Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Seminar

Protocol for a Systematic Literature Review on Scaling up Machine Learning

Author: Sara Salimzadeh (2614810,12021520)

1st supervisor: Dr. Adam Belloum

A literature study is submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

February 8, 2019

Abstract

Cutting edge applications in the field of machine learning and deep learning are prompted by the availability of large-scale datasets and modern computational hardware like many-core CPU and GPU. To automate the analysis of data and use such large-scale datasets in the learning process, a new field of machine learning emerges: *large-scale learning*. It intends to develop efficient and scalable algorithms to handle this amount of data and gain accurate results. Efficiency could be defined in terms of memory usage, processing time, communication and number of required computational units. One approach to applying *large-scale learning* is distributed learning. As most of the datasets are distributed across different physical locations and scaling up machine learning led to apply learning process in many data centers and clusters, a *distributed learning* method seems appropriate. This report provides a broad overview of challenges and issues of deep learning and machine learning applications at large scale. Then, it covers few of recent technical contributions namely frameworks, systems and algorithms have made over the recent decade in these domains.

Keywords: Machine Learning, Large-scale Learning, Distributed Learning, Scalability

Contents

List of Figures

1

Introduction

The rapid growth of data especially in the recent decade opens the way for new applications in the domain of machine learning which led to emerging a new field in this area: *large-scale learning* (15). *Large-scale learning* has received considerable attention from both academia and industry. In order to find out solutions for real use cases, data analysis must be automated; since it is beyond the ability of human experts to handle such large-scale data sets. As most of the datasets are naturally distributed and assigning learning process to multiple workstations is a perceptible method to scale up machine learning tasks, many research projects focus on *distributed learning* (15).
Distributed and *large-scale learning* bring new challenges regarding scalability and efficiency of algorithms associated with computational and memory resources. In the literature study, many techniques including frameworks, architectures, algorithms, and optimization are proposed to address these challenges.

Deep learning as a class of machine learning has a significant influence on recent achievements in the artificial intelligence area. This approach was investigated since the 1980s. However, the lack of sufficient training data and slow hardware limited the use of deep learning for large scale problems. Considering the availability of a large amount of data and modern hardware architectures, it is the time to take advantage of deep learning in large-scale problems.

The systematic literature review aims to describe new challenges and issues related to this topic. As real-world applications both in academia and industry engage in deep learning and machine learning domain, this study should help them classify the available techniques and tools in order to detect suitable options for their use cases. Furthermore, distinguishing the limitations and gaps and also possible future work give researchers opportunities to find out the current research line and let them contribute technically in these fields.

1.1 Research Questions

To clarify the direction of this literature review, research questions must be determined. This current literature study targets researchers, practitioners and fellow students as a reader.

RQ1: What are the new challenges facing the machine learning application at large-scale?

Motivation: The common algorithms of machine learning are designed to deal with small or medium size datasets. To apply such algorithms on large datasets, new issues appear. These issues are related to the following topics: optimization and update of algorithms, storage, computational units, process time and so on.

RQ2: How researchers/practitioners address the challenges facing machine learning applications and what are their solutions?

Motivation: To provide deep consideration, we review some of the state-of-the-art solutions for real-world applications and their evolution trends covering the period 2010 to 2018.

RQ3: What are the limitations and possible future work in this domain?

Motivation: Limitations and mentioned future work of papers can determine the research line. Being aware of the direction of researches and cutting edge techniques, one can make a technical contribution and introduce new solutions for real-world applications.

In section 2, we discuss methodology applied in this literature study followed by the result. Section 4 concludes based on the findings.

2

Methodology

According to Dyba et al. (7), systematic reviews on a topic consists of three major steps. Figure 2.1 depicted the schema in details.

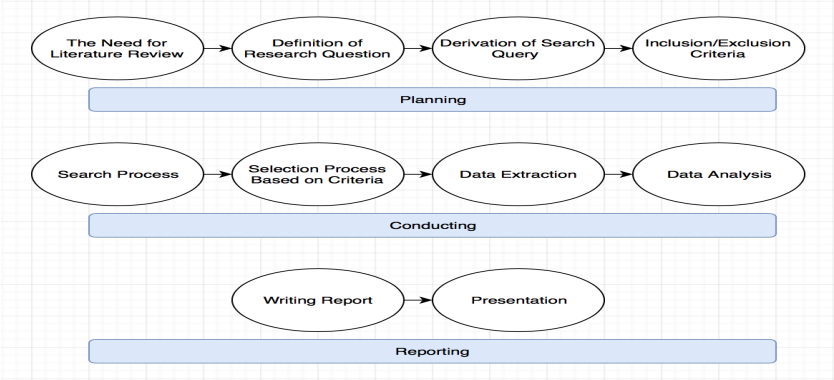

Figure 2.1: Pipeline of Systematic Literature Review: we follow this pipeline to conduct the current literature study.

1. Planning: In this step, the need for doing the literature review is clarified. Based on research questions, we extract keywords and then search queries. Before initiating the searching process, it is necessary to define some rules to add or filter out a paper. In other words, we should determine the inclusion/exclusion criteria.

2. Conducting: the searching process is the first step of conducting the review. Primary studies are selected papers fulfilling all inclusion criteria and none of the exclusion criteria. A typical study should cover 30-150 cases and finally, target 15-20 of most relevant ones as primary studies. There are many resources to select papers. Google Scholar, ACM digital library, IEEE explore, Springer link and Elsevier Scopus can be used as a search engine. Afterward, we apply data extraction and data analysis

on primary studies. Classification of extracted data can help to answer the research question accurately covering the period 2010 to 2018.

3. Reporting: In the last phase, the most important part of the current literature review, we prepare the final report and presentation.

Further details of the methodology are explained in th Appendix.

3

Result

3.1 New Challenges

Before diving into the explanation of this question, it is useful to discuss the rationales for partitioning data to different locations instead of centralizing them. Data sets can be distributed in three ways (15):

- Horizontally: Subsets of instances stored at different sites
- Vertically: Subsets of attributes of instances stored at different locations.
- Mixed: Includes vertically and horizontally

Classical machine learning algorithms are designed to learn from a unique dataset. To apply these algorithms, it is required to collect all the data in one center. However, this is not effective and feasible because of (15):

- The great cost of storing in a central database
- The great computational cost of mining central database
- Transferring huge data volumes over the network
- Privacy and sensitivity issue of data
- More latency to end user
- Communication intensity

These limitations led to the emerging of distributed learning. With the help of distributed learning, it is possible to train several classifiers to increase the accuracy of the final result. To combine the information, there are two approaches with different issues. One way is to combine the classifiers. To do so, it is essential to define a standard representation to encapsulate all other representations. This procedure leads to information loss. The second approach is based on merging the predictions of classifiers which is less severe than the first method. In this case, we just need to convert numerical to categorical values. A great majority of learning algorithms in the literature focus on the second method. Here, we just name some of the most popular distributed algorithms. To learn more, look into (15). Algorithms are 1.Decision Rules, 2.Stacked Generalization, 3.Meta-learning, 4.Knowledge Probing, 5.Distributed Pasting Votes, 6.Effective Stacking, 7.Distributed Boosting, 8.Distributed Clustering, 9.Effective Voting.

Regardless of the specific algorithm, the accuracy of heterogeneity of data among various partitions and preserving the privacy can be mentioned as new issues of distributed learning (15).

Note that although centralizing data at one site seems inefficient in most cases, there is a rule of thumb that can help practitioners to decide the appropriate method. If the total size of the data is more than 2-3 orders of magnitude greater than the dimensionality d, it is always better to apply distributed learning (4).

In addition to data partitioning to multiple workers, we can exploit model parallelism to parallelize training process as well. Data-parallel algorithm concurrently computes a partial update of all model parameters in each worker, then aggregates these partial updates to obtain a global estimate of model parameters. In contrast, the model-parallel algorithm concurrently update a subset of parameters on each worker(using either all data or different subsets of data). In the data-parallel technique, it is required that every worker has full access to all global parameters. However, in order to apply the model-parallel method, the trade off between memory efficiency, scheduling efficiency and intrinsic dependencies within models should be managed. Furthermore, as parameters are divided up, workers only need to access a fraction of parameters at a given time (12).

In the class of data parallel models, two subclasses exist: synchronous and asynchronous type. In synchronous type, there is a step Allreduce communication among different workers to exchange related parameters. On the other hand, the asynchronous type uses special workers as parameter servers. Parameter servers are responsible to control model parameters (1).

Large-scale ML problems also encounter a big model challenge in which models with millions or even billions of parameters/values must be estimated (12).

Looking into the performance of machine learning applications, It is crucial to fit the data into distributed main memory (8). General-purpose compression techniques can't provide both good compression ratio and fast decompression. Therefore, the need for new techniques arises here. Moreover, different kinds of noises can degrade the performance of distributed learning such as straggler nodes, system failures, communication bottlenecks (11).

One of the challenging processes in predictive applications is identifying an appropriate model. This search process is usually manual and done in a sequential greedy method which is time consuming (16).

Focusing on deep learning, we first should define it. Deep learning covers these areas: Advances of algorithms, large-scale data, high computing powers. In order to do the prediction, three steps are carried out.

- Forward Computation
- Backward Computation
- Optimization

To tackle unsolved challenges and making deep learning faster and bigger, GPUs are utilized. Parallelization methods of deep learning are similar to other algorithms. Data parallel method is more extensible and faster. Additionally, there is no need to modify algorithms significantly. To compare the synchronous and asynchronous type, experimental results show that asynchronous is less stable regarding convergence while faster convergence occurs in the synchronous model (1).

Centering on an industry, it is required to have a system as a part of a service capable of doing predictions on demand on interactive timescales. Using batch-oriented design such as Spark incurs latency and not suitable for these perquisites (6).

3.2 Solution for New Challenges

Here, we briefly target some of the recent systems and algorithms implemented in this domain. The order of systems is based on the time the corresponding publications were published, figure 3.1.

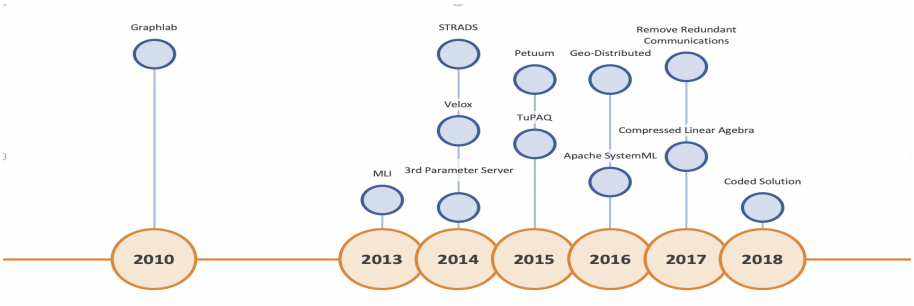

Figure 3.1: Timeline showing discussed frameworks and algorithms of machine learning

Apache Hadoop and Apache Spark are used to facilitate cluster computing and handle the storage and process of big data. However, they are not appropriate for scaling up machine learning algorithms and mentioned challenges because (18):

- Hadoop takes advantage of MapReduce abstraction which is too simple to be applied machine learning properties like error tolerance.
- Spark doesn't offer fine-grained scheduling of computation and communication.

These limitations motivate scientists and practitioner to look for suitable solutions.

According to Low et al. (14), Graphlab is a first graph-based framework expressing asynchronous iterative algorithms while considers the computational dependencies. It also ensures sequential-consistency and a high degree of parallelization. Graphlab can't support fault tolerance. It achieves a balance between high-level and low-level abstractions. To manage dependencies, Graphlab maps them into a data graph and provides sophisticated scheduling primitives. Computations are done in two phases: (1)update function assigned for local computation and (2)sync mechanism which defines global aggregation. In terms of consistency, Graphlab ensures that update functions never share overlapping sets. The main three approaches for maintaining sequential consistencies are full consistency, edge consistency, and vertex consistency. Moving on a scheduler, Graphlab benefits from a synchronous scheduler and task scheduler. Synchrony among vertices is guaranteed via a synchronous scheduler. Task scheduler is in charge of adding or reordering tasks(vertexes of a graph). The experiments prove that Graphlab can achieve a factor of 12 speedups on 16 cores. Figure 3.2

MLI (17) is an API designed by Sparks et al. for distributed machine learning. Its goal is to cover the gap between prototypes and industry allowing developers to control the communication and parallelization patterns. It has two kinds of an interface: MLTable

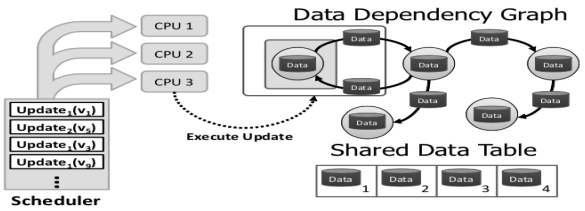

Figure 3.2: A Summary of Graphlab Framework. The user benefits from the graph to represent the data dependencies. Furthermore, the scheduling method is selected or defined by the user as well (14).

and LocalMatrix. MLI is built upon Spark and provides scalability up to 32 machines and ease of development.

Lee et al. develop (12) a programmable framework named STRADS (STRucture-Aware Dynamic Scheduler) to perform automatic scheduling and parameter prioritization while adheres to model-parallelization. The aim of this system is memory management and accelerating the convergence of machine learning algorithms. STRADS provides a simple abstraction for users to program machine learning algorithm consisting of three conceptual actions: schedule, push, pull. Push specifies how individual workers compute partial results while taking advantage of data partitioning. Pull focuses on how partial results are aggregated to perform the full model parameter update. To evaluate STRADS, three case studies were used: Latent Dirichlet Allocation(LDA), Matrix Factorization(MF) and Lasso. By pipelining schedule, master machines don't become a bottleneck even with a large number of workers. The globally accessible model parameters from the key-value store prevent pull-push communications between masters and workers and consequent bottleneck. To sum up, STRADS provides scalability and efficient memory utilization allowing larger models. The experiments across 128 machines confirm this statement. It is also able to invoke dynamic schedules which decrease model parameters dependencies, lower parallelization error, and faster correct convergence. Figure 3.3.

Velox (6), a model management platform provides an end-user application with low latency. Velox keeps the model up-to-date by taking advantage of offline and online incremental management strategies. Offline phase runs infrequently to adjust feature parameters. Online phase exploits the independence of user weights. If the error rates on any of the metrics exceed a threshold, the model will be retrained offline. Low latency is achieved via intelligently caching computation, scaling out model prediction, online training, and fast incremental model. Additionally, the model life cycle is managed and statistics about model performance and version histories are maintained. Simple rollback to earlier model

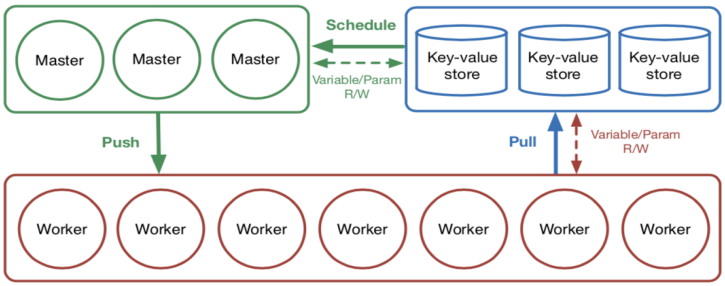

Figure 3.3: High-level Architecture of STRADS. Schedule(), Push() and Pull() are three conceptual actions which provide an abstraction for users to program machine learning algorithm (12).

version and adaptive feedback besides mentioned features describes the characteristic of Velox. The architecture of Velox consists of model predictor and model manager. Evaluation of Velox indicates that it can perform well on model update tasks and recovers prediction accuracy. Figure 3.4

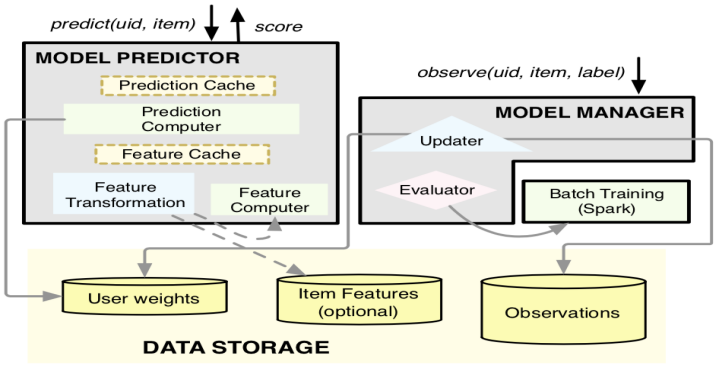

Figure 3.4: Architecture of Velox. Velox is able to serve large data product, originally in batch, with low latency (6).

Li et al. discuss (13) the third generation of the parameter server framework. In the parameter server approach, the resulting data inconsistency is trade-off between performance and algorithm convergence. Two relaxations are applied in this system for handling inconsistency: flexible consistency models and user-specific filters. Flexible consistency is done via task dependency graph in three ways: sequential consistency, eventual consistency and bounded delay. The system efficiency just tested on two problems including 0.6 PB on 1000 machines. Furthermore, the system provides data replication, instantaneous failover, and elastic scalability.

Petuum (18) is a platform for distributed machine learning aiming at industrial scale

problems using data and model parallel approaches. The main trade-off of Petuum are efficiency and correctness, programmability and generality. Based on their knowledge, Petuum was most expressive than previously introduced frameworks(before 2015). Three main properties of Petuum are as follows:

- Error Tolerance - robust against error in intermediate computation
- Dynamic Structural Dependency
- Non-Uniform Convergence

To accelerate the convergence of algorithms Petuum carries out these tasks:

- Synchronization of the parameters state with a bounded staleness with cheaper communication costs which results in correct outcome due to the error tolerance
- Dynamic scheduling policies taking the changing structural dependencies into account led to less parallelization error and synchronization costs
- Prioritizing computation toward non-converged models to gain faster convergence and avoiding wasteful updates of zero parameters.

In data parallelism, additive update making the update process much faster. On the other hand, a scheduler selects the model parameters to achieve model parallelism. Therefore, this framework consists of a parameter server, scheduler, and worker. A scheduler is responsible to control which model parameters are updated in workers using dependency-aware, prioritized or fixed scheduling. A worker runs the function push. Parameter server makes model parameters accessible globally. These tasks are done in three functions: schedule(), push(), pull(). Petuum expresses many ML algorithms such as Lasso, topic model, matrix factorization and deep learning. It also can handle larger model sizes. However, it scales less than 1000 machines and lacks automatic recovery. In general, Petuum is 2-6 times faster than other platforms. Figure 3.5

TuPAQ (16) is a system designed to efficiently and scalably automate the process of selecting a model(only supports a limited number of machine learning algorithms) and training data. The entire system is built upon Apache Spark. TuPAQ consists of four components: driver, planner, hyperparameter tuner, executor. This system is able to estimate the cluster size which is the appropriate number of machines required. TuPAQ algorithm for large-scale model search combines advanced hyperparameter tuning techniques with physical optimization for efficient execution. The aim of the model search,

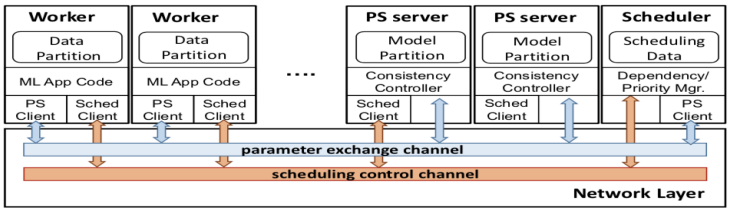

Figure 3.5: Petuum System. Three components of Petuum: scheduler, workers and parameter servers. A scheduler is in charge of controlling model parameters. Workers run push functions. Parameter servers make parameters accessible (18).

figure 3.6 is finding models with low generalization error. TuPAQ can cover shortcomings conventional sequential grid search and focus on quality and efficiency. The main four optimizations of this system are as follows:

- Determining ideal execution strategy based on data and budget
- Advanced hyperparameter tuning strategies
- Physical optimization via batching
- Bandit resource allocation via runtime inspection to make on-the-fly decisions

TuPAQ scale to models trained on Terabytes of data over hundreds of machines. The results prove that TuPAQ gains order-of-magnitude performance over the baseline method. Figure 3.7.

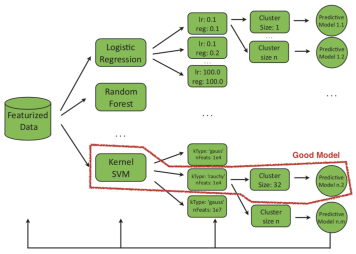

Figure 3.6: Model Search. The process of finding appropriate predictive model is automatic in TuPAQ (16).

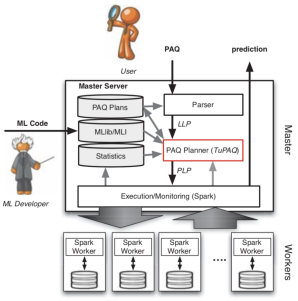

Figure 3.7: TuPAQ Architecture. TuPAQ is built upon Apache Spark in which the planner is a critical component. (16)

Cano et al. (4) propose a system built upon Apache Hadoop and Apache REEEF. This system is designed to accelerate the learning process and reduce communication overhead. It consists of three layers, from bottom to top: (1)resource manager, (2)control

flow(provides group communications), (3)geo-distributed machine learning framework(contains one algorithm of deep learning). This system achieves less communication passes, less storage and also less running time on up to 8 data nodes. However, in the case of larger model size, the performance of distributed version degrades rather centralized approach. Figure 3.8.

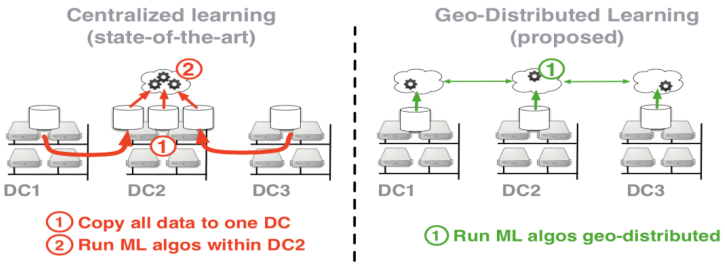

Figure 3.8: Centralized vs. Geo-distributed Learning

Apache SystemML (3) is a declarative machine learning framework on Spark separating algorithms semantic from underlying data representation and run time execution plan. SystemML covers a wide range of machine learning algorithms and generates a hybrid run time execution plan including single-node (scale-up) and distributed operations on a cluster of nodes (scale out). The hybrid plan is crucial for performance. This framework contains two main components: optimizer and runtime. In the optimization phase, the memory budget is estimated and the operator is selected based on characteristics of cluster and data. The key optimizations are automatic RDD caching, checkpointing and repartitioning as well as partitioning-preserving to minimize the number of data scans and shuffles. SystemML is also capable of managing task parallelization. In the next phase, runtime, data serialization, and partitioning are performed. The experiment was successful on the cluster of 7 nodes and up to 800 GB of data.

Due to the communication overhead of distributed machine learning particularly models with more dimensions, Yokoyama and Araki (19) propose a synchronous communication method to decrease redundant transmissions of parameters without changing the structure of algorithms. This method identifies collective communication that could be omitted and replaced with direct communications between workers. Since not all elements of model parameters changes in each mini-batch of the input at a time, transmitting whole parameters are inefficient and redundant. Furthermore, as the training dataset get sparser, the numbers of inefficient communication arise. The configuration of the system is symmetric to balance the workload. It means that all machines are both parameter server and

worker. The result across five computing nodes proves that traffic decreases as the size of mini-batches get smaller. Because small batches lead to less overlapping feature updates. Experiments also indicate the reduction of total training time in general. See figures 3.9 and 3.10.

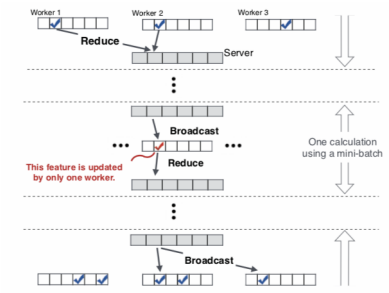

Figure 3.9: Redundant Communications

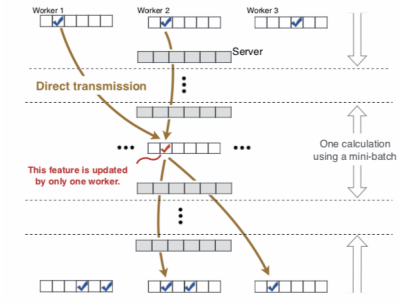

Figure 3.10: Efficient Communications

One approach to improve performance is taking advantage of a compression method to reduce memory requirement. Compressed Linear Algebra(CLA) can apply lightweight lossless compression to matrices in which linear algebra operations such as matrix-vector multiplication are executable on compressed representation. CLA achieves good compression ratio and also performance close to an uncompressed method. Furthermore, it generalizes the representation of dense and sparse matrices. CLA is a column-based and value-centric method. It is also based on Finding out the common characteristics of data(Column Encoding Format) and even grouping correlated columns of matrices(Column Co-Coding) to speedup the processing phase. Offset-List Encoding (OLE) and Run-Length Encoding (RLE) are the main algorithms for compression. The proposed system by Elgohary et al. (8) automatically chooses the most efficient approach according to the random sample of data. This is done in the phase of planning. In compression step, after the main procedure and in case of false positive, the system carries out the correction and repeats this step again. Overall, CLA experiments prove significant performance across a cluster of seven nodes for iterative algorithms over real-world data due to the reduced I/O and smaller memory bandwidth.

Coded solutions are proposed to deal with the performance of distributed learning. Since most building blocks of DL algorithms are matrix multiplication and data shuffling, Lee et al. (11) concentrate only on them. Codes mitigate the effect of stragglers and speedup matrix multiplication by adding redundancy. Exploiting excess storage, data shuffling combined with codes encounters fewer communication bottlenecks. The benefit of coded

solutions is that there is no need to modify distributed systems to accommodate them. The result across 10 workers shows that the coded version outperforms other approaches in most cases. Coded shuffling takes advantage of extra memory at each node while coded MapReduce introduces redundancy in the computation of the mappers. See figures 3.11 and 3.12

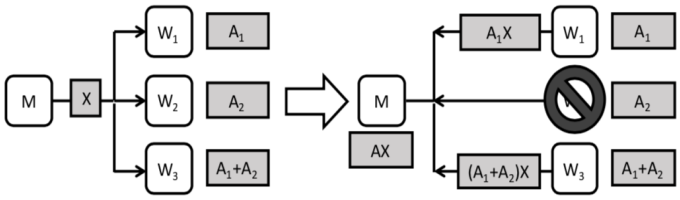

Figure 3.11: Coded Matrix Multiplication. Data matrix A is partitioned into 2 sub-matrices: A1 and A2. Node W1 stores A1, node W2 stores A2, and node W3 stores A1 + A2. Upon receiving X, each node multiplies X with the stored matrix and sends the product to the master node. Observe that the master node can always recover AX upon receiving any 2 products, without needing to wait for the slowest response. For instance, consider a case where the master node has received A1X and (A1 + A2)X. By subtracting A1X from (A1 + A2)X, it can recover A2X and hence AX (11)

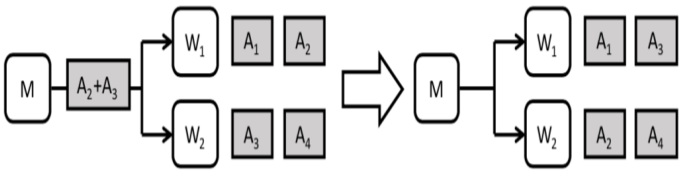

Figure 3.12: Coded Data Shuffling. Data matrix A is partitioned into 4 sub-matrices: A1 to A4. Before shuffling, worker W1 has A1 and A2 and 4 worker W2 has A3 and A4. The master node can send A2 + A3 in order to shuffle the data stored at the two workers (11).

The following paragraphs concentrate on deep learning figure.

Chilimbi et al. (5) propose Adam, a system for efficient and scalable deep learning training system which balances the workload of communication and computation in an asynchronous way. Larger DNN and more training data effectively increase the accuracy of the image recognition task. Consisting of both convolutional followed by fully connected layers, Adam can be adapted in different applications. In order to model the training process, the following list is used:

- Multi-threaded learning to update local parameters

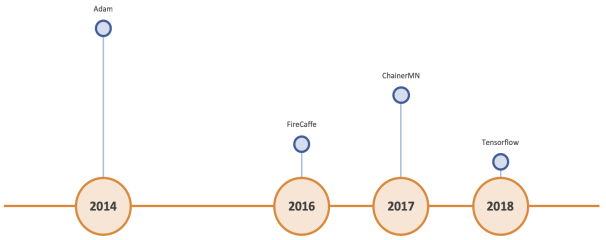

Figure 3.13: Timeline showing discussed frameworks of deep learning

- Fast weight updates
- Reducing memory copies
- Memory system optimization using L3 Cache
- Mitigating the impact of slow machines by parallel processing of input
- Parameter server communication by accumulating the updates for convolutional layers and sending activation and gradient to do local matrix multiplication for fully connected Layer

Furthermore, distributed parameter server machines in Adam are useful to manage load balancing, batch updates and fault tolerance. The result of testing Adam indicates the growth of accuracy which contradicts the common knowledge that asynchrony lowers the prediction accuracy(The testbed was a cluster of 120 machines). Adam is also capable of training very large models with good scaling.

Iandola et al. (9) introduce FireCaffe, a framework for scalable DNN training focusing on communication overhead reduction without the degradation of accuracy over 128 GPUs. FireCaffe benefits from the reduction tree in communication which is more efficient than parameter server and larger batch size. Reduction tree scales at the log of the number of workers while parameter server scales linearly. So, choosing the efficient tree as a communication protocol is more efficient. FireCaffe achieves 39x speedup across 128 GPUs over single-GPU training. In selecting the architecture of DNN, the main goal was fewer parameters which means less communication.

ChainerMN (1), distributed deep learning framework, can achieve 90% efficiency by using 128 GPU training ImangeNet dataset. Tow main features of ChainerMN are as follows:

- Flexibility: It can be used even in complex use cases such as: dynamic neural networks, reinforced deep learning.
- High Performance: To achieve this, hardware performance is fully utilized.

ChainerMN focuses on computing performance and benefits from the synchronous data parallel model. In order to increase performance, it attempts to minimize communication time by using appropriate libraries. Empirical experiments (up to 128 GPUs) proves the scalability of ChainerMN near to ideal.

Tensorflow (2) is by far the most popular framework adopted for ML and DL applications. Several libraries are integrated with Tensorflow to enable distributed Deep Neural Network (DNN) training at scale. To name some communication libraries, we can point to gRPC(point to point RPC) and CUDA-aware MPI. Bigger batch size and faster GPUs offer better performance up to the point of diminishing returns. Awan et al. (2) drew a comparison among the design and performance of these libraries across up to 128 nodes.

3.2.1 Academic and Commercial Support for Frameworks

Among the mentioned frameworks and platforms, some of them found their way into academia, industry or even both. In this section, we look into them in details.

GraphLab Create, one of the top 50 Artificial Intelligence Software products, is a large scale distributed machine learning platform created by Turi. This project was started by Carlos Guestrin and his students at Carnegie Mellon University. Although Turi supports this software for non-commercial and academic use, GraphLab can be adapted to meet all the needs of different company types, sizes, and industries.

Petuum Symphony, an AI software platform, was developed by professors and scientists of Carnegie Mellon University allowing enterprises to design, build, experiment, customize, operate and own vertical AI solutions in a wide range of industries and areas. With $108 million in investor funding, Petuum is one of the highest-funded AI and Machine Learning startups.

Apache SystemML provides an optimal workplace for machine learning using big data on top of Apache Spark. This platform is supported by Apache Software Foundation. Future SystemML developments include additional deep learning with GPU capabilities.

ChainerMN is a Python-based, standalone open source framework for deep learning models. The development of Chainer is running on the official repository at GitHub enabling developers to contribute to this project. There is more details regarding the comparison of ChainerMN with other frameworks such as Tensorflow and Pytorch in its blog.

TensorFlow is an open source software library for numerical computation using data flow graphs. TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization for the purposes of conducting machine learning and deep neural networks research. This library comes with strong support for machine learning and deep learning use cases. Many well-known companies benefit from TensorFlow such as Google, Airbnb, Twitter, Uber, Intel, Dropbox and so on.

3.3 Limitations and Future Work

Lack of standard measures for the evaluation of distributed learning is considered as one of the main limitations (15). These measures are regarding efficiency and scalability of approaches in term of memory, computational and communication resources. As a result, it is difficult to compare the approaches from different sides like test error, training time, memory requirement based on the size and dimensionality of data sets.

Secondly, as data is distributed over several locations, the probability of network partitioning occurrence goes up; because providing consistent network connectivity is much more difficult than within a single data center (4).

Fault tolerance issue in case of persistent or transient data unavailability is still an open problem (4). Running a computational task at a computing node involves unpredictable latency due to the factor like, network latency, shared resources, maintenance activities and power limits (11).

In addition to standard measurement, network consistency, and fault tolerance, each time data is shuffled, the communication could become performance bottleneck (13, 19).

What prevents machine learning algorithms at scale widely applied in many use cases is the difficulty of migration from trusted and easily controllable computer platforms to corporate cluster and cloud which are less predictable (18).

Training DNN, especially at scale, is a computation and communication intensive task. Stochastic Gradient Descent(SGD) is used to solve DNN which is an inherently sequential algorithm. Therefore, to parallelize training DNN, we are only able to use a faster computation unit in which communication time exceeds computation time only after scaling up to a few nodes. Based on the model size, the training problem becomes communication bound. Moreover, Limited network bandwidth is one of the key bottleneck limiting performance and scalability. Distributed learning of DNN contains two types of parallelization. (1) Inner parallelization is used to forward and backward operations by parallel algorithms

within the computation units. (2)Outer parallelization over distributed batches. In case of non-square matrices, inner parallelization hurt outer one leading to performance degradation. In order to reduce the number of iterations to converge the algorithm, the step size and also batch sizes must go up. Although the system reaches linear speedup, the validation accuracy suffers significantly. Furthermore, some of the layers of the network aren't fully scalable. The other latency caused by random access to a file system which may bring out break down (10).

Moving on the direction of future work, leveraging coded solutions for a broader class of distributed algorithms, convergence analysis of coded algorithms and compression techniques for more operations are still open problems (11).

4

Conclusion and Discussion

In this paper, the main challenges of applying machine learning at scale are discussed in details. In large-scale machine learning problems, models with billions of parameters must be estimated. Data-parallel and model-parallel approaches are utilized in order to distribute the processing phase inside a cluster of workers. In data parallelism, full access to global parameters and consequent communications may become a bottleneck. On the other hand, model-parallelism resolves this issue while dealing with intrinsic dependencies within models. The performance of such systems depends on whether the data fit the memory or not. New compression techniques coming along to decrease the volume of data. Furthermore, noises such as slow workers, system failures and communication bottlenecks can affect the performance as well. In such cases, data replication (coded solutions) become an applicable method. In order to benefit from technical contributions in industry, latency is the major issue.

In addition to the general issues of machine learning, deep learning and some related frameworks are studied separately. Due to the availability of data and modern hardware such as GPU, training of deep learning turns applicable. Adam, FireCaffe, ChainerMN and, Tensorflow are some of the well-known frameworks of DNN. Some approaches take advantage of parameter servers while others implement deep learning in a synchronous way.

Majority of these solutions were tested across clusters of less than 1000 machines. Moreover, some of the frameworks only focus on a limited number of algorithms in machine learning. Therefore, it is still a long way in this domain for scientists and practitioners.

5

Appendix

5.1 Planning

5.1.1 Rationale for Primary Studies Selection

Primary studies can be grouped into two main classes. First class focuses on surveys to give an overview of the topic and introduces a various classification of methodologies, algorithms, and tools. These surveys are also useful to detect the limitations and challenges from a high-level point of view. The second class looks into some of the recent technical contributions such as frameworks, architectures, systems, and optimizations. Each of these approaches targets some of the challenges. For instance: communication overhead, data and model parallelization, fault tolerance and so on.

5.1.2 Search Query

According to research questions, we extract keywords and query strings. With the help of query strings, the search area is narrowed down.

(Machine Learning **OR** Deep Learning) **AND** (Distributed **OR** Scalability) **AND** (Large-scale)

5.1.3 Inclusion/Exclusion Criteria

Inclusion Criteria

- **I1**: A study developed by either academics or industry.
- **I2**: A study proposing a new framework, system, architecture, tool, algorithm, methodology regarding scaling up machine learning algorithms especially deep learning.

- **I3**: A study reviewing available approaches for scaling up machine learning.
- **I4**: A study related to distributed and large-scale learning.
- **I5**: A study published in the domain of large-scale data engineering and artificial intelligence.
- **I6** A study written in English.

Exclusion Criteria

- **E1**: A study addressing only theoretic parts of machine learning algorithms at large-scale.
- **E2**: A study not considering large-scale dataset as input.
- **E3**: A study targeting mainly hardware architectural side of large-scale learning.

5.2 Conducting

This section addresses the main phase of the literature review. The goal of conducting activity is retrieving potential papers touching upon the topic. Then, irrelevant options are discarded and primary studies are classified based on the research questions. Finally, the data is extracted to answer the research questions and make a conclusion.

5.2.1 Search and Selection Process

The major fund of the literature review is publications. Therefore, it is significant to pick a number of relevant papers while having a comprehensive study.

1. **Initial Search**: According to the search queries specified in the planning section, we have selected 166 papers as potential cases. Zotero software was used to store and categorize papers. These papers were chosen from different digital libraries namely IEEE explore, Springer and ACM digital library.
2. **Classification**: As the goal of this study is to give a broad view of the topic and also provide a number of technical contributions, classification can help to focus on this direction. We categorized all potential papers into two groups: (1)surveys (2)frameworks, systems, algorithms, and tools.

3. **Spam Detection**: After filtering out selected papers using inclusion/exclusion criteria, around 60 papers have remained.

4. **Final Selection**: To carry out a valid literature review according to (7), primary studies should cover 15-20 publications. Looking into the abstract, introduction and conclusion we picked 40 papers and then each of us took over 20 studies.

5.2.2 Data Extraction

The aim of data extraction is preparing data for further analysis and answering the research questions. Since we classified the primary studies in the previous step, answering the questions becomes much more straightforward. During the consideration of each publication, we highlighted the most relevant paragraphs and then summarized the whole idea of paper respectively. Finally, all extracted data were aggregated for the next step, data analysis. We found out following grouping useful.
RQ1: New Challenges: To answer this question, we picked related surveys. Furthermore, we derived all motivations for introducing a new framework or techniques from different primary studies.
RQ2: Current Solutions: The summary of each specific technical publications was merged to explain this question.
RQ3: Limitations and Future Work: Similar to the previous question, we took advantage of surveys and different specific techniques. The future work section of each publication was helpful as well.

5.2.3 Data Analysis and Result

By accumulating the output of previous activity, we thoroughly answered the research questions and made a conclusion.

5.3 Reporting

5.3.1 Report

In this part, we looked into the primary studies and extracted a summary of them in addition to the answer of research questions. Finally, merging derived data, we made a clear conclusion of this literature study.

5.3.2 Presentation

Based on the reports, we prepared slides for a presentation.

References

[1] Akiba, T., Fukuda, K., and Suzuki, S. (2017). Chainermn: Scalable distributed deep learning framework. *CoRR*, abs/1710.11351. 6, 7, 16

[2] Awan, A. A., Bédorf, J., Chu, C.-H., Subramoni, H., and Panda, D. K. (2018). Scalable distributed dnn training using tensorflow and cuda-aware mpi: Characterization, designs, and performance evaluation. *CoRR*, abs/1810.11112. 17

[3] Boehm, M., Dusenberry, M. W., Eriksson, D., Evfimievski, A. V., Manshadi, F. M., Pansare, N., Reinwald, B., Reiss, F. R., Sen, P., Surve, A. C., and Tatikonda, S. (2016). Systemml: Declarative machine learning on spark. *Proc. VLDB Endow.*, 9(13):1425–1436. 13

[4] Cano, I., Weimer, M., Mahajan, D. K., Curino, C., Fumarola, G. M., and Krishnamurthy, A. (2017). Towards geo-distributed machine learning. *IEEE Data Eng. Bull.*, 40:41–59. 6, 12, 18

[5] Chilimbi, T., Suzue, Y., Apacible, J., and Kalyanaraman, K. (2014). Project adam: Building an efficient and scalable deep learning training system. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 571–582, Berkeley, CA, USA. USENIX Association. 15

[6] Crankshaw, D., Bailis, P., Gonzalez, J., Li, H., Zhang, Z., Franklin, M. J., Ghodsi, A., and Jordan, M. I. (2015). The missing piece in complex analytics: Low latency, scalable model management and serving with velox. *CoRR*, abs/1409.3809. ii, 7, 9, 10

[7] Dyba, T., Dingsoyr, T., and Hanssen, G. K. (2007). Applying systematic reviews to diverse study types: An experience report. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 225–234. 3, 23

[8] Elgohary, A., Boehm, M., Haas, P. J., Reiss, F. R., and Reinwald, B. (2017). Scaling machine learning via compressed linear algebra. *SIGMOD Rec.*, 46(1):42–49. 7, 14

REFERENCES

[9] Iandola, F. N., Ashraf, K., Moskewicz, M. W., and Keutzer, K. (2016). Firecaffe: Near-linear acceleration of deep neural network training on compute clusters. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2592–2600. 16

[10] Keuper, J. (2016). Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability. *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, pages 19–26. 19

[11] Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D., and Ramchandran, K. (2018). Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529. iii, 7, 14, 15, 18, 19

[12] Lee, S., Kim, J. K., Zheng, X., Ho, Q., Gibson, G. A., and Xing, E. P. (2014). On model parallelization and scheduling strategies for distributed machine learning. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2834–2842. Curran Associates, Inc. ii, 6, 7, 9, 10

[13] Li, M., Andersen, D. G., Smola, A., and Yu, K. (2014). Communication efficient distributed machine learning with the parameter server. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 19–27, Cambridge, MA, USA. MIT Press. 10, 18

[14] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. (2010). Graphlab: a new framework for parallel machine learning. In *UAI 2010*. ii, 8, 9

[15] Peteiro-Barral, D. and Guijarro-BerdiÃśas, B. (2013). A survey of methods for distributed machine learning. *Progress in Artificial Intelligence*, 2(1):1–11. Exported from https://app.dimensions.ai on 2019/02/07. 1, 5, 6, 18

[16] Sparks, E. R., Talwalkar, A., Haas, D., Franklin, M. J., Jordan, M. I., and Kraska, T. (2015). Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 368–380, New York, NY, USA. ACM. ii, 7, 11, 12

[17] Sparks, E. R., Talwalkar, A., Smith, V., Kottalam, J., Pan, X., Gonzalez, J., Franklin, M. J., Jordan, M. I., and Kraska, T. (2013). Mli: An api for distributed machine learning. In *2013 IEEE 13th International Conference on Data Mining*, pages 1187–1192. 8

[18] Xing, E. P., Ho, Q., Dai, W., Kim, J. K., Wei, J., Lee, S., Zheng, X., Xie, P., Kumar, A., and Yu, Y. (2015). Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2):49–67. ii, 8, 10, 12, 18

[19] Yokoyama, H. and Araki, T. (2017). Efficient distributed machine learning for large-scale models by reducing redundant communication. In *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1–8. 13, 18