Harp SVM: a Parallel SVM Algorithm Based on Hadoop Using Harp

Yiming Zou
Indiana University Bloomington
107 S. Indiana Avenue
Bloomington, IN 47405-7000
yizou@iu.edu

Yueqi Tan
Indiana University Bloomington
107 S. Indiana Avenue
Bloomington, IN 47405-7000
yueqtan@umail.iu.edu

ABSTRACT

Support vector machines(SVM) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Harp is good for SVM implementation but it needs to overcome synchronizing the global support vectors. In our experiment, we work on the Iterative SVM version implemented on Hadoop. In each iteration, each machine computes the support vectors and does an all-reduce to collective the whole global support vector set and treat it as the extra training data. From the result we can see that with larger number of mappers, the number of support vectors has greater gradient decent. Less iterations are needed to reach the final result which however remains the same. The speed up time of Hadoop-SVM and Harp-SVM both has an logit-linear acceleration along with the number of mappers. Harp performance is better than Hadoop since it reduces I/Os of communication.

1. INTRODUCTION

Machine learning is one of the most powerful areas we are interested in. It helps to solve numerous problems and are widely used in both academy and industry. There are a certain of machine learning algorithms that probably most of us are all familiar with, such as K-means, support vector machine[17], neural networks[10] and so on. Each time we implement these algorithms, there is always the question to ask, how can we do better? Even with the same algorithm, the runtime can varies a lot depending on the method of implementation. Distributed systems[18] pop up in late 20th century and thus give us a great way to answer the question. Distributed system is a model whose components located on networked computers that communicate and coordinate their actions by passing messages. From the lecture in class, we have a more easy-understanding concept of it: a distributed system is a collection of independent computers that appears to its users as a single coherent system. With a distributed system, we can solve computational problem by ease, since we can allocate the jobs to multiple machines, and thus largely reduces the runtime. Distributed computing is also great for iterative problems such as PageRank[16]. MapReduce[3] is a programming model and an associated implementation to process and generate large data sets. We have a users specify "map" function and a "reduce" function. The map function processes a key/value pair to generate a set of intermediate key/value pairs, and the reduce function that merges all intermediate values associated with the same intermediate key. We write programs in this functional style so they are automatically parallelized and can be executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning, scheduling, handling failures, and managing communication. This greatly benefits programmer who has little experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

The High Performance Computing(HPC)[4] and Grid Computing[6] are for large-scale data processing by using Application Program Interfaces(APIs) as Message Passing Interface(MPI)[8]. MapReduce helps to accelerate data access by collocating the data with the compute node. Therefore we have MapReduce implementations to control data locality. However, MapReduce only operates at the higher level. MPI gives control to the programmer and handles the mechanics of the data flow.

When dealing with large scale data set, we can choose to divide the work into many tasks, and give the task to one or more machines. When the computation is simple and uses little time to get the job done, we can only run the work on a single machine. When the computation is large and involving complex algorithm or iterations, the work is "cooperated" by the coherent system of multiple machines. Hence we have the necessarily of communication among all machines, for they need to know the current state of the on-doing job and receive commands from the master. Each single machine communicate with others by message passing.

Message passing sends a message to a process relying on the process as well as the supporting infrastructure to select and invoke the actual code to run. When we analyze the runtime of an algorithm, we also need to take into consideration of the communication cost. There are typically 2 communications of one node in a communication round: receiving messages from the neighbors in the parallel, sending new messages to its neighbors after the local computation. Let N be the number of rounds, then the total times of communication of a node is $2N$. This can be very large when the number of nodes is considerable. Here we choose another way of communication called broadcast. It sends the updated message from the master node to all slaver nodes at the beginning of each round, and receive messages from all nodes at the end of a round of computation. This is much more convenient and saves I/Os.

With all the features above, there are many excellent framework for distributed applications. One of the most promising one is Apache Hadoop[20], which is also our focus of our course. Apache Hadoop is a software framework for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. The reason we love it correspond with its reliability, high efficiency and fault tolerance. Also, it is open-source and can be build on commercial machines, thus make it easy-use for both personal users and companies. It is designed for high-throughput applications, so it is very suitable for distributed computation.

Harp[24] is a plugin to Hadoop. It provides collective communication library as well as associated data abstraction. By plugging Harp into Hadoop, we can convert MapReduce model into Map-Collective model and enable efficient in-memory communication. In this way we have parallel processes cooperate through collective communication for efficient data processing. What's more, Harp is neither a replication of MPI nor an attempt to transplant MPI into the Hadoop system.

It is highly recommended when we want to run iterative algorithms because of its high speed. For most of the algorithms, we can implement them simply on Hadoop. However, the reason we use Harp is that Harp's communication is based on memory while Hadoop is based on disk. Too many I/Os will largely reduce the performance. This results in less of runtime. Take PageRank as an example, Harp is much more faster than Hadoop on different experiment settings. We also has result from experiment on Pig showing Harp provides fast data caching and customized communication patterns among iterations[21]. Since Hadoop and Apache open-source stacks are designed as the mainstream tools for handling big data problems, the development of a Hadoop plug-in to support Iterative MapReduce can lead to best performance.

Among of certain algorithms in machine learning, here we choose support vector machines for implementation on Harp. Support vector machines(SVM) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a training data set, an SVM training algorithm divide the datas into several parts(mostly two) which each projects to a class it belongs. It can not only solve linear problems in machine learning but also polynomial ones. An SVM model represents every data point as a point in space, and maps them onto a plate of a new dimension. After one or several mappings, the points can finally be divided by a clear gap that is as wide as possible. For each mapping, SVM uses a mapping function. By reversing all the functions, we can go back from the single line(the dividing gap) to a curved surface that divides the original data set.

Related Work

When training a support vector machine, we need to find out the bound constraints and a linear equality constraint. This quadratic optimization problem may goes into many issues we need to consider in designing an SVM model[11]. When the training set is large and the learning tasks are considerable, time and memory are big limitations. At worst time, SVMs scale badly with large data size due to the quadratic optimization algorithm and the kernel transformation[15]. SVM can also be very sensitive to the chosen parameter and noise. Making the correct choice of kernel parameters is crucial for obtaining good results, which also means that an extensive search must be conducted on the parameter space before results can be trusted, and this often complicates the task. The current implementation is only optimized for the radial basis function kernel, which might still be suboptimal for the data.

Harp is good for SVM implementation but it needs to overcome the huge amount of memory used and the computation complexity. Cascade SVM[7] was proposed right for this challenge. Dataset is split into parts in feature space in this method. For each sub-dataset, the non-support vectors are filtered and only the support vectors are transmitted to the next round. Therefore, the margin optimization process need only the combined sub-dataset and can thus find out the support vectors. Another method proposed for parallel SVM training is that, for each subset of a dataset, it is trained with SVM and then the classifiers are combined into a final single classifier function. We also see propose in distributed SVM based on strongly connected network[14]. In this way, a dataset is split into equal parts for each computer in a network. Then the support vectors are exchanged among the related computers. Upon the use of subSVM, there is an interesting paper develops a parallel SVM model based on MapReduce. As well as methods used before, training samples are divided into subsections, but each subsection is trained with a SVM model and libSVM[2] is used to train each subSVM. Then the non-support vectors will be filtered with subSVMs, and the support vectors of each subSVM are taken as the input of next layer subSVM. Finally, the global SVM model can be obtained through iterations and the output is obtained.

2. SUPPORT VECTOR MACHINE

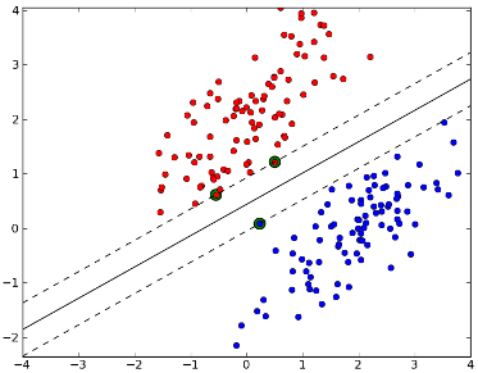

Figure 1: Finding the maximum gap hyperplane in SVM algorithm

Machining learning algorithms range from original K-means, Latent Dirichlet Allocation[1] to more complex ones such as Support Vector Machine, Neural Networks and Random Forest[13]. These algorithms deal with classification or regression problems of all kinds. When choosing an algorithm to apply to a specific problem, we need to take a certain factors into consideration. Time complexity is one of the most limiting factors. When learning a large scale dataset, we of-

ten face the problem of runtime limitation. Support Vector Machine(SVM) is an important algorithm universally used is classification. It has high performance when working on certain training data and is good for simplifying polynomial problem into linear problem. In SVM algorithm, the computation time depends on large scale kernel matrix. Lots of effort have been made to optimize the runtime of SVM models.

One effective way is to reduce the number of selected features, called feature selection. Usually within a dataset there is only part of the data that we need to complete a classification. Many features are not that important in the final result and do not count for the classification. We only need to manually pick up the dependent ones regardless of the others which may take up a lot of space. This is a basic approach to reduce feature vector size[19]. However, there may be sometime that we do not know if one feature weights over another and can not manually omit some of the features, or the feature numbers are large so selection can be lots of work. Feature subsets can then solve the problem by applying various algorithm. The most widely used algorithms are information gain[12], , Gini[22] and correlation based feature selection[9].

Aside of modifying the dataset, we can also reduce time of I/O operations caused by communication. Within an iteration of SVM computation, the information of each round needs to be updated and therefore caused read and write operations. Harp is a plugin to Hadoop which specifically deal with this problem.

3. HARP

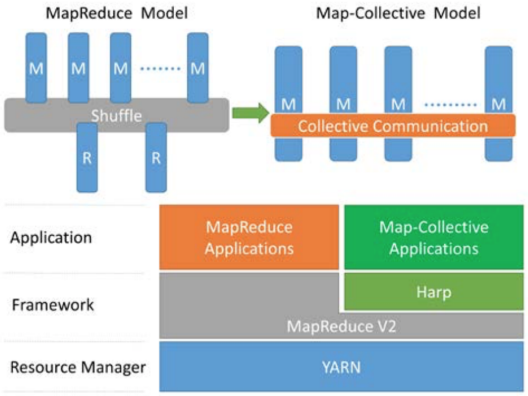

Figure 2: Parallelism and Architecture of Harp

Harp provides collective communication library to Hadoop. It also provides associated data abstraction. Based on Hadoop, we can convert MapReduce model into Map-Collective model with the use of Harp. It is also possible to enable efficient in-memory communication, which is one of the greatest advantage Harp produces. When running iterative algorithms, runtime depends largely on communication operations arise by information updating of each loop. Harp is therefore highly recommended because of its high speed of extracting data from cache. There is no doubt that implementing machine learning algorithms on Hadoop is an efficiency and high performance way. However, Harp works especially well on iterative models since its communication is based on memory while Hadoop is based on disk. Too many I/Os will largely increase runtime. With the implementation of Harp, we can reach high performance of algorithms and reduce runtime. Upon this, we have parallel processes cooperate through collective communication for efficient data processing.

Benefits from Harp are plenty. When doing communication in MapReduce, the "reduce-gather-broadcast" strategy is applied through on-memory communication in frameworks such as Twister[5] and Spark[23]. However, when the data set is large and so do the centroids scales, using "gather-broadcast" is no longer an efficient way. When the work loads on iterative algorithms, communication execution time is even more vital to our consideration. For each iteration, all machines need to communication with other nodes, or by simplification, with the master node. This gives out at least O(MN) operations where M is the number of machines and N is the number of iterations. By then the communication performance contributes lot to the total performance. "Collective Communication" in MPI is a good way which can abstract the communication layer out and therefore provide collective communication abstractions. Yet it has limitation supporting high level data abstractions and is not transparency for users. In order to improve this, we use Harp library. We can explain how Harp works as following. Harp works an optimized implementation to provide data abstractions as well as their related communication abstractions. By plugging Harp into Hadoop, we can then convert MapReduce model to Map-Collective model. This enables efficient in-memory communication between Mapper nodes whose tasks can therefore convert messages and finally work across various important data analysis applications.

4. ITERATIVE SVM

The Iterative SVM algorithm works as Figure 3. The training set of the algorithm is split into subsets. Each process classifies sub dataset locally via SVM algorithm from LibSVM and gets the support vectors, and then passes the calculated support vectors to global support vectors to merge them. In Map stage of the Harp job, the subset of training set is combined with global support vectors first, and after get the new support vectors, the processes do an all-reduce to broadcast the global support vectors. And finally they can continue working on next round.

Algorithm 1 Iterative SVM algorithm

Require: *dataPoints*
Ensure: *globalSV*
  split *dataPoints* into n pieces
  each process gets its own piece
  $globalSV = \{\}$
  for $iteration = 1 \rightarrow n$ **do**
    combine *subDataPoints* and *globalSV*
    train data set
    allreduce SV to *globalSV*
  end for
  if Master **then**
    output *globalSV*
  end if

The pseudo code can be found in Algorithm 1.

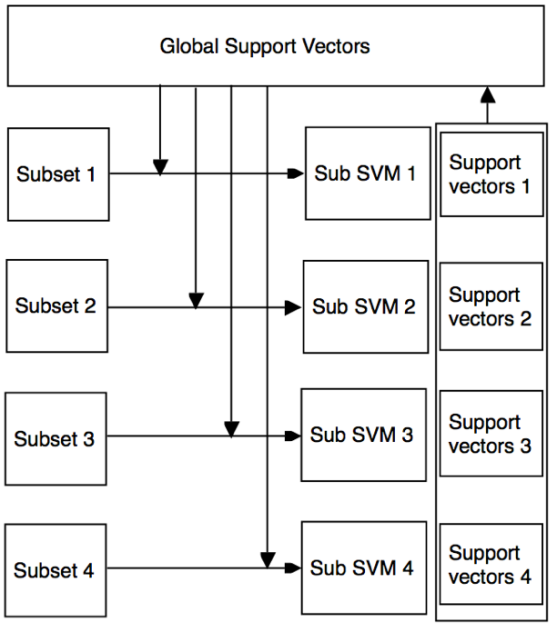

Figure 3: Procedure of paralleling Iterative SVM algorithm

5. EXPERIMENTS

5.1 Data Sets

Fourclass. This is a very small data sets we use for test the implementation. It has 862 data points which belong to two classes. Each data point has only two features.

vehicle. The vehicle data set has more data points which is 846 with 18 features. It's combined by 4 classes.

MNIST. This data set is a huge one with 8100000 data points in which are 784 features representing to the pixels. There are 10 classes from 0 to 9. We repeat to choose a very small part randomly from this data set and compute support vectors.

5.2 Experimental Setup

Experiments are done on a cluster with Intel Haswell architecture. We use 2 node each with two 12-core 24-thread Xeon E5-2670 processors. All the nodes have 128GB memory and are connected with 1Gbps Ethernet (eth) and Infiniband (ib).

5.3 Results

From figure 4 to figure 6 we can see the support vectors' set becomes stable with the increase of the iteration which means that the parallel algorithm works well on training part of data and then merge the support vectors gain by the SVM algorithm. And the more mappers we use, the faster the support vectors' set becomes stable.

While, if we choose to compute by more mappers, the global support vectors at the very beginning is usually larger. That's reasonable because for small data sets, the shape of the clusters can vary from others which means that the support vectors can be very different. So after doing all-reduce, the

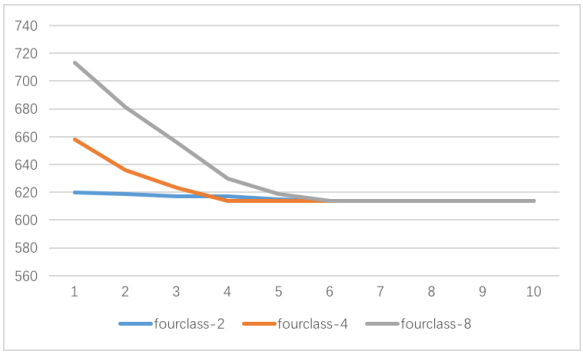

Figure 4: The number of support vectors versus the number of iteration on fourclass data set

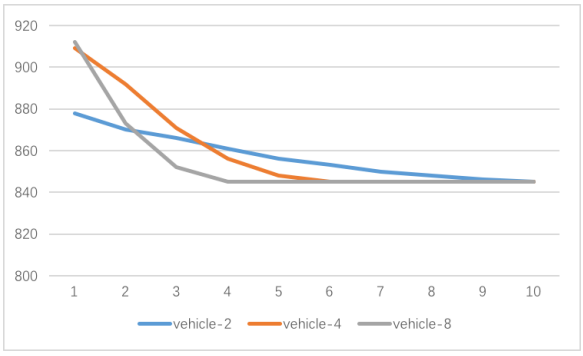

Figure 5: The number of support vectors versus the number of iteration on vehicle data set

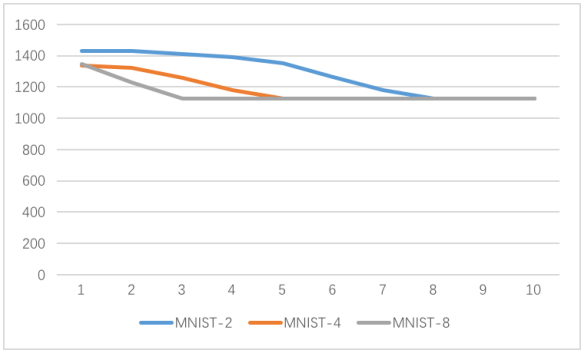

Figure 6: The number of support vectors versus the number of iteration on MNIST data set

global vector sets will be larger.

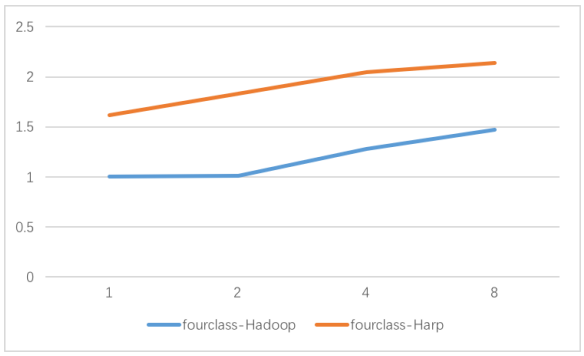

Figure 7: Speedup versus the number of mappers on fourclass data set

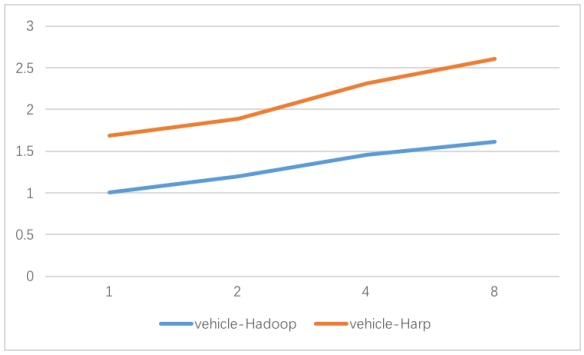

Figure 8: Speedup versus the number of iteration on vehicle data set

Figure 7 to figure 9 shows that the performance of using Harp is better than only using Hadoop. In the small data sets like fourclass and vehicle, we can see the performance improves at least 40% and in the MNIST data set, the performance is better 80% than the Hadoop implementation. The most saving execution time is the I/Os' time. Due to Harp can use all-reduce to synchronize data only through the cache, it will reduce much file reading and writing time which is the indispensable part in Hadoop.

6. CONCLUSION

We implement the iterative SVM algorithm based on Hadoop with the plug-in Harp. The above result has given us sufficient proof of preference using Harp to implementing SVM algorithms over other method. By using Hadoop, we fully benefit from the characteristics of distributed computing. As for multiple iterations, Harp gives us inspiration of how to communicate among the nodes with least operations. If not the best, it is a great way to balance between getting the least of necessary communication and the most of important information.

Aside of runtime, we are also concerned about the efficiency. Our result has shown the Harp implementation uses less execution time than simply Hadoop, while not sacrificing the accuracy. From the result of our implementation, we can see that support vector machine models have better performance on Hadoop by adding on the plugin Harp. The performance is optimized by 80% of MNIST-Harp compared to MNIST-Hadoop. Hadoop generate read and write operations at each iteration while Harp stores support vectors in cache so the runtime is largely reduced. Finally, for each model we can find a global support vector set which is stable and therefore represent the output result. This is easy to understand, since Harp targets to improve the communication process in distributed computing. Moreover, we can therefore apply the presumption that Harp works better on iterative distributed computing to all related algorithms. Experiments need to be done in order to give absolute proof, though. However, for other non-iterative algorithms, we may still prefer Hadoop for its simplicity.

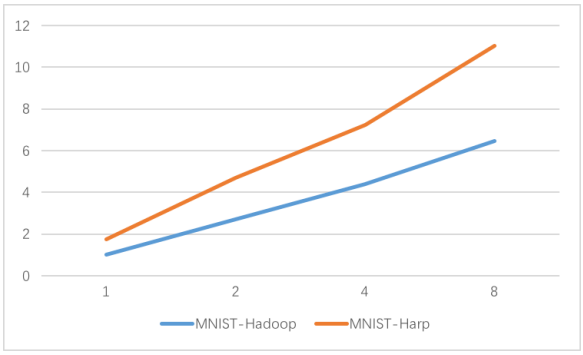

Figure 9: Speedup versus the number of iteration on MNIST data set

Acknowledgements

Thanks to Professor Judy Qiu for giving us insightful lectures on distributed systems. Thanks to Professor Bo Peng for the deeply discussion on SVM algorithms. Thanks to all associate instructors for help on questions.

7. REFERENCES

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[2] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[3] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[4] K. Dowd. *High performance computing*. O'Reilly & Associates, Inc., 1993.

[5] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.

[6] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[7] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade svm. In *Advances in neural information processing systems*, pages 521–528, 2004.

[8] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[9] M. A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[10] S. Haykin and N. Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.

[11] T. Joachims. Making large scale svm learning practical. Technical report, Universität Dortmund, 1999.

[12] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[13] A. Liaw and M. Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[14] Y. Lu, V. Roychowdhury, and L. Vandenberghe. Distributed parallel support vector machines in strongly connected networks. *IEEE Transactions on Neural Networks*, 19(7):1167–1178, 2008.

[15] D. Meyer and F. T. Wien. Support vector machines. *The Interface to libsvm in package e1071*, 2015.

[16] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[17] J. A. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[18] A. S. Tanenbaum and M. Van Steen. *Distributed systems*. Prentice-Hall, 2007.

[19] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for svms. 2000.

[20] T. White. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[21] T.-L. Wu, A. Koppula, and J. Qiu. Integrating pig with harp to support iterative applications with fast cache and customized communication. In *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds*, pages 33–39. IEEE Press, 2014.

[22] S. Yitzhaki. Relative deprivation and the gini coefficient. *The quarterly journal of economics*, pages 321–324, 1979.

[23] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010.

[24] B. Zhang, Y. Ruan, and J. Qiu. Harp: Collective communication on hadoop. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pages 228–233. IEEE, 2015.