Dataset

the the cross-validation accuracy. [1]

In Ijcnn1, we considered the block sizes 1,2,4,8,1024 and block sizes 1,2,4,8,4096 for Webspam dataset to analyze the effect on cross-validation accuracy. Figures 1 and 3 show experiments on Ijcnn1 and Webspam data sets. The effect from MSF over cross-validation accuracy is approximately negligible in these two data sets. In the experiments, we initialized multiple experiments with unique Gaussian initialization for each experiment. Then we averaged the cross-validation accuracy over multiple experiments with better convergence. From experiments we learned that the convergence of Ijcnn1 data set is highly sensitive on the initialization unlike Webspam data set. With this experiment setting, we were able to get an accurate conclusion on the cross-validation accuracy variation with respect to variable MSF. Figures 2 and 4 shows the training time variation with MSF variation. When the

[1]Block size of b has b data points in the block. The highest MSF occurs when $b = 1$ and b=∞ the lowest MSF occurs. We state MSF = $1/b$, when the block size used is b.

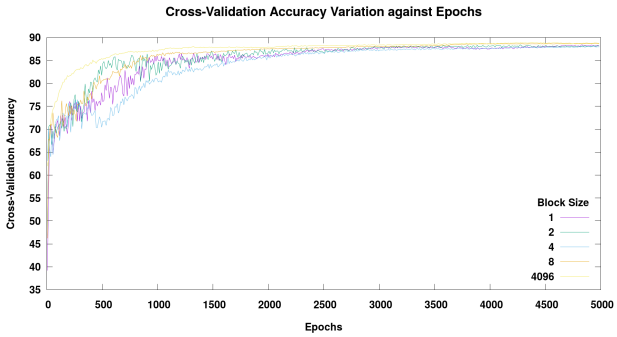


Fig. 3. Cross Validation Accuracy Variation in Sequential Algorithm for Webspam Dataset : Block Size = [1,2,4,8,4096]

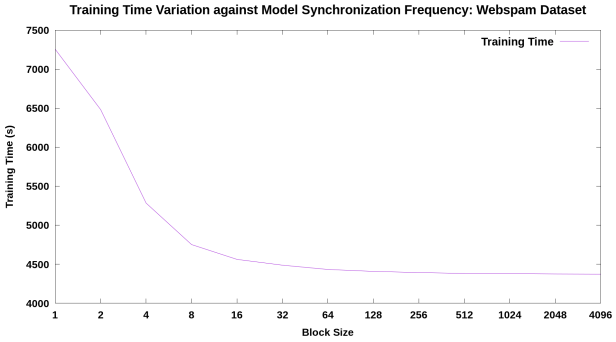


Fig. 4. Training Time Variation with Variable Block Sizes on Webspam Dataset

MSF is high (lower block size), the training time is much higher and the training time dilutes down and becomes steady after a threshold MSF value. This observation implies that with a higher block size (lower MSF) algorithm to run much faster with the same convergence. The reason for time dilution with higher block size comes with the less overhead caused by average model calculation and cross-validation accuracy calculation as they are done when the model synchronization is done. When MSF is lower, the frequency of model averaging and cross-validation accuracy calculation is lower and it provides a performance boost.

B. Model Synchronization Effect on Parallel and Sequential Algorithm Convergence

In understanding how the frequent model synchronization can affect the cross-validation accuracy in sequential version and parallel version, we conducted experiments to see how the cross-validation accuracy behaves in the training period. The main objective of these experiments is to see whether parallel algorithm and sequential algorithm behaves in a similar way towards convergence. Using these experiments, we can verify the approach we used in the experiments are accurate. The comparison of distributed model synchronization along with the replica of it in sequential mode allows us to show that the developed model is functionally accurate. We used Ijcnn1 and Webspam data sets to see how the distributed and sequential

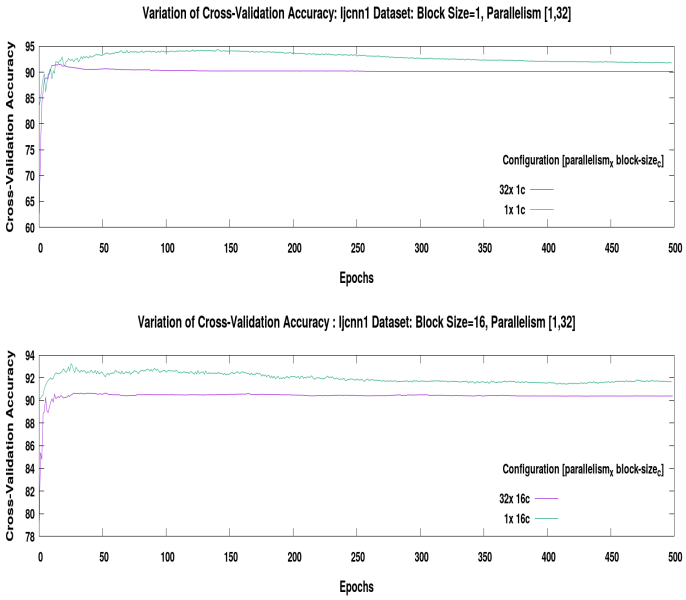


Fig. 5. Cross Validation Accuracy Variation against Block Size with Parallelism : Ijcnn1 Dataset

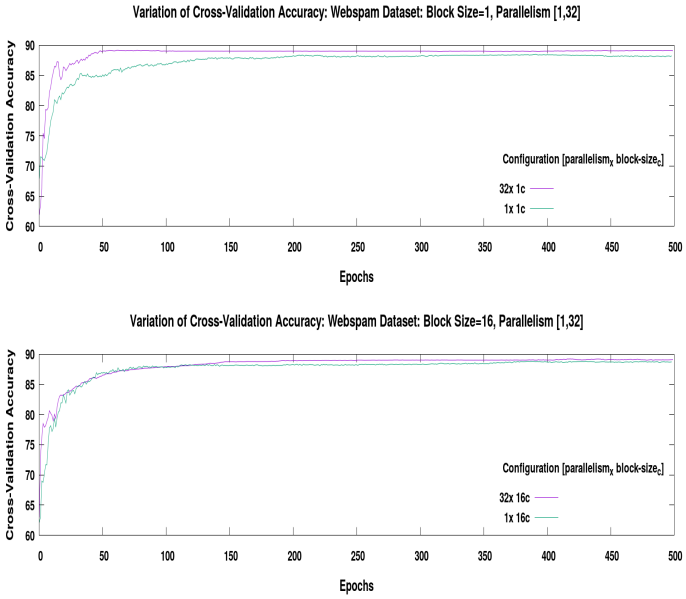


Fig. 6. Cross Validation Accuracy Variation against Block Size with Parallelism : Webspam Dataset

model can provide similar results and the experiments results can be referred from [20] [2]. These results show that distributed model and sequential model provide similar results ensuring that the distributed model functions accurately.

[2]All the experiments carried out to cover 2,4,8,16,32 parallelisms against 1-4096 block sizes for Ijcnn1 and Webspam data sets.

C. Distributed Model Synchronizing Experiments

1) Distributed Model Synchronization on Convergence: In the distributed model synchronizing experiments, we evaluate how the algorithm convergence is affected by the model synchronization frequency along with the variation of the parallelism. We consider three groups of experiments for this, in the first one we consider the high frequency range where it involves 1-8 block sizes. In the second group, 16-512 block sizes for mid range frequencies and for third group, lower frequencies with 512-4096 block sizes were used. The reason behind variable frequency groups comes when we have a data set with a limited data size, we can only pick up to a certain set of frequencies. For instance, in Ijcnn1 data set, the total training data points are 28,000 (80% of data for training) and if we have 32 processes to do the computation, a single process will have only 875 data points, so the maximum block size we can use is 875 and the minimum is 1 and they corresponds to lowest MSF and highest MSF respectively. In figures 7, 8 and 9, the parallel experiments on parallelism 32 for variable MSF is shown. It is clear from these experiments that the variation of cross-validation accuracy and value of the objective function remains approximately same with the denoted MSFs on all three datasets. From these observations, we can understand that frequent model synchronization is not highly vital to obtain a higher cross-validation accuracy or a lower objective function value, irrespective of the nature of the data set. As same as in the sequential experiments, the random Gaussian initialization is vital to identify the best initialization which provides an accurate output. This is the same realization we obtained from the sequential version of the algorithm and by these results, we can verify the parallel model we developed is consistent with the sequential form of the modified SGD-based SVM algorithm. We conducted these experiments for parallelism 2,4,8,16 and 32 and all these results can be seen in [20].

2) Distributed Model Synchronization on Efficient Training: Now we have know that MSF effect on the convergence of the algorithm is relatively independent on faster convergence. The next important factor is to determine how to select a MSF to obtain an efficient training model. In analyzing this fact, first we considered the overall training time needed on variable MSF with variable parallelisms. In figure 11 the training time variation for variable MSFs on 32 MPI processes is shown. It is evident from these results that higher MSF (lower block sizes) has a higher overhead in the training process. This effect dilutes down when the MSF is reduced and dilutes down to a less variable region at the lowest MSFs for each data set. In understanding the reason for the lagging in efficiency for higher MSF region, we conducted the same experiment by doing benchmark on the communication and computation time for variable MSFs on variable parallelisms.

The training time breakdown for Ijcnn1, Webspam and Epsilon data sets for parallelism of 32 is in figure 10. In the higher MSF region (block sizes 1-8) the communication overhead is very high with respect to the lower MSF region (block size beyond 8). It is evident from this experiment that

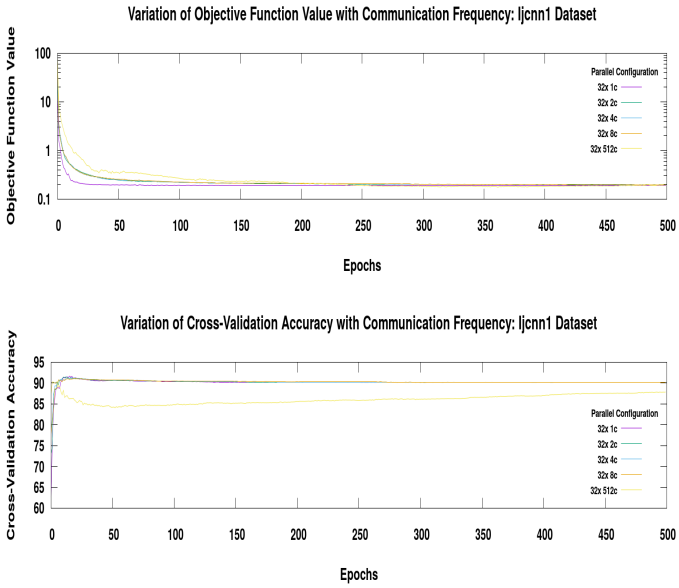


Fig. 7. Distributed Model Synchronizing Algorithm on Ijcnn1 Dataset for Parallelism [32] and Block Size = [1,2,4,8,512]

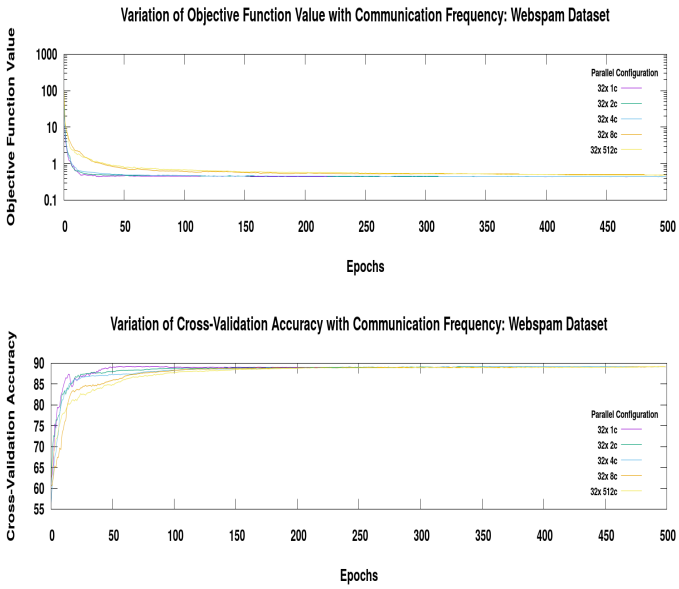


Fig. 8. Distributed Model Synchronizing Algorithm on Webspam Dataset for Parallelism [32] and Block Size = [1,2,4,8,512]

when MSF is higher the total communication overhead is high and it makes the training model to run much slower than that of lower MSF. In considering the total communication overhead, there are two main factors governing the communication delay. The main reason is the MSF value and the second reason is the magnitude of the vector which is communicated over the network or over the processes. Ijcnn1 communicates a vector with 22 dimensions, Webspam a vector of 254 dimensions and Epsilon a vector of 2000 dimensions. It is clear from the time

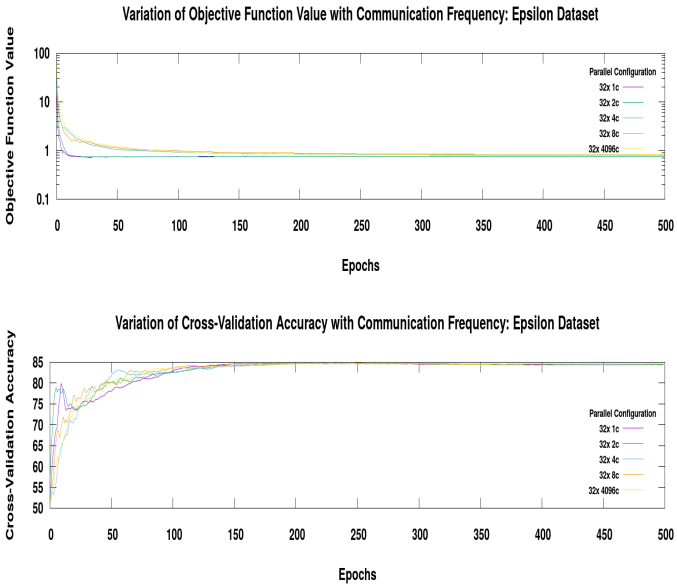


Fig. 9. Distributed Model Synchronizing Algorithm on Epsilon Dataset for Parallelism [32] and Block Size = [1,2,4,8,512]

breakdown experiments that, when the vector shared over the network or processes has a higher dimension, the communication overhead is very high. When training data sets with higher data sizes and higher dimensions the communication overhead is very high. To mitigate this issue in training, we can increase the block size (decreasing MSF) so that minimum amount of communication is done. Along with higher MSF there is an additional computation overhead added in calculating the cross-validation accuracy and objective function value per each communication. The algorithm convergence is decided by the optimum value recorded by each of these variables. So each time we do a communication to synchronize the model, we check the convergence of the algorithm. This computation overhead is proportional to the size of the cross-validation data set. This effect dilutes down to a negligible value when a lower MSF is used. The effect can be clearly seen when the computation time for block size 1 and 2 is analyzed. The computation time for block size 1 is obviously high for all the data sets, but this effect dilutes down as the MSF lowers. But the effect from this computation overhead on efficiency of the algorithm is lesser than that of communication overhead.

3) Distributed Model Synchronization on Prediction Model: The most important fact after training is to obtain a higher testing accuracy from the optimum trained model. It is important to see whether there is an effect from MSF on the testing accuracy for each data set. In figure 12, the testing accuracy variation on variable MSFs for all three datasets is shown with respect to parallelism of 32. It is evident from these results that the testing accuracy is not affected not more than $\pm 0.5\%$ in all three data sets. This observation provides us the final proof we needed to understand the effect from the MSF variation over a highly optimized training model. The

TABLE II
EXPERIMENT SUMMARY : 60% TRAINING DATA

Dataset	Sequential Timing (s)	Parallel Timing (s)	Sequential Accuracy	Parallel Accuracy	Speed Up (x1 vs x32)
Ijcnn1	22.19	1.37	90.63	91.51	16.20
Webspam	2946.49	120.02	87.69	89.12	24.55
Epsilon	22208.07	968.782	80.06	84.36	22.92

MSF effect on prediction model, convergence and efficiency of training model shows that a lower MSF is the best fit to train the SGD-based SVM algorithm to obtain an efficient high accurate model.

Table II shows the summary of the experiments done in this research covering the aspects of improved efficiency, high testing accuracy and better speed up. From these results, it is clear that the MSF optimized distributed model training is a better solution in training larger data sets with SVM.

VI. CONCLUSION AND FUTURE WORK

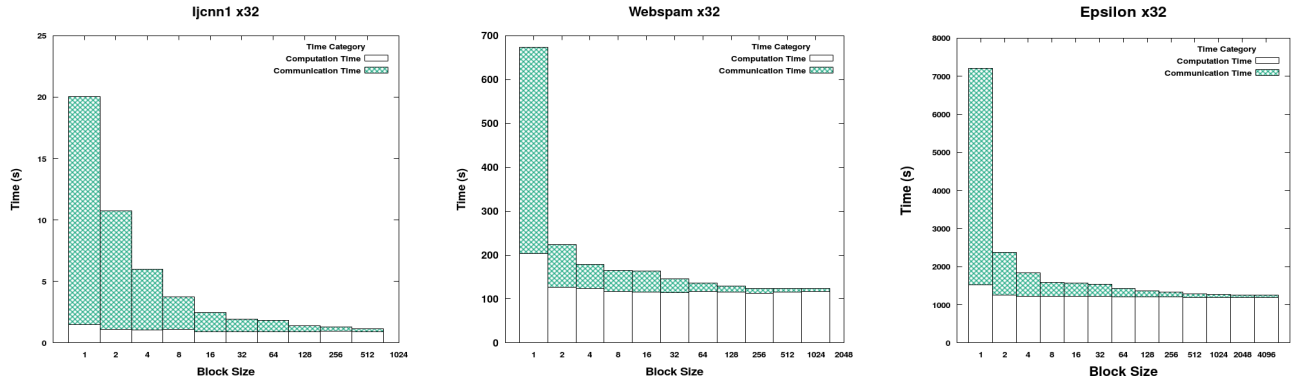
In understanding the distributed scaling of the SGD-based SVM algorithm, it is vital to keep track on the effect of the model synchronization frequency on the convergence of the algorithm. The convergence is governed by the cross-validation accuracy and value of the objective function. Both parameters need to be simultaneously optimized such that the fluctuation of each variable must be the least or satisfy an expected threshold where we call, the algorithm has reached the convergence. The final outcome from the trained model is to guarantee the expected high accuracy in the testing phase. In our research, it is evident that the model synchronization frequency is a key factor as far as fast execution and fast convergence is considered. Our research also shows that high MSF value is loosely coupled with the convergence of the training model, test accuracy and efficient execution. This allows us to use the MSF as lower as possible to obtain an efficient training model with high accuracy. The efficiency of the training model is highly valuable when training large data sets with higher dimensions and this is evident with the observations we made on Ijcnn1, Webspam and Epsilion data sets. With reference to standard SGD-based sequential SVM algorithm, our distributed model provides higher accuracy for all three data sets with less over-fitting on training data. This shows that our model synchronization optimization is better in gaining a higher testing accuracy with efficient execution.

We plan to expand this model on training high volume data in edge devices in sensory networks to obtain high efficient training with limited resources. In addition to that we plan to work on implementing a streaming model to support high volume data streams in cloud environments.

ACKNOWLEDGMENT

This work was partially supported by NSF CIF21 DIBBS 1443054 and we extend our gratitude to the FutureSystems team and Digital Science Center for their support with the infrastructure.

Fig. 10. Training Time Breakdown in Data sets for Parallelisms 32, 80% Training Set



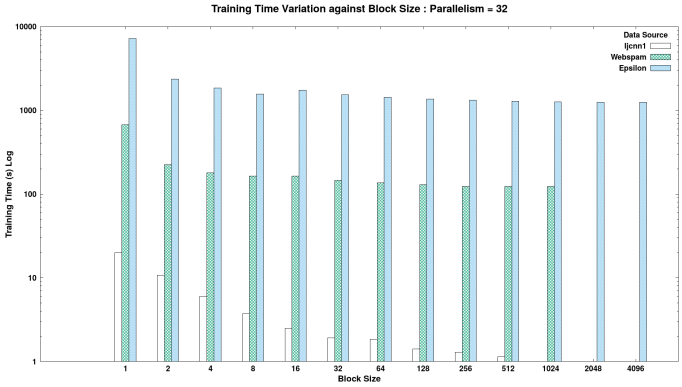


Fig. 11. Distributed Training Time Variation: Parallelism = 32, 80% Training Set

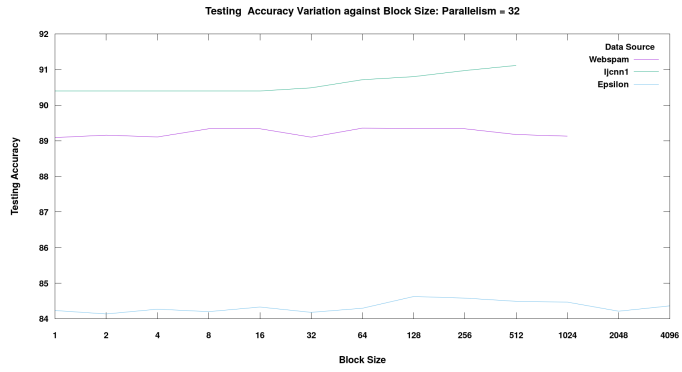


Fig. 12. Distributed Testing Accuracy Variation with Block Size: Parallelism 32

REFERENCES

[1] Zhang, Kai, et al. "Scaling up kernel svm on limited resources: A low-rank linearization approach." Artificial intelligence and statistics. 2012.

[2] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. Mach. Learn. 20, 3 (September 1995), 273-297iu-psgdsvmc.

[3] Cho-Jui Hsieh, Si Si, Inderjit S. Dhillon, A Divide-and-Conquer Solver for Kernel Support Vector Machines, Article:CoRR, eprint. 1311.0914,(2013).

[4] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. 2, 3, Article 27 (May 2011), 27 pages.

[5] John C. Platt. 1999. Fast training of support vector machines using sequential minimal optimization. In Advances in kernel methods, Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola (Eds.). MIT Press, Cambridge, MA, USA 185-208.

[6] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. Journal of Machine Learning Research, 6:1889–1918, 2005.

[7] Yang, Jifei and Ye, C.-Z and Quan, Y and Chen, Ni-Yun. (2004). Simplified SMO algorithm for support vector regression. 33. 533-537.

[8] Machine Learning Course, The Simplified SMO Algorithm, North Eastern University, College of Computer and Information Science

[9] Kim HC, Pang S, Je HM, Kim D, Bang SY. Constructing support vector machine ensemble. Pattern recognition. 2003 Dec 1;36(12):2757-67.

[10] Cao, Li Juan, et al. "Parallel sequential minimal optimization for the training of support vector machines." IEEE Trans. Neural Networks 17.4 (2006): 1039-1049.

[11] Edward Y. Chang, Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, and Hang Cui. 2007. PSVM: parallelizing support vector machines on distributed computers. In Proceedings of the 20th International Conference on Neural Information Processing Systems (NIPS'07), J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis (Eds.). Curran Associates Inc., USA, 257-264.

[12] Chang, Edward Y. "Psvm: Parallelizing support vector machines on distributed computers." Foundations of Large-Scale Multimedia Information Management and Retrieval. Springer, Berlin, Heidelberg, 2011. 213-230.

[13] Zinkevich, Martin, et al. "Parallelized stochastic gradient descent." Advances in neural information processing systems. 2010.

[14] A. Z. Zeyuan, C. Weizhu, W. Gang, Z. Chenguang and C. Zheng, "P-packSVM: Parallel Primal grAdient desCent Kernel SVM," 2009 Ninth IEEE International Conference on Data Mining, Miami, FL, 2009, pp. 677-686. doi: 10.1109/ICDM.2009.29

[15] Y. Lin et al., "Large-scale image classification: Fast feature extraction and SVM training," CVPR 2011, Colorado Springs, CO, USA, 2011, pp. 1689-1696. doi: 10.1109/CVPR.2011.5995477

[16] Shalev-Shwartz, Shai, et al. "Pegasos: Primal estimated sub-gradient solver for svm." Mathematical programming 127.1 (2011): 3-30.

[17] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. An improved GLMNET for l1-regularized logistic regression. Journal of Machine Learning Research, 13:1999-2030, 2012.

[18] Danil Prokhorov. IJCNN 2001 neural network competition. Slide presentation in IJCNN'01, Ford Research Laboratory, 2001.

[19] De Wang, Danesh Irani, and Calton Pu. "Evolutionary Study of Web Spam: Webb Spam Corpus 2011 versus Webb Spam Corpus 2006". In Proc. of 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2012). Pittsburgh, Pennsylvania, United States, October 2012.

[20] Vibhatha Abeykoon, Geoffrecy Fox. "Parallel Stochastic Gradient Descent based Experiment Web Results", Digital Science Center, Indiana University Bloomington, United States, February 2019. [Online]. Available: https://iu.app.box.com/s/eqauk3bknywifdmkao6wpp4gp36o2es9. [Accessed: 22-Feb-2019]