KunPeng: Parameter Server based Distributed Learning Systems and Its Applications in Alibaba and Ant Financial*

Jun Zhou†, Xiaolong Li†, Peilin Zhao†, Chaochao Chen†, Longfei Li†, Xinxing Yang†, Qing Cui‡,

Jin Yu‡, Xu Chen‡, Yi Ding‡, Yuan Alan Qi†

†Ant Financial Services Group, Hangzhou, China
‡Alibaba Cloud, Hangzhou, China
{jun.zhoujun,xl.li,peilin.zpl,chaochao.ccc,longyao.llf,xinxing.yangxx,yuan.qi}@antfin.com
{cuiqing.cq,kola.yu,xuanle.cx,christopher.dy}@alibaba-inc.com

ABSTRACT

In recent years, due to the emergence of Big Data (terabytes or petabytes) and Big Model (tens of billions of parameters), there has been an ever-increasing need of parallelizing machine learning (ML) algorithms in both academia and industry. Although there are some existing distributed computing systems, such as Hadoop and Spark, for parallelizing ML algorithms, they only provide synchronous and coarse-grained operators (e.g., Map, Reduce, and Join, etc.), which may hinder developers from implementing more efficient algorithms. This motivated us to design a universal distributed platform termed **KunPeng**, that combines both distributed systems and parallel optimization algorithms to deal with the complexities that arise from large-scale ML. Specifically, KunPeng not only encapsulates the characteristics of data/model parallelism, load balancing, model sync-up, sparse representation, industrial fault-tolerance, etc., but also provides easy-to-use interface to empower users to focus on the core ML logics. Empirical results on terabytes of real datasets with billions of samples and features demonstrate that, such a design brings compelling performance improvements on ML programs ranging from Follow-the-Regularized-Leader Proximal algorithm to Sparse Logistic Regression and Multiple Additive Regression Trees. Furthermore, KunPeng's encouraging performance is also shown for several real-world applications including the Alibaba's Double 11 Online Shopping Festival and Ant Financial's transaction risk estimation.

CCS CONCEPTS

•**Theory of computation** →**Distributed algorithms;** *Communication complexity;*

*KunPeng (pronounced like \kunpɛŋ\) is a giant fish that could turn into a giant bird in Chinese myth, thereby being metaphorized into a transformer that can play unimpededly between large-scale distributed computing systems and distributed optimization algorithms.

KEYWORDS

Parameter Server, distributed computing system, distributed optimization algorithm

ACM Reference format:
Jun Zhou†, Xiaolong Li†, Peilin Zhao†, Chaochao Chen†, Longfei Li†, Xinxing Yang†, Qing Cui‡, Jin Yu‡, Xu Chen‡, Yi Ding‡, Yuan Alan Qi†. 2017. KunPeng: Parameter Server based Distributed Learning Systems and Its Applications in Alibaba and Ant Financial. In *Proceedings of KDD'17, August 13–17, 2017, Halifax, NS, Canada., ,* 10 pages.
DOI: http://dx.doi.org/10.1145/3097983.3098029

1 INTRODUCTION

As the popularity of mobile internet and data collection equipments increases, the industrial community has collected huge amounts of data (terabytes or petabytes) which reveals great commercial values in recent years. Many internet companies have invested a lot of resources to study how to efficiently extract more valuable information from Big Data. Traditional data processing platforms, such as MATLAB, SAS, and R can hardly deal with the TB level of data. To meet this challenge, various large data processing frameworks have emerged, such as Hadoop [27], Spark [33], GraphLab [20], and GraphX [12].

One of the most widespread distributed computing frameworks is Hadoop, which is a typical bulk-synchronous processing flow. Through coarse-grained primitives like map and reduce [9] which are succinct and easy to master, it greatly improves the efficiency of data processing, and has been ubiquitously used in the industry. With immutable data abstraction, aka Resilient Distributed Dataset (RDD), Spark extends Hadoop with the ability to cache previously computed datasets in memory, and therefore accelerates data processing efficiency. GraphLab or GraphX provides a useful encapsulation of graph manipulation interfaces which introduce new techniques to construct, slice and modify graphs, thus can efficiently perform complex graph algorithms. Among the Big Data processing techniques, ML has become a critical technique which can learn the potential patterns and knowledge from data thus can help internet companies to provide high quality services for users. Some examples include click-through rate (CTR) prediction in online advertisement, ranking for web search, recommendation for online shopping, and so on. However, due to various limitations of MapReduce (such as intermediate results written to disks,

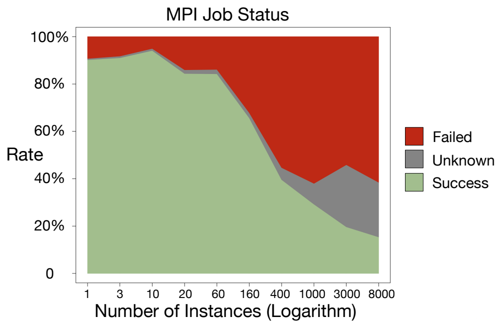


Figure 1: MPI job status in a production cluster of Alibaba.

communication only during shuffling, etc.), many ML algorithms implemented on it are quite inefficient and have been transferred to other frameworks [28]. While Spark MLlib[1] is easy to use and integrate different ML algorithms, it can only support medium scale of features and suffers from low computational and communication efficiency. To tackle the problem of large-scale ML, some companies try to utilize an independent component to make up for the disadvantages of Spark, but the combination with Spark may be far-fetched which is not an elegant solution [4, 32] . GraphLab or GraphX exposes to user fine-grained control over the access patterns on vertices, yet it is not suitable for general ML algorithms such as regression and deep learning and it also often suffers from the low efficiency problem.

Due to the disadvantages of existing frameworks, many internet companies also directly take low-level approaches such as Message Passing Interface (MPI) to implement ML algorithms, which is very flexible and expressive, and has little restriction on the code structure (e.g., all the processes can communicate among each other at any time). But this increases development cost, because developers need to write quite a lot of codes to implement the algorithm (for example, a sophisticated asynchronous matrix factorization program [31] has more than 2,000 lines of low-level code using MPI). Even worse, some common components in distributed ML platform are missing in MPI, such as distributed data loading, memory management and multi-threaded parallelization, which may result in bug-prone programming. More seriously, MPI does not have a native solution to a single point of failure (SPOF) in distributed systems. For instance, in a real production cluster, as shown in Figure 1, we find that when the concurrent worker instances in one job rise to between 160 to 400, the success rate is only 40% $\sim$ 65%, and with more than 1,000 instances, the success rate is further down to 30%.

With rapid prosperity of large-scale ML, especially deep learning, industry and academia pay attention to a framework called parameter server. A parameter server system consists of two disjoint sets of components: stateless workers that perform the bulk of the computation when training a model, and stateful servers that maintain the current version of the model parameters [8, 17, 25, 28]. In parameter server, the huge model parameters are distributed on the servers and the workers can obtain the model parameters through network communications, thus it can handle hundreds of billions of model parameters in a single ML job which is impossible for aforementioned platforms. Apart from the ability of handling large-scale problems, the parameter server also makes a native solution to node failures in the clusters. At specified intervals, parameter server can save model parameters on servers to a cache location as a checkpoint, and when some nodes fail, the framework can automatically recover the model parameters from cache location and start over from the latest checkpoint. Because the original parameter server requires the workers to be stateless, so the recovery process is relatively simple. This conventional parameter server design is widely used in industrial practice, but also suffers from drawbacks, e.g., the communication interface is not as flexible as MPI and programming primitives are not as easy-to-use as Spark.

Those weaknesses discussed above motivate us to propose a production-level distributed learning system called KunPeng. It combines the advantages of parameter server and MPI with additional flexibility. Besides all those fundamental properties, KunPeng also comprises of many optimizations for industrial scenarios such as: (1) A robust failover mechanism which guarantees the high success rate of large-scale jobs in busy production settings with all kinds of jobs (e.g., SQL, Spark, MPI, etc.) running together; (2) An efficient communication implementation for sparse data and general communication interfaces which are compatible with MPI, such as AllReduce, ReduceTo, Bcast, etc; (3) A user-friendly C++ and Python SDKs which provide a development environment similar to local machine; (4) A tight combination with Apsara (to be seen shortly) [34] which provides a full toolkit for ML tasks including data preprocessing with the help of SQL and MapReduce, prediction, evaluation and so on.

We also implemented many popular ML algorithms based on KunPeng which are broadly applied inside Alibaba and Ant Financial. For example: (1) The largest scale of online learning system to our best knowledge. We built a bounded asynchronous Follow-the-Regularized-Leader Proximal (aka FTRL-Proximal) [21] with trust region which can process tens of billions of samples within 10s of minutes; (2) The state-of-the-art Multiple Additive Regression Trees (MART) [11] algorithms, named as KunPeng-MART, which supports distributed LambdaMart [5] as well as all functionality of XGBoost (a parallel tree boosting system based on Rabit[2]) [6]; (3) Large-scale deep learning platform based on CPU cluster and a Latent Dirichlet Allocation (LDA) algorithm which has the same largest scale as reported in literature [30]. Besides, we also built a user-friendly ML pipeline which provides simple interfaces, convenient progress monitoring, and robust online serving API.

Over the past years, KunPeng has been deployed in more than 100 products in Alibaba and Ant Financial which cover many important scenarios, including but not limited to Alibaba's Double 11 Online Shopping Festival [1], Taobao's (a consumer-to-consumer e-commerce platform) search advertisements, Tmall's (a business-to-consumer e-commerce platform) personalized item recommendation, music/news/video/feeds recommendation, the Cainiao[3] logistics platform, UCWeb's mobile search ranker, and several important applications of Ant Financial.

[1]http://spark.apache.org/mllib/

[2]https://github.com/dmlc/rabit

[3]https://www.cainiao.com/

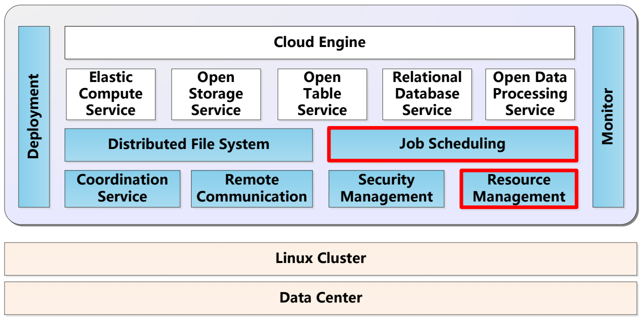


Figure 2: Apsara system overview. The boxes with red borders are Fuxi.

2 SYSTEM OVERVIEW

In this section, we provide a brief introduction to the Apsara cloud platform, and then present an overview of KunPeng.

2.1 Apsara Cloud Platform

Apsara is a large-scale distributed operating system developed by Alibaba, which has been deployed on hundreds of thousands of physical servers across tens of data centers at Alibaba since 2009, with the largest clusters consisting of more than 10,000 servers each. The core concerns of Apsara are resource management/allocation in distributed computing environment and scheduling/running all kinds of distributed jobs. To achieve these goals, several related systems cooperate with each other. The Apsara system architecture is illustrated in Figure 2, where the sky blue boxes are Apsara modules, and the white boxes are cloud services running on top of Apsara. The resource management and job scheduling modules are collectively called Fuxi (shown by boxes with read borders) [34] which follows the common master-slave architecture similar to Yarn [26]. To scale Fuxi to tens of thousands of nodes and hundreds of thousands of concurrent processes, the system adopts many key strategies, including incremental resource management protocol, user transparent failure recovery, faulty node detection and multi-level blacklist, to maintain high resource utilization, and shield low-level failures from affecting various applications. Empirically, these techniques are very effective to improve the scalability and fault-tolerance of similar systems that handle internet-scale traffic.

2.2 KunPeng Platform

KunPeng is designed to make it easy and effective for a wide variety of end-users (e.g., data analysts and data mining engineers) to discover patterns efficiently in Big Data. While KunPeng can deal with the complexities that arise from large-scale ML, it also gives ML algorithm developers the impression of writing codes for a single machine. It offers two crucial sub-systems to achieve these goals (Figure 3): (1) ML-Bridge: a practical ML pipeline based on a set of cloud services running on top of Apsara which is mainly a script-style workflow that enables end-users to go quickly from inspiration into production; (2) PS-Core: a parameter server based framework with a distributed key-value store semantics. The PS-Core not only encapsulates the characteristics of data/model parallelism, straggler handling, model sync-up (e.g., Bulk Synchronous Parallel (BSP) [7],

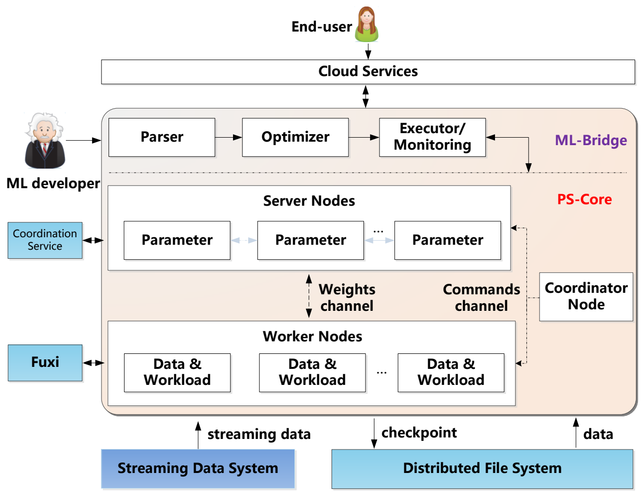


Figure 3: KunPeng architecture overview, including ML-Bridge and PS-Core. ML developers use SDK of KunPeng to implement distributed ML algorithms. End-users enjoy the convenience by tight combination with other cloud services.

```
    --data synchronization
    sync xxSrcData to xxDestTable;
    fea_engineering -i xxDestTable -o feature_engineering_result;
    --prepare training data
    create table if not exists demo_batch_input (labels string,kvs
string);
    insert overwrite table demo_batch_input  select labels,kvs
    from feature_engineering_result where xxCondition;
    --KunPeng training
    ps_train -i demo_batch_input -o demo_batch_result -a xxAlgo -t
xxTermination;
    --deployment
    run prediction and evaluation
    copy demo_batch_result to xxSystem;
    --other workflow
    AB Testing
```

Figure 4: A typical example of end-users using KunPeng, including: data synchronization, preprocessing, feature combination, model training, etc.

Asynchronous Parallel (ASP) and Stale Synchronous Parallel (SSP) [14, 18]), sparse representation, industrial fault-tolerance, etc., but also provides easy-to-use Python and C++ APIs to help users focus on the core ML logics.

ML-Bridge: We draw inspiration from the high-level programming models of dataflow systems [2, 29]. At Alibaba and Ant Financial, there are already many dataflow frameworks on Apsara, such as data synchronization systems and data pre-processing systems. ML-Bridge builds a full ML pipeline with the help of these systems to harness the power of ML algorithms. It is composed of three components: (1) a simple parser, which supports plenty of commands to specify ML tasks that are compatible with existing systems; (2) an optimizer, which analyses, debugs, and optimizes configuration of ready-to-submit tasks by historical running statistics and heuristic methods; (3) an executor, which generates task description (e.g., application type and resource demand) in accordance with the arguments of Fuxi, requests Fuxi to allocate resources and launch the task, watches the whole lifecycle of the submitted task (aka PS-Core), and provides convenient progress monitoring UI to end-users. A typical demonstration of end-users using KunPeng is displayed in Figure 4, which shows that end-users can flexibly combine different data processing modules in SQL-like way to perform end-to-end ML tasks, even without a strong background in distributed systems and low-level primitives.

As shown in Figure 3, PS-Core follows a common master-slave pattern and has three roles: iteration controlling (coordinator node), distributed model vectors (server nodes), and data-computing executors (worker nodes). Since PS-Core is an extension of the typical PS framework, we will avoid repeating the same details for conciseness, but highlight the major differences that are specialized for industrial scenarios in Alibaba and Ant Financial. The detailed descriptions are shown as follows.

Coordinator: It is a light-weighted and fully-centralized driver that declares the computation on the data and operations on the distributed vectors. The coordinator constructs task graph of ML workflows, schedules them to run on the workers/servers, and coordinates all the nodes and overall execution of tasks. Coordinator looks like a single point, but does not suffer from the SPOF, since it synchronizes the simplest iteration states (e.g., the current step) with distributed meta storing service of Apsara at the end of each iteration. Meantime, its status is monitored by Fuxi. If it crashes, Fuxi will restart it instantly (its resource demand is extremely low). Then it reads out previous bookkeeping states, and forces the other nodes to achieve a consistent state. In this way, only a little time is needed to resume algorithm iteration. Furthermore, this design makes coordinator simpler and cleaner than the typical hot-standby approach [24].

Servers: The server nodes store the distributed huge parameters (which sometimes cannot be stored on a single machine) where the partitions are implemented using consistent hashing. The main communication interfaces for servers are push and pull primitives with a convenient encapsulation, based on which most of the communication semantics needed for ML can be easily implemented. The number of the server nodes on the same machine is intelligently capped to avoid network congestion. The above design not only makes the distributed key-value storage of server nodes reliable, but also guarantees the full communication functionality.

Workers: In general, each worker loads an evenly partitioned subset of data, which can be processed in stochastic mini-batch or batch ways. Each worker maintains a rather small working set of full parameters, due to sparse representation, through a step called sparse statistics computation, in which each worker aggregates all the features that appear in its local data partition. The sparse representation can greatly improve communication/computation efficiencies, and reduce memory consumptions. Additionally, workers adopt the atomic operation in concurrent programming instead of locking on the local vectors (e.g., gradient vector), which further reduces the memory cost. In short, the major functionality of the worker nodes is to take heavy workload and maximize resource utilization.

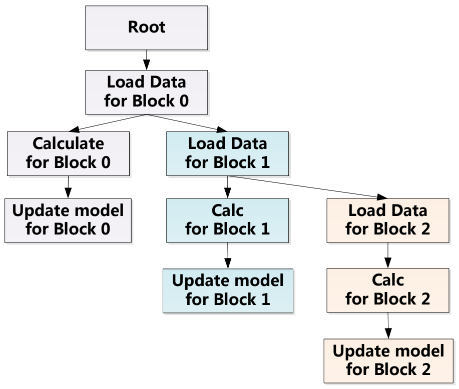


Figure 5: An example of DAG scheduling. Vertices of the same depth can be executed concurrently. For example, the operation of "Calculate for Block 0" and "Load Data for Block 1" can conduct at the same time on each worker.

Fault Tolerance: In the scenario of large-scale ML tasks in busy production environment, it is critical to have robust fault tolerance because machines could crash accidently at any time. The parameters on server nodes will be asynchronously dumped to the distributed file system (DFS) as checkpoints. To further speed up the recovery process, we also backup two copies of parameters in memory for each server node at the cost of more memory consumption, which is actually a hot failover. In most cases, the server nodes instantly recover from failure via the memory backups, and even though the training is completely interrupted (e.g., resource preemption) or killed, the training can resume from the latest checkpoint. Worker nodes are usually stateless in the original design, we extend it with interfaces for workers to save checkpoints, so that ML algorithms with state on workers can also be easily recovered, such as LDA. During its failover, a worker node first re-pulls the latest working sets of parameters from server nodes, and then re-computes on local datasets. When coordinator receives asynchronous node failure notifications from Fuxi, it breaks algorithm iteration, sends requests to recover server nodes and worker nodes to the right step, and waits them completing state restoration. The whole recovery is fully automatically completed, and user-transparent. Different from Petuum [28], this strategy can be applied to tens of thousands of machines. In Section 4, we empirically show that KunPeng's fault tolerance is much more robust than checkpoint-and-restart methods.

Scheduling: All the general algorithm workflows can be constructed as a directed acyclic graph (DAG) on the coordinator, as presented in Figure 5, which automatically guarantees dependencies and concurrency. Specifically, the vertices with no dependencies (e.g., communication/computation vertices and loading/computation vertices) can execute concurrently on the workers/servers, which fully improves resource utilization. The DAG interface can be used by ML developers to customize the IO, computation and communication processes to maximize the overall efficiency of the tasks. Besides, all kinds of model sync-up techniques can be supported, for instance, SSP can be very easily achieved by programming API (Figure 6), and even more, the modes of BSP,

```
ParameterVector mServerParam, mServerGrad;
DataVector mWorkerParam, mWorkerGrad;
tr1::shared_ptr<SubDataset> mSubDatasetPtr;
// core code
bool Iterate()
{
   mServerGrad.Reset(0);
   // how many steps before an all-worker sync
   for (int i = 0; i < mConf.nSync; ++i)
   {
      // determine if need to pull parameter from server
      if (0 == i % mConf.nPull)
      {
          mWorkerParam = mServerParam;
          mWorkerGrad.Reset(0);
      }
      DatasetCal("CalculateGrad", *mSubDatasetPtr,
         CalcNDArrayHelper().AddNDArray(mWorkerPara
         m).AddNDArray(mWorkerGrad));

      // determine if need to push grad from server
      if (0 == i % mConf.nPush)
      {
         mWorkerGrad.ReduceTo(mServerGrad,
          "UpdateParam",
          CalcNDArrayHelper().AddNDArray(mServerGrad
          ).AddNDArray(mServerParam) );
       }// end if
   }// end for
   SyncBarrier(); // do synchronization
   // other codes
}// end Iterate()
```

Figure 6: An implementation of bounded delay ASGD algorithm using C++ API of PS-Core. "nPush": how many steps before a push operation; "nPull": how many steps before a pull operation.

ASP, and SSP can be arbitrarily switched in the same algorithm. All these are performed through the Iterate() function which can be overridden by users-defined algorithm logic.

Load Balancing: Backup instance mechanism is designed to deal with the straggler problem. Specifically, when a node is identified as a slow-node (e.g., running time is larger than $1 + p$ average-time, $p \in [0, 1]$), coordinator will mark it as a "dead" node, request Fuxi to restart a new node and move the workload to it. Empirically, this mechanism can generally bring 10-20% efficiency improvement. In addition, to avoid possible resource deadlock, coordinator will release the resources that have been assigned to PS-Core if waiting time for resources allocation exceeds a pre-defined limit.

Communication: Communication efficiency is a key indicator for a distributed ML system. To improve it, we employ optimization for the communications of sparse data with different degrees of sparsity and deep optimization of the communication message partitioning for different data structures such as vectors, matrices, and user customized objects. To further improve the flexibility and reduce the development cost, we also provide point-to-point communication interface among worker nodes and more semantical communication interfaces which are compatible with MPI, such as AllReduce, ReduceTo, and Bcast. All these flexible communication interfaces are all armed with the efficiency optimization techniques described above, such that developers can achieve similar performance with much fewer codes.

Programming API: Figure 6 shows how to implement a bounded delay ASGD algorithm via C++ API. Specifically, "SubDataset" represents samples read from DFS or streaming data system. "ParameterVector" is the abstraction of the distributed vector in the server nodes; "DataVector" is a local vector hosted in the worker nodes. "CalculateGrad" and "UpdateParam" are user defined operators to complete gradient calculation that is computed by all the worker nodes and parameter updating which is done by server nodes. Push and pull is achieved by overloading assignment operator and "ReduceTo". Users only need to care about specific algorithm logics, such as data loading, initialization, communication, while failover is implemented by the framework. Through these interfaces, the goal of giving the programmer similar impression to coding for a single computer can be easily achieved.

3 KUNPENG ALGORITHMS

In this section, we will present the development of distributed ML algorithms that have been extensively deployed on a series of scenarios in Alibaba and Ant Financial. Firstly, we introduce an online learning algorithm called bounded delay FTRL-Proximal [21] with trust region which can handle abundant training data using comparatively small computing resourcesl. To our best knowledge, it is the first distributed implementation of such a large-scale FTRL-Proximal algorithm. We then briefly describe KunPeng-MART that reliably scales MART to support hundreds of billions of samples and thousands of features in production. Meanwhile, we show that the frequently used ML algorithms in industries can be easily implemented on KunPeng.

3.1 Bounded Delay FTRL-Proximal with Trust Region

For large-scale data stream, online algorithms for generalized linear models (e.g., logistic regression) enjoy many advantages. Specifically, they can quickly adjust the model according to user feedbacks, thus enable the model to reflect user's intentions and preferences in real time and improve the accuracy of online prediction. Owing to high efficiency of the FTRL-Proximal algorithm [21], it is primarily adopted for predicting the CTR of items (e.g., search ads, recommended products, feeds, etc.), which is crucial to most of internet companies. To introduce the FTRL-Proximal algorithm, $g_t \in \mathbb{R}^d$ is used to denote a vector, where t specifies the index of the current training instance and $g_{1:t} = \sum_{s=1}^{t} g_s$. The i-th element in vector g_t is denoted as $g_{t,i}$. Given these notations, the FTRL-Proximal algorithm iteratively updates the model w by the following formula:

$$\begin{aligned} w_{t+1} = \arg\min_{w}(g_{1:t}w + \frac{1}{2}\sum_{s=1}^{t} \delta_s \parallel w - w_s \parallel_2^2 \\ + \lambda_1 \parallel w\|_1 + \lambda_2 \parallel w - w_s \|_2^2), \end{aligned} \tag{1}$$

where (1) $g_{1:t}$ is the sum of the gradients of the loss functions (for all the examples from index 1 to t) with respect to w; (2) δ_s is defined in terms of the learning-rate schedule. Specifically $\delta_t = \frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}$, such that $\delta_{1:t} = \frac{1}{\eta_t}$. And $\eta_{t,i}$ is per-coordinate learning rate as $\eta_{t,i} = \frac{\alpha}{\beta + \sqrt{(\sum_{s=1}^{t} g_{s,i}{}^2)}}$, where $\beta = 1$ is usually good enough and α depends on the features and datasets; (3) λ_1 and λ_2 are regularization parameters.

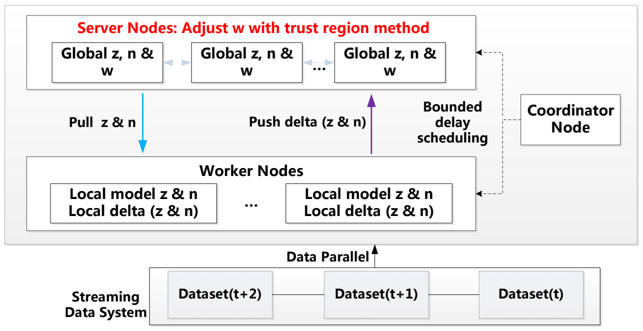


Figure 7: The workflow of FTRL-Proximal for online learning based on KunPeng.

As reported in [21], a recursive closed-form solution on a per-coordinate base can be obtained for FTRL-Proximal. As a result, it can be efficiently implemented for non-distributed settings. However, it is not easy to efficiently implement it in a distributed setting and keep its convergence, especially when there are billions of features. For a distributed setting, BSP is inefficient since the workers need to wait for communication whereas ASP may not converge [14]. Therefore, we made an intuitive modification to FTRL-Proximal, which uses bounded delay methods (its accuracy can be theoretically preserved [14, 18]) to fulfill consistency models and adjust the w parameter with trust region. Empirically, this method is more stable and efficient in online learning scenarios.

The specific approach is as follows: (1) workers maintain local models, i.e., z and n (through a bounded-asynchronous way to keep pace with the servers), and update local models following the standalone program. At the same time, workers use δ_z and δ_n to keep track of the increments of the local models, and send them to server nodes for updating the parameter vector; (2) servers add up all model increments from workers and update the global models, then adjust w parameter at the end of each iteration to ensure that the target model w is in the trust region so that the algorithm will not diverge.

We design two strategies of trust region, (1) **rolling back**. That is, if the element of w is outside the trust region, all of the current parameters (w, n, z) will be rolled back to the corresponding latest normal backup, and (2) **projection**. That is, we project the element of w to trust region if it is out of trust region.

Based on the KunPeng SDK, the workflow of the distributed bounded delay FTRL-Proximal is shown in Figure 7. At the beginning, all local and global parameters are initialized as zeros. Workers then load data-partitions from streaming data systems, and divide data into smaller mini-batches. After that, workers pull the working set of z and n, and then calculate gradients using the mini-batch samples and update local parameters correspondingly. Then, workers push the local delta parameters instead of gradients to the servers. Servers provide access to global models via push and pull interfaces, and enforce incremental accumulation to global z and n. Coordinator performs bounded delay scheduling to coordinate workers and servers to complete optimization task. According to the above workflow, the code snippet in Figure 6 enables the whole algorithm implementation.

We find that the data are often extremely sparse (e.g., parameter per min-batch only account for 0.01% of the full parameter) in real online learning scenarios. The sparse representation approach based on mini-batch can additionally improve the performance of communication and reduce resource consumptions. Due to these optimizations on the algorithm and sufficient engineering efforts, FTRL-Proximal implemented on KunPeng can handle tens of billions of samples and tens of billions of features within several 10s of minutes with small computing resources.

3.2 Multiple Additive Regression Trees (MART)

MART and its extensions, such as LambdaMART, are popularly used in industry practices [5], because of their high adaptability, practical effectiveness, and strong interpretability. Here we introduce KunPeng-MART [35], a parameter server based distributed learning system that reliably scales MART to support hundreds of billions of samples and thousands of features: (1) We built a general MART system which enjoys efficient sparse communication, better fault tolerance, and etc; (2) We implemented a distributed LambdaMART with a query id based partitioning strategy. Due to these techniques, the system can scale near linearly with the number of worker nodes, and it is the fastest and the most stable MART implementation to the best of our knowledge. Due to the limit of space, we do not introduce the background knowledge of MART algorithms and only describe the implementation of KunPeng-MART.

3.2.1 MART. To overcome the challenge of the huge storage need, KunPeng-MART employs a data parallelization mechanism. Specifically, each worker only stores a subset of the whole dataset for each feature. Under this mechanism, the main workflow for splitting a node is as follows: (1) Each worker calculates the local weighted quantile sketch for the subset of the nodes stored on this worker; (2) Each worker pushes the local weighted quantile sketch to servers. The servers merge the local sketches to a global weighted quantile sketch by a multiway merge algorithm, and then use it to find the splitting value; (3) Each worker pulls the splitting value from servers and splits samples to two nodes.

The details on producing local weighted quantile sketches and merging local sketches to a global sketch can be found at [6]. Repeating the above procedure can build a tree. In this way, we can build trees one by one following the gradient boosting framework to get a MART. Although the key idea is straightforward, there exists one key challenge that the computation and communication cost of split-finding algorithm will become very high when the number of features gets bigger or the tree gets deeper. To remedy this issue, we leverage the communication schema of KunPeng to reduce the cost of merging local sketches. This can further speed up the whole split finding algorithm.

3.2.2 LambdaMART. For LambdaMART, the procedure of building trees is the same as the general MART algorithm. However, the training set is composed of pairwise examples from each query group, which may cause very high communication cost when samples are irrationally distributed across workers. To avoid this problem, we propose two methods to divide the training samples appropriately.

The first method is to take a multiway number partitioning algorithm [16] to divide the query ids to different workers. Then, samples with the same query id are divided to the same worker where it belongs to. As a result, the training samples are divided as evenly as possible, and the workload on workers is balanced. The detailed steps are listed as follows: (1) For each query id, scan the whole dataset to count the number of samples with it; (2) Each worker calculates the partition according to the multiway number partitioning algorithm; (3) Each worker loads the training samples with query ids that belongs to the worker.

The second method is an approximate method, in which we need to store the samples of the same query id continuously in the file system. Then each worker loads evenly divided data as usual. If samples with the same query id are divided into two workers, they will be treated as samples with two different query ids. Even though the approximate method could lose some pairwise samples, it empirically works well for most of our tasks.

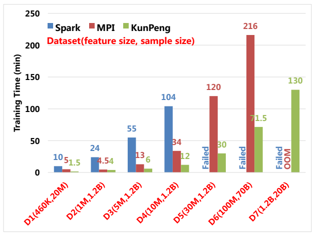
3.3 Other Deployed Algorithms

We have also implemented many other distributed ML algorithms on KunPeng, some of which have been deployed in production. We only mentioned some algorithms which are universally adopted in Alibaba and Ant Financial[4]: (1) Large-scale sparse Logistic Regression (LR) which is used for CTR prediction in a variety of scenarios due to its simplicity, scalability, and practicality. To solve LR, we implement L-BFGS, more specifically, Orthant-Wise Limited-memory Quasi Newton (OWL-QN) [3], and a Block Coordinate Descent (BCD) algorithm [17] based on simultaneous data-and-model-parallel approach. In the implementation of BCD, the coordinator of PS-Core selects a random block of parameters to be updated in parallel according to some rules (e.g., prioritization and random), workers compute gradient vectors, and diagonal Hessian matrix on a subset of datasets and pull/push the block of parameters with servers accordingly; (2) Distributed Factorization Machines, denoted as KunPeng-FM, which is remarkably better than existing models in terms of both model learning and expressiveness (easy to represent any user-based or item-based features in the model) [23]. To optimize KunPeng-FM, we adopt the AdaGrad [10] method in which workers compute distributed gradients on mini-batches asynchronously; (3) Deep learning framework based on CPU cluster. When applying deep learning to industrial tasks, there are bottlenecks in terms of both the sizes of the model and the training data. In order to tackle this problem, we build a generalized CPU-based large-scale deep learning platform based on a deep integration of Caffe with KunPeng, which has the advantages of both. In addition, it enables easily implementing new DL methods, e.g., Deep Semantic Similarity Model (DSSM) [15] with SGD and SSP.

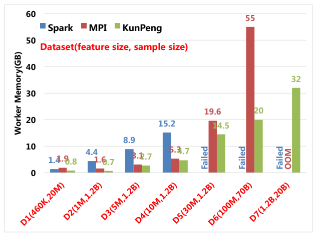
4 PERFORMANCE

In this section, we compare algorithms on KunPeng with the state-of-the-art methods to demonstrate the scalability and efficiency of KunPeng[5]. Specifically, we compare the LR algorithm implemented on KunPeng with Spark and MPI, then KunPeng-MART with XGBoost, finally KunPeng-FM with LibFM and DiFacto. All the comparisons are conducted on a production cluster, which has 5000 servers with each machine consisting of two twelve-core Intel Xeon CPU E5-2430 (2.2 GHz) and 96 GB of memory.

[4]Some other algorithms developed on KunPeng are not included here, e.g., LDA, which has been proven to be very effective in handing up to 100 million document/vocabulary size and 10K topics (totally 1 trillion parameters), and is on par with the largest scale of LDA implementation reported in [30]. However, it has not been deployed in real application yet.



(a) Training time



(b) Per-worker memory cost

Figure 8: The comparisons of training time and per-worker memory cost among Spark, MPI, and KunPeng.

4.1 Experiments

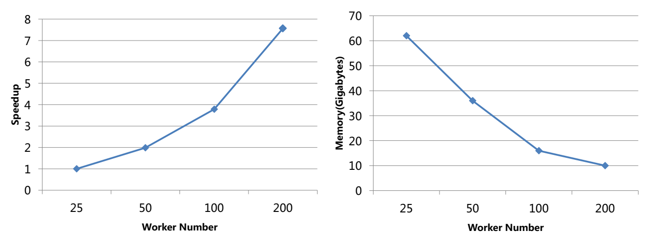
4.1.1 Sparse Logistic Regression. We take LR for example to demonstrate the scalability and efficiency of KunPeng because it is widely applied for CTR prediction on most ML platforms. Please note that similar results on scalability and efficiency comparison are also applied to other algorithms.

We conduct experiments on seven real-world datasets for CTR prediction tasks in recommendation scenarios in Alibaba and Ant Financial, among which four are from Ant Financial (D2, D3, D4, D5) and the others are from Search Ads in Taobao (D1, D6, D7). We compare LR on KunPeng, with LR implemented on Spark and MPI. The memory history parameter of L-BFGS is set to 10 for all systems and the experiments are concurrently conducted using the same CPU resources on one production cluster when having enough resources.

Figure 8 summarizes the training time and worker average memory consumption respectively, from which we can observe that KunPeng-LR outperforms Spark-LR and MPI-LR significantly in both scalability and efficiency. Firstly, Spark ran out of the available memory when feature size scales to tens of millions, due to coarse-grained primitives and JVM-based methods of Spark. Secondly, compared with Spark-LR, MPI-LR achieves better efficiency and consumes less memory, but is still much slower than KunPeng-LR, especially when the number of samples and features is extremely large. Specifically, when the feature size increases to 1.2B, MPI-LR also runs out of memory. Finally, KunPeng is about 2-9 times faster than other platforms with 1/4-1/2 memory, according to Figure 8. This advantage is more significant when the model size gets larger.

We then conduct experiments on another real dataset from Cainiao logistics platform that contains about 18 billion samples and 7 billion features by varying worker numbers in [25, 200] using LR algorithm (the optimization algorithm is OWL-QN with BSP). The baseline systems cannot handle so many features, thus

[5]We omit FTRL-Proximal in this section, since we do not find any distributed implementation of large-scale FTRL-Proximal algorithm in literature currently.



(a) Speedup (b) Memory usage

Figure 9: Speedup and per-worker memory consumption of KunPeng with LR algorithm using L-BFGS solver (feature size ≈ 7 billion, sample size ≈ 18 billion).

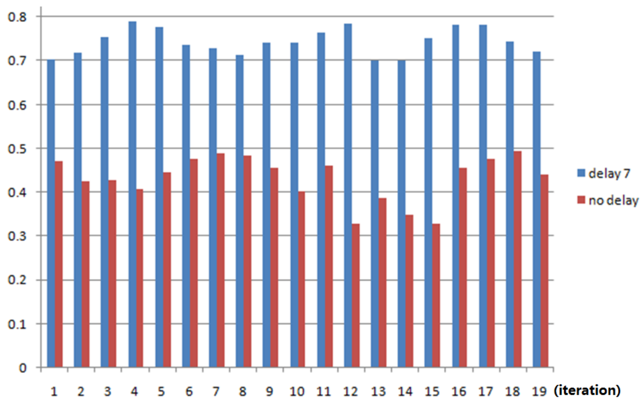


Figure 10: The proportion of calculation in total time.

we focus on the speedup and memory consumption of KunPeng itself on a production cluster. As reported in Figure 9, 25 workers are able to handle problems with ten billions of features. We can also see that KunPeng has near-linear speedup with the increase of workers. Specifically, KunPeng achieves 1.97x speedup and about 0.63x memory consumption reduction when worker number increases from 100 to 200. The major reasons are three folds: (1) Sparse representation greatly reduces resource consumption and the cost of network communication; (2) KunPeng effectively keeps away from the blocking of stragglers in each iteration; (3) KunPeng uses fine-grained memory lock and multi-thread strategies to further reduce the computational cost. Furthermore, we also test BCD algorithm (as described in section 3.3) with staleness on this dataset and show the result in Figure 10. Compared with BSP mode ("no delay"), the proportion of calculation time in each iteration is significantly improved ("delay=7" denotes staleness bound =7), which means the proportion of communication time is significantly reduced. However, due to the cost of calculating and pushing the Hessian matrix, we do not observe a significant reduction in overall training time over L-BFGS on this dataset.

4.1.2 KunPeng-MART. In this subsection, we report the performance of our KunPeng-MART on four industrial datasets in terms of CTR and CVR (click value rate), which are summarized in Table 1. We also successfully tested even bigger datasets with hundreds of billions of samples, which XGBoost cannot handle at all.

Table 1: Datasets descriptions and statistics used in MART experiments.

Dataset	Description	#Features	#Sample number (million)
Rec CTR	Items recommendation for sale	78	84
Search CTR	Search results ranking	592	1,000
Ads CVR1	Search ads ranking	698	2,067
Ads CVR2	Search ads ranking	698	10,245

Table 2: Comparison of peak memory usage on training sets.

Dataset	KunPeng-MART	XGBoost
Rec CTR	162G	176G
Search CTR	1,775G	1,810G
Ads CVR1	3,711G	3,912G
Ads CVR2	22,157G	N/A

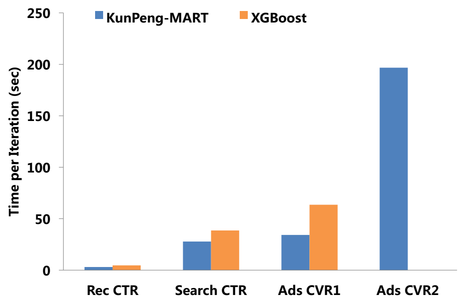


Figure 11: Runtime comparison on four training sets.

We adopt the well-known XGBoost as our baseline. Since all the compared methods can achieve accuracy without statistically difference, they are mainly compared on memory consumption and computational time which are two key factors in the production environment. To make a fair comparison, all algorithms adopt the same experimental setup. Specifically: tree number=500, max depth=6, tree method=approximate split finding, min child weight=100, data and feature sample rate=1.0, and bin number=128.

Memory consumption: The peak total memory used is summarized in Table 2, which shows KunPeng-MART consumes slightly less memory than XGBoost. Moreover, we tried XGBoost on Ads CVR2 more than ten times and never succeed. This is because when the dataset is very large many computation nodes are needed so that the failure probability of some nodes is very high. Note that Rabit only offers limited fault tolerance for XGBoost and thus can not recover when the tracker node fails. In contrast, our KunPeng-MART system supports robust failover mechanism so that it can recover easily.

Computational time: Figure 11 shows the running time of all the algorithms on all the datasets. From it, we observe that KunPeng-MART is much more efficient than XGBoost. This is because that KunPeng enjoys sparse communication and traffic control optimizations, which make it more efficient than Rabit. Thus the waiting time of KunPeng-MART is extremely reduced and the CPU resources are fully utilized.

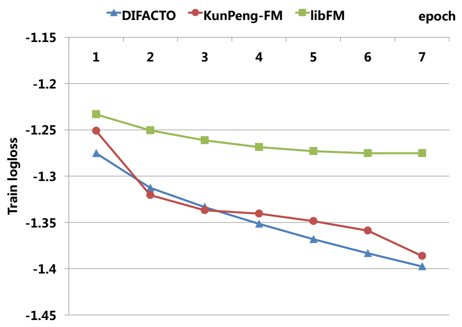


Figure 12: A single-machine comparison with libFM and DiFacto on a small ads dataset (feature size≈0.48M; training sample size≈0.8M; 1 epoch = 0.8M samples).

Table 3: Statistics of KunPeng tasks in a production cluster.

Job Number	Worker Number	Input Data (Gigabytes)
7,012	2,247,681	141,191,630

Finally, we would like to emphasize that when the scale of dataset becomes larger, more computation nodes are needed so that the failure probability of some nodes is highly increased. However, if the tracker node failed for XGboost, the whole job can not be recovered quickly, which makes it unsuitable for extremely big training data. Conversely, our KunPeng system dumps the intermediate models and local variables into DFS, thus the training can be recovered from the nearest iteration easily. Due to these differences, KunPeng-MART has higher scalability and robustness than XGBoost, and thus it is more suitable for industrial production applications.

4.1.3 Factorization Machines. We implemented KunPeng-FM with AdaGrad solver using about 300 lines of codes, which shows the flexibility and conciseness of KunPeng API. We compared KunPeng-FM with both DiFacto[6] [19] and libFM [22] on a single machine on 0.8 million samples with 0.4 million features (set the SGD min-batch size to 1000 and the learning rate to 0.01 for all systems). The result is shown in Figure 12. As we can see, DiFacto outperforms libFM significantly in terms of training loss, but KunPeng-FM is very close to it. This indicates that we can achieve good performance with small amount of codes by fully utilizing the flexibility of KunPeng.

5 APPLICATIONS

KunPeng plays an important role in transforming Big Data into actionable knowledge and has been deployed in Alibaba and Ant Financial's production systems since 2014. Table 3 illustrates the statistics of our total ML training tasks on KunPeng based on the trace-logs collected from one production cluster in a short period of time. The workload was arranged onto roughly 2.2 million workers. Approximately 140 petabytes of input data were processed by the system. The largest single task required about 3000 workers with more than 200 billion samples and more than 10 billion features.

[6]Since we can not find any its distributed implementation currently, we only compare KunPeng-FM with the single machine version of DiFacto using the code from https://github.com/dmlc/difacto (version 78e3562).

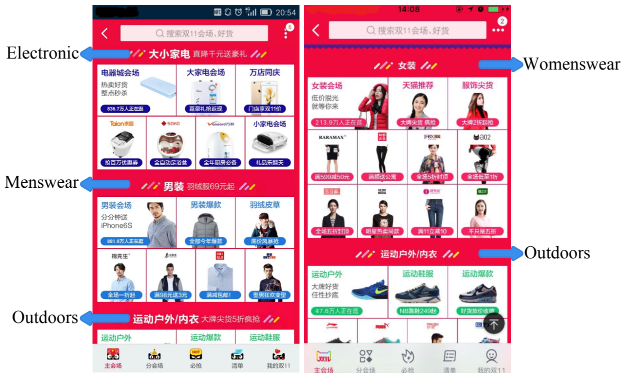


Figure 13: The personalized "floor" rank venue of 2015 Double 11. Left: result for user A. Right: result for user B.

Next, we illustrate some important applications of KunPeng in Alibaba and Ant Financial, including "floor" recommendation (in Taobao) for Alibaba's Double 11 Festival, CTR prediction for Taobao's search ads, and estimation of online transaction risks for Ant Financial.

The Alibaba's Double 11 Online Shopping Festival has been held annually on November 11 since 2009, which creates an innovative retail experience for consumers. In order to better enhance the user experience, a near realtime learning pipeline was constructed in 2015 Double 11 Festival, which applied bounded delay FTRL-Proximal with trust region to precisely match buyers and items. Specifically, user's behavior logs (mainly shopping actions) are collected in real time. After data preprocessing and feature engineering (e.g., feature combination and discretization), the generated training data during a time window (e.g., half hour) may contain tens of billions of features at a peak and then been dumped into data-streaming and processing services. The training data is exploited by KunPeng to get a near realtime model to reflect a better understanding of user's preferences and intentions. To better understand the above workflow, we show the personalized "floor" rank venue in Figure 13, where "Electronic", "Menswear" , "Womenswear", and "Outdoors" are "floors" that contain some related shopping contents. For Double 11 Festival, the FTRL-Proximal algorithm is utilized to predict user's preference for each "floor", and returns the most relevant "floors" to help users make purchasing decisions. In this scenario alone, applying near realtime learning increased CTR by more than 52 percent on Double 11 day.

In Taobao's search ads, we aim to make the original LR model more scalable, stable, and exploit more data and update faster. To achieve these goals, we use simultaneous model-and-data-parallelism to distribute tens of billions of model parameters into a large set of servers, and divide hundreds of billions of training samples into many workers for parallel training with robust failover mechanisms. The model training can be made even faster using sparse representation, practical straggler handling, fine-grained memory lock strategies, and communication optimization. All these approaches together create better matches between items and a particular user in a given context. As a result, the advertising revenue is greatly increased.

Alipay under Ant Financial is an important online payment platform in China, which has hundreds of millions of users, and generates billions of online transactions per day. The ability of detecting and preventing fraud transactions is very important in Ant Financial. Random Forest (RF) has been broadly adopted to estimate online transaction risks and achieved impressive results in industrial practice. Based on KunPeng-MART which scales GBDT to hundreds of billions of samples and thousands of features, the traditional RF model is migrated to large-scale GBDT model for improving the performance. Then, the concatenation of boosted decision trees and deep neural network classifier, inspired by Facebook's "GBDT + LR" method [13], is used to further boost GBDT. Consequently, deploying all the methods in production significantly improves the capabilities for early detection of fraud transactions and provides users within a highly secure environment.

6 CONCLUSION AND FUTURE WORK

We presented KunPeng, which is a parameter server based highly robust distributed learning system with an easy-to-use interface. We showed that KunPeng can scale to tens of billions of features and hundreds of billions of samples in online and offline scenarios. We also verified that KunPeng is robust, flexible, scalable, and efficient in a production cluster. We have shown encouraging results on several real-world applications. Better supports for GPU-based DL and graph algorithms are our future work.

ACKNOWLEDGMENTS

We would like to acknowledge contributions from our colleagues from Alibaba and Ant Financial, including: Xiaoyuan Cui, Yingda Chen, Wei Chu, Yang Gu, Jun Huang, Xu Hu, Minghui Qiu, Shenquan Qu, Ang Wang, Huaidong Xiong, Jiaxing Zhang, Xiaolin Zhang, and Youzhi Zou. We would also like to extend our sincere thanks to the entire Large-Scale-Learning and Distributed-Learning-and-Systems team members. In particular, we appreciate the feedback we have received from the many users of KunPeng. Finally, we thank KDD anonymous reviewers for their helpful suggestions.

REFERENCES

[1] 2016. MIT Technology Review. https://www.technologyreview.com/s/602850/big-data-game-changer-alibabas-double-11-event-raises-the-bar-for-online-sales/. (2016). Accessed Feb 12, 2017.

[2] Rami Al-Rfou and others. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688* (2016).

[3] Galen Andrew and Jianfeng Gao. 2007. Scalable training of L 1-regularized log-linear models. In *Proceedings of the 24th International Conference on Machine Learning*. ACM, 33–40.

[4] Badri Bhaskar and Erik Ordentlich. 2016. Scaling Machine Learning To Billions of Parameters. *Spark Summit* (2016).

[5] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[6] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 785–794.

[7] Thomas H Cormen and Michael T Goodrich. 1996. A bridging model for parallel computation, communication, and I/O. *Comput. Surveys* 28, 4es (1996), 208.

[8] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, and others. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*. 1223–1231.

[9] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[10] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[11] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[12] Joseph E Gonzalez, Reynold S Xin, Ankur Dave, Daniel Crankshaw, Michael J Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In *OSDI*, Vol. 14. 599–613.

[13] Xinran He and others. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. ACM, 1–9.

[14] Qirong Ho and Others. 2013. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems*. 1223–1231.

[15] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. ACM, 2333–2338.

[16] Richard E Korf. 2009. Multi-Way Number Partitioning. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence*. Citeseer, 538–543.

[17] Mu Li and others. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, Vol. 14. 583–598.

[18] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. 2014. Communication efficient distributed machine learning with the parameter server. In *NIPS*. 19–27.

[19] Mu Li, Ziqi Liu, Alexander J Smola, and Yu-Xiang Wang. 2016. DiFacto: Distributed factorization machines. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 377–386.

[20] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. 2012. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment* 5, 8 (2012), 716–727.

[21] H Brendan McMahan and others. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1222–1230.

[22] Steffen Rendle. 2012. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology* 3, 3 (2012), 57.

[23] Steffen Rendle and Lars Schmidt-Thieme. 2010. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*. ACM, 81–90.

[24] RohitShetty. 2011. hot-standby. https://www.ibm.com/developerworks/community/blogs/RohitShetty/entry/high_availability_cold_warm_hot?lang=en. (2011). Accessed Feb 12, 2017.

[25] Alexander Smola and Shravan Narayanamurthy. 2010. An architecture for parallel topic models. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 703–710.

[26] Vinod Kumar Vavilapalli and others. 2013. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 5.

[27] Tom White. 2012. Hadoop-The Definitive Guide: Storage and Analysis at Internet Scale (revised and updated). (2012).

[28] Eric P Xing and others. 2015. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data* 1, 2 (2015), 49–67.

[29] Yuan Yu and others. 2008. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, Vol. 8. 1–14.

[30] Jinhui Yuan, Fei Gao, Qirong Ho, Wei Dai, Jinliang Wei, Xun Zheng, Eric Po Xing, Tie-Yan Liu, and Wei-Ying Ma. 2015. LightLDA: Big topic models on modest computer clusters. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 1351–1361.

[31] Hyokun Yun, Hsiang-Fu Yu, Cho-Jui Hsieh, SVN Vishwanathan, and Inderjit Dhillon. 2014. NOMAD: Non-locking, stOchastic Multi-machine algorithm for Asynchronous and Decentralized matrix completion. *Proceedings of the VLDB Endowment* 7, 11 (2014), 975–986.

[32] He Yunlong, Sun Yongjie, Liu Lantao, and Hao Ruixiang. 2016. DistML. https://github.com/intel-machine-learning/DistML. (2016). Accessed Feb 12, 2017.

[33] Matei Zaharia and others. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 2–2.

[34] Zhuo Zhang, Chao Li, Yangyu Tao, Renyu Yang, Hong Tang, and Jie Xu. 2014. Fuxi: a fault-tolerant resource management and job scheduling system at internet scale. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1393–1404.

[35] Jun Zhou, Qing Cui, Xiaolong Li, Peilin Zhao, Shenquan Qu, and Jun Huang. 2017. PSMART: Parameter Server based Multiple Additive Regression Trees System. Accepted. In *Proceedings of the 26th International Conference on World Wide Web*. ACM.