2017 14th International Symposium on Pervasive Systems, Algorithms and Networks & 2017 11th International Conference on Frontier of Computer Science and Technology & 2017 Third International Symposium of Creative Computing

DPS: A DSM-based Parameter Server for Machine Learning

Chenggen Sun*, Yangyang Zhang*, Weiren Yu*, Richong Zhang*, Md Zakirul Alam Bhuiyan†, Jianxin Li*
*School of Computer Science and Engineering
Beihang University
Email: {suncg, zhangyy, yuwr, zhangrc, lijx}@act.buaa.edu.cn
†Department of Computer and Information Sciences
Fordham University
Email: mbhuiyan3@fordham.edu

解决学习过程中模型参数的有效存储和更新问题

***Abstract*—To solve the problem of efficient storing and updating of model parameters in the learning process, the parameter server is concerned as a high-throughput distributed machine learning (ML) architecture with the emergence of big models with billions of parameters. Current parameter servers, such as the Parameter Server and the Petuum, do not address data management and lack high-level data abstraction. Moreover, they have no task scheduling and do not fully utilize the computing resource as well as possibly lead to load imbalance. Their programming interface is too complicated and they do not support data flow operations (e.g. $map/reduce$) which are very useful for data preprocessing. These drawbacks limit the performance and ease of use of such parameter servers.**

In this paper, we proposed DPS, a parameter server based on Distributed Shared Memory (DSM) for machine learning. DPS provides flexible consistency models, high-level data abstraction and management that support data flow operations, lightweight task scheduling system and user-friendly programming interface to solve the problems of existing systems mentioned above. The experimental results show that DPS can reduce networking time by about 50%, and achieve up to 1.9x performance compared to Petuum while the algorithms implemented on DPS use less code than those implemented on Petuum.

当前的PS和Petuum等不处理数据管理，缺乏高级数据抽象。

没有任务调度，没有充分利用计算资源，可能导致负载不平衡

这些缺点限制了性能

在本论文中，提出了一个基于分布式共享内存的PS框架。

DPS提供了一种灵活的一致性模型

I. Introduction

In recent years, big data technology has developed rapidly, which causes a great upsurge of research in the world and attracts great attention in industry, academia and governments. Big data often implicates a lot of deep knowledge and hidden value that are not available in small data. However, the deep value of big data is difficult to find by simple analysis only. It usually needs complex intelligent analysis based on machine learning (ML) and data mining.

There are two challenges for big data machine learning: one is big data and data of training sample is huge.The other one is the big model and big data machine learning is often used to process big data with hundreds of billions of samples and billions of features. For example, Seti[1] project use up to hundreds of billions of samples, and billions of features of data for machine learning, and Tencent Peacock[2] topic model analysis system needs up to billions of documents, millions of words, millions of topic for model training. Such a large number of training samples and model parameters, coupled with the complexity of the machine learning algorithm itself, bringing a very serious computational performance problem.

Therefore, with the rapid growth of data scale and the emergence of the large model of billions of parameters, efficiently storing and updating the model parameters in machine learning is challenging. As a high throughput distributed machine learning architecture, the parameter server attracts more attention. A Parameter Server is essentially a distributed shared memory system that enables clients to easily share access to the global model parameters stored in multiple servers via a key-value interface, in a logical bipartite server-client architecture for storing model parameters.

Current parameter servers, such as the Parameter Server and the Petuum, do not address data management and lack high-level data abstraction. Moreover, they have no task scheduling, do not fully utilize the computing resource and possibly lead to load imbalance. Their programming interface is too complicated and they do not support data flow operations (e.g. $map/reduce$) which are very useful for data preprocessing. These drawbacks limit the performance and ease of use of such parameter servers.

工作调度

We proposed DPS, a new parameter server based on Distributed Shared Memory (DSM) for machine learning. DSM is a technology that provides a global view of the cluster, so users can program on it as if on a single machine. DPS is built based on a well-performed DSM system, Grappa[3]. The key contributions of DPS are as follows:

1) Flexible consistency models. DPS provides flexible consistency models such as Bulk Synchronous Parallel (BSP), Stale Synchronous Parallel (SSP).

2) Data management and high-level data abstraction. Data is abstracted as $GlobalTables$ which is used to load and transform training data. The $GlobalTable$ provides data flow operations such as Map, $Reduce$, $GroupBy$, et.al. Users can easily preprocessing training data and do feature engineering using $GlobalTable$.

3) Lightweight task scheduling. To overlap computation with network communication and utilize computing resource and network resource better, a lightweight task scheduling system is implemented. Additionally, training process is split into many small tasks to reduce network waiting time.

4) User-friendly programming interface. The system mainly provides three kinds of programming interface. 1) Data APIs,

2375-527X/17 $31.00 © 2017 IEEE
DOI 10.1109/ISPAN-FCST-ISCC.2017.48

such as $LoadData$, Map, $Reduce$, $GroupBy$, et.al. 2) PS APIs, $CreateTable$, Get, $Update$. 3) Tasking APIs, Run, Complete, Wait. All of these APIs are designed briefly. With these APIs, users can quickly implement various machine learning applications on our system.

The rest of the paper is structured as follows. In Section II, we introduce the background. Section III describes the design and implementation of DPS system. In Section IV, then perform a comprehensive evaluation and analysis on DPS. Section V presents the related work. Finally, conclusion and future work are given in Section VI.

II. Background and Motivations

In this section, parameter server and some problems of the current parameter server systems are introduced, thus lead to the motivation of the proposed system, and then introduce the related backgrounds of consistency model and distributed shared memory.

A. Parameter Server

Many machine learning algorithms often face the problem of efficiently storing and updating of model parameters in the learning process. To effectively cope with and meet the needs of such machine learning algorithms in big data scenarios, the researchers have proposed an architecture called Parameter Server (PS). A Parameter Server is essentially a distributed shared memory system that enables clients to easily share access to the global model parameters stored in multiple servers via a key-value interface.

There are several parameter server systems proposed recently, e.g. Parameter Server[4], Petuum[5], Project Adam[6]. The Parameter Server[4] manages asynchronous data communication between nodes and supports flexible consistency models, elastic scalability, and continuous fault tolerance. Petuum [5], implements the Stale Synchronous Parallel (SSP) consistency model, which reduces network synchronization and communication costs while maintains bounded-staleness convergence guarantees. However, all these systems have problems. 1) They do not address data management and lack high-level data abstraction. It is hard to do data transformation and feature engineering which are necessary for general machine learning systems. 2) They have no task scheduling, cannot fully utilize the computing resource and possibly lead to load imbalance. 3) Their programming interface is very complicated. They do not support data flow operations (e.g. $map/reduce$) which are very useful for data preprocessing. Without data management, users have to partition the data according to the number of servers first, then process the data worker by workers, partition by partition. Writing applications on these systems is cumbersome. These parameters server's performance and ease of use both need to be improved.

B. Consistency Models

For parallel programs, there is a need to synchronize data on different threads. There are several different synchronous consistency models.

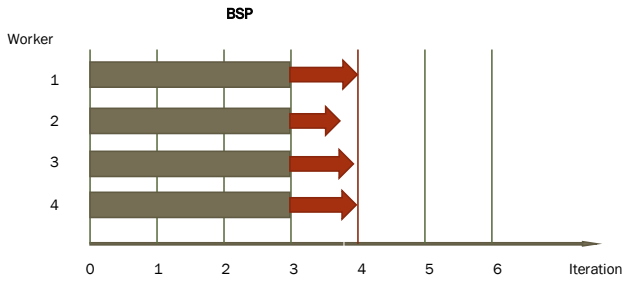


Fig. 1. Bulk Synchronous Parallel.

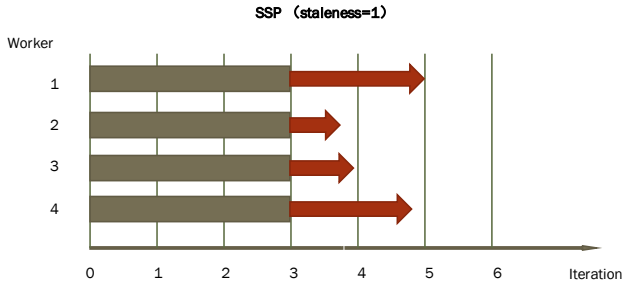


Fig. 2. Stale Synchronous Parallel.

1) Bulk Synchronous Parallel: Many parallel frameworks follow the Bulk Synchronous Parallel (BSP) consistency model. As shown in Figure 1, in BSP model, the application operates on a snapshot of the data that was produced by the previous iteration. To do this, all threads must execute the same iteration at the same time. A single iteration of BSP consists of three steps. In the computation phase, all threads compute on the previous iterations output in parallel. In the communication phase, threads produce new output and share it either by explicit communication, or write to a shared data structure. Lastly, in the synchronization phase, threads execute a barrier to ensure that they dont begin the next computation step until all other threads have finished the communication step. However, a well-known problem of BSP is the straggler problem, where each iteration proceeds at the speed of the slowest thread. This problem even gets worse as the parallelism is increased.

在计算阶段，所有线程在前面的迭代输出中并行计算。

在通信阶段，线程产生新的输出，并通过显式通信或写入共享数据结构来共享它

2) Stale Synchronous Parallel: Stale Synchronous Parallel (SSP) is a consistency model based on BSP. As shown in Figure 2, the SSP model allows workers to read and write parameters of stale version in the local cache (instead of taking a lot of time to wait for the latest parameters every time). It can be thought of as BSP with the addition of bounded staleness, read-my-writes, and soft synchronization. The design of SSP is to improve the iterative throughput as much as possible in the premise of the correct theory. The advantage of SSP is to avoid the stragglers slowing down the whole system, and ensure the correctness of the machine learning algorithm.

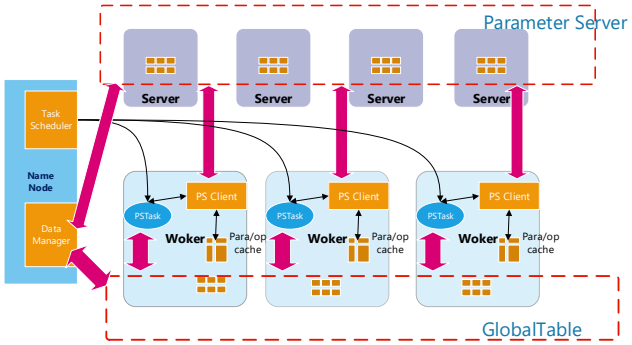


Fig. 3. DPS Architecture.

一个逻辑统一的全局地址空间

C. Distributed Shared Memory

DSM的特点是使用分布式内存架构，具有高扩展性，高通用性，高可移植性，易编程性等特点。

Distributed shared memory (DSM) is an important technology in the development of parallel computing. It is characterized by the use of distributed memory architecture with high scalability, high versatility, high portability, easy programmability and so on. The role of distributed shared memory is providing a logical unified global address space, where any node in the cluster can read and write directly on the unified global address space, making writing distributed parallel computing program as if on a single machine.

Grappa [3] is a recently proposed DSM system for data-intensive applications on commercial clusters. It is inspired by the MTA [7], [8], a custom hardware-based system. Like the MTA, grappa does not depend on localization to reduce memory access overhead. Instead, Grappa relies on parallelization, makes full use of processor resources and reduces the high cost of communication between nodes. Grappa also adopts the idea of the MTA shared memory and fine-grained parallel programming. To support fine-grained messages, Grappa aggregates small messages into large network packets, thus maximizes the utilization of commercial networks.

III. DPS Design and Implementation

In this section, the architecture of our DPS system and the implementation details of the key features are introduced, including SSP consistency model, lightweight task scheduler, $GlobalTable$ data abstraction and programming interface.

A. System Architecture

As shown in Figure 3, the system is mainly composed of PS Server, PS Worker, Data Manger and Task Scheduler. Each component is described below.

1. **PS Server** stores global shared parameters. The parameters are stored uniformly in the memory of each node. PS Server also stores the iteration round of each worker, and uses the SSP consistency model to control the synchronization of each worker.

2. **PS Worker** is the component for iterative training on each node.It stores the local parameters in parameter cache, using get and $update$ and other APIs to synchronize parameters with PS Server. It also provides an update buffer to aggregate the update operations before submitting to the server.

3. **Data Manger** is used to manage the data used in ML applications, partitioning the data automatically. It uses $GlobalTable$ data abstraction based on $GlobalAddress$ provided by Grappa, to implement map, $reduce$ and other data flow operation interfaces.

4. **Task Scheduler** is used to assign tasks based on the load of each node, and then schedule and manage tasks.

In addition, these components communicate through the message aggregation layer to make full use of the network bandwidth of the cluster.

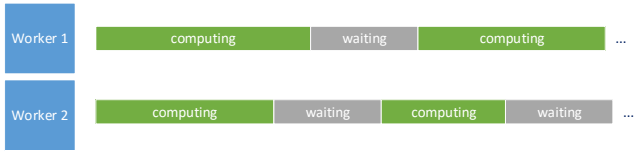


Fig. 4. Parameter server without task scheduling.

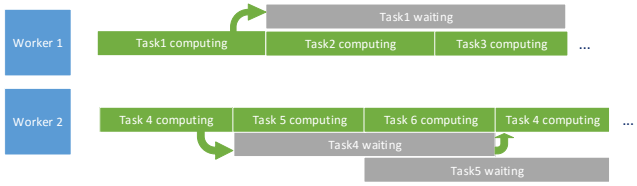


Fig. 5. Parameter server with task scheduling.

B. Parameter Server and SSP Consistency

The parameter server uses a distributed key-value store model to store parameters, providing an effective mechanism of synchronous model parameters between different workers. Each worker does not need to save all the parameters, but only needs to save a small subset on which the worker depends.

The proposed system DPS is a parameter server based on distributed shared memory system (DSM). In DSM, each node can access the global memory, and every node plays the same role. There is no specific role for server and worker and in the design of our parameter server, each node acts as both server which stores global parameters, and worker for task computation and storing local parameters.

在SSP中我们使用时钟同步参数

In the implementation of SSP model, we use clock to synchronize parameters.Each parameter appends a clock value to mark the age of the parameter and each worker also has a clock value to mark the iteration count of the worker. Usually the clock value of worker increase by 1 after each iteration. The worker's task can use the updates of parameter after clock $t - staleness$ at clock t (usually t round iterations). If the parameters stored on the server are too old, the task of clock t needs to wait until the other slower tasks are finished.

每个参数都附有一个时钟值，每一个Worker也有一个时钟值以标记迭代次数

通常每次迭代之后，时钟值加1

In DPS, each worker has a parameter cache caching local parameters. It stores the parameters obtained by the worker and the clock of the parameter. When a worker is to take

在DPS中每个工作节点都会有一个参数存储区来缓存本地参数，

当一个worker要获取参数时，它首先检查参数缓存，如果缓存中的参数不存在或者是过于陈旧，将会从远程服务器中获取参数

parameters, it first checks parameter cache. If the cached parameter does not exist or is too stale, it will fetch the parameter from the remote server.

一个更新缓冲区

In addition, each worker also has an update buffer. The updated value of the parameter computed on each worker (usually an incremental delta) will not be sent immediately to the remote server, but will be buffered in the update buffer first and will not be submitted to the server until some specified condition (e.g. end of iteration, after a fixed period of time.) is met. Furthermore, before committing, updates of the same parameters in the update buffer are merged in advance to avoid redundant updates.

C. Lightweight Task Scheduling

As shown in Figure 4, there is no task scheduling in the traditional parameter server architecture. Tasks are running with heavyweight system-level thread. However, thread switching needs through system calls, the overhead is so large that limits the number of tasks. In addition, the general PS application is network intensive,which requires frequent network waiting. However, when one task is waiting for network, the worker does not switch to other tasks for computing, but transfers the computing resource to OS and waits to be waked up by OS too. A large number of computing resources are wasted during that time.

Grappa提供了一个简单的轻量级任务调度程序，它可以帮助在参数服务器上轻松实现轻量级任务调度程序。

As shown in Figure 5, in the new parameter server DPS, the lightweight thread scheduling system in Grappa[3] is used. Grappa provides a simple light-weight task scheduler which helps easily implement a lightweight task scheduler on parameter server. When the task is executing a blocking operation such as networking, the user-space task scheduler will suspend the task, then quickly switch to run other tasks to make full use of CPU resources. During task execution, the task scheduler will cause Worker to hand over control of the processor each time a long delay operation is performed, allowing the processor to remain busy while waiting for the operation to complete. In addition, the programmer can explicitly indicate the schedule. In order to minimize the context switch overhead, the task scheduler used in DPS is completely in the user space. The scheduler is to save the state of a task and restore the state of another task. When a task encounters a long delay operation, its worker will be paused and then awakened when the operation is completed.

As shown in Figure 6, in the DPS implementation, there is a scheduler on each node, which maintains two task queues: one is the ready queue, store tasks that are ready to execute; the other one is waiting queue, store tasks which are blocked by IO operations. When the task do IO operations, such as read and write disks, network traffic, etc., the operation is put into the background asynchronously, and the current task is blocked waiting for the IO result to return. The current task will be put into the waiting queue, then the new task will be fetched from the ready queue, and its state will be loaded into the worker. After that, the worker begins to run the new task. In addition, the scheduler generates a periodic task to check whether the task in the waiting queue has ended (usually checking whether

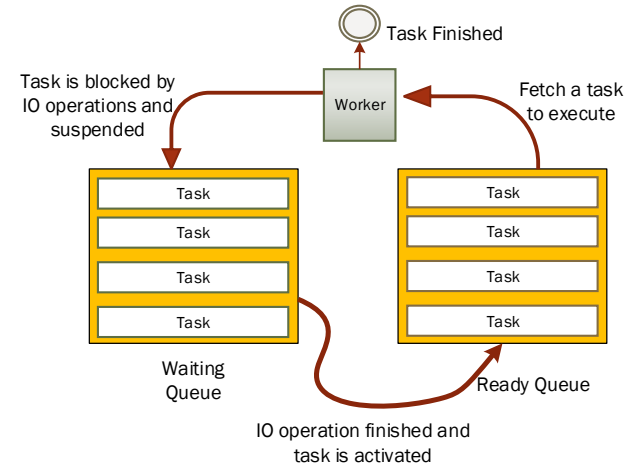
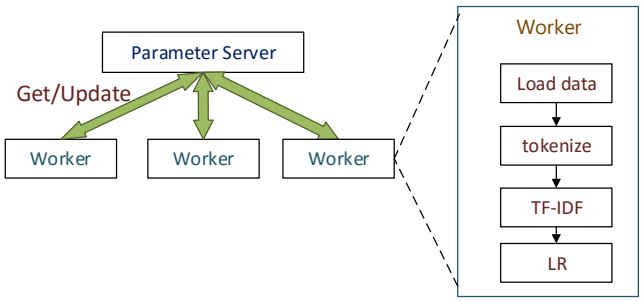


Fig. 6. task scheduling process.



当任务正在执行阻塞操作时，任务调度器将会挂起任务并且用CPU中所有资源来运行其他的任务。

Fig. 7. Workflow of text classification in parameter server.

a condition variable is satisfied), and if it is satisfied, the task will be moved the task from the wait queue to the ready queue.

For a parameter server, the main interface such as parameter get and parameter update are long delay blocking operations. In the DPS system implementation, we wrap these operations into asynchronous tasks, which would be scheduling to execute and suspend by task scheduler, thereby avoiding the waste of computing resources during that waiting time.

D. Data Management and Data Abstraction

As shown in Figure 7, the traditional parameter server workflow is as follows: each worker load and process data independently, and lack of a high-level data abstraction. The intermediate results of data processing cannot be used, meanwhile, each application needs to do data loading and processing independently. Algorithms and data processing are too coupled and hard to be reused. Moreover, it is difficult to complete some complicated data processing like *join* operation.

As shown in Figure 8, in general data flow systems such as Hadoop [9], [10], spark[11], the data is abstracted as a high-level dataset, and each step of data processing is converting from one dataset to another. The intermediate dataset can be reused too. The programming interface provides with *map*,

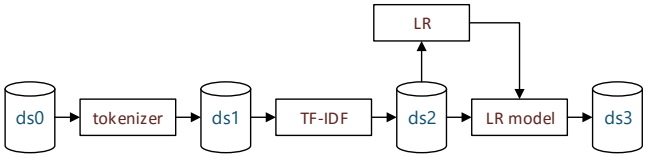


Fig. 8. Work flow of text classification in map/reduce system.

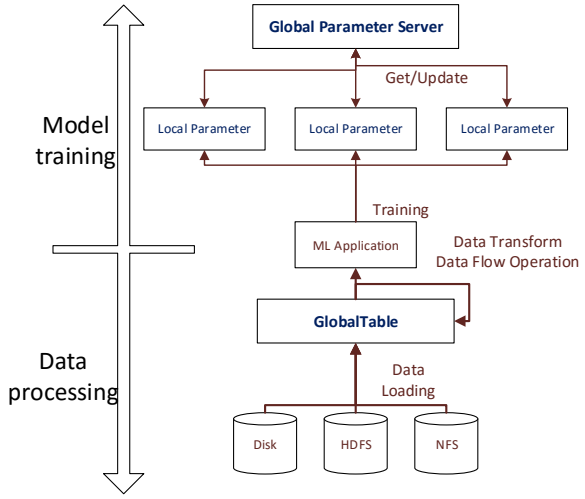


Fig. 9. Using GlobalTable to data processing and Parameter Server to model training.

$reduce$, $groupby$ and other data flow operation interfaces, so it is convenient to manage the data with the data abstraction of the dataset.

We combine the ease of use of data flow system such as Spark to DPS. a data abstraction is designed for parameter server, which called $GlobalTable$. $GlobalTable$ is a distributed dataset, which provides map, $reduce$, $groupby$ and other data flow operation interfaces to support the conversion between different distributed datasets. In DPS, We wrapped $GlobalAddress$ which is provided in Grappa to implement $GlobalTable$. Additionally, $GlobalTable$ can also be used as training input of machine learning applications. The system that combines the advantages of data processing systems and parameter servers is shown as Figure 9.

E. Programming Interface and Example Code

The system mainly provides three kinds of programming interface. 1) Data APIs, such as $loadData$, map, $reduce$, $groupBy$, et.al. 2) PS APIs, $createTable$, get, $update$. 3) Tasking APIs, run, $complete$, $wait$. All of these APIs are designed very briefly. With these APIs, users can quickly implement various machine learning applications on the system.

Figure 10 shows a good programming interface and code style for the system through an example code. The code shows the process of loading the dataset from HDFS, preprocessing it, and then using the processed data for iterative model training.

IV. Evaluation

We evaluated DPS on several machine learning algorithms: matrix factorization (SGD MF) and logistic regression. We compare the proposed system, DPS, with the popular parameter system, Petuum[5].

A. ML applications

1) Multi-class Logistic Regression (MLR): Multi-class logistic regression, or Multinomial logistic regression (MLR) generalizes binary logistic regression to handle settings with multiple classes. It can be applied directly to the multi-class classification problem, or used within other models (e.g. the last layer of a deep neural network). Our MLR app is implemented on the DPS system.

2) Non-negative Matrix Factorization (NMF): Non-negative matrix factorization (NMF or NNMF), also non-negative matrix approximation[12] is a group of algorithms in multivariate analysis and linear algebra where a matrix V is factorized into (usually) two matrices W and H, with the property that all three matrices have no negative elements. This non-negativity makes the resulting matrices easier to inspect. The NMF app uses the projected Stochastic Gradient Descent (SGD) algorithm to learn the two matrices L and R.

B. Cluster Setup

The experiments were conducted on a research cluster consisting of 4 high-end computers running Debian 8.0. Each machine contains 2 $\times$ 8-core Intel(R) Xeon(TM) E5-2650 CPUs (16 physical cores per machine) and 256GB of RAM. The machines are distributed and connected via two networks: 1Gb Ethernet and 40Gb Infiniband. Every machine is used to host PS server and Ps worker.

C. Experiment Results

1) MLR Performance: As shown in Figure 11, the DPS system designed in this paper has achieved a performance improvement of about 30% compared to that of Petuum, whether it is a small dataset or a large data set, it indicates that the performance of DPS has a relatively strong performance stability. The reason is that DPS reduces the network time and improves the computing resource utilization.

2) NMF Performance: As shown in Figure 12, the matrix of different sizes is decomposed in nonnegative matrix decomposition experiments. The experimental results show that DPS achieved performance improvement from 21% to 90% compared to that of Petuum. With the matrix size of 10,000 by 10000, DPS achieved nearly 2x performance. It owed to the reduced networking time and improved computing utilization. Surprisely, it can be observed that the performance improvement increased significantly with the increase of training matrix size.

3) Networking time comparison: Reducing network overhead is the goal of the proposed system design. A network time experiment is conducted to compare the network time reduction of DPS and Petuum by gathering statistics of computing time and networking time while running machine learning

```
//load data from hdfs
GlobalTable<string> rawdata=dataManager.loadTextFile("hdfs://...");
//transform rawdata to Lable and Points
GlobalTable<LabeledPoint> labels=rawdata.map([](string& line){
    return parseSparseData(line);
});
PSTable model=DPS::createSSPTable(2);//create parameter server table
PSTable another_model=DPS::createBSPTable();
int num_epoch=100;
//trainning each partition, this will spawn plenty of tasks
CompletionEvent ce;
labels.forEachPartitions([model,num_epoch](Partition<LabledPoint>& batch){
    clock=DPS::resetClock();
    for(int i=0;i<num_epoch;i++){//iterate to trainning
        model.batchGet(...);//get parameter
        model.batchInc(...);//update parameter
        clock.clock();//set worker clock
    }
}).run(&ce);
//another async job: traning anther model by a different method
labels.forEachPartitions(...).run(&ce);
ce.wait();//wait two job to finish
model.save("hdfs://...");//save model to hdfs
```

Fig. 10. Example code.

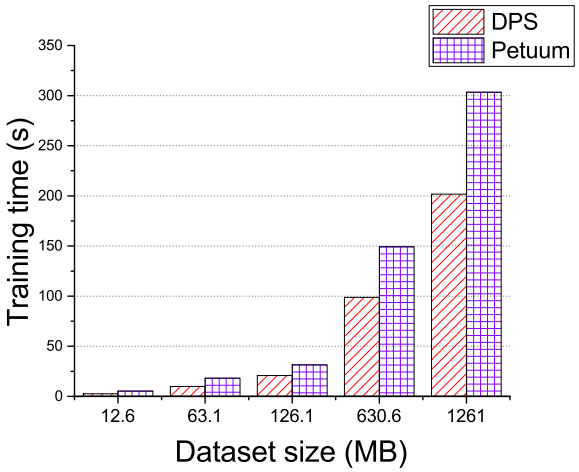


Fig. 11. MLR training time: DPS vs Petuum.

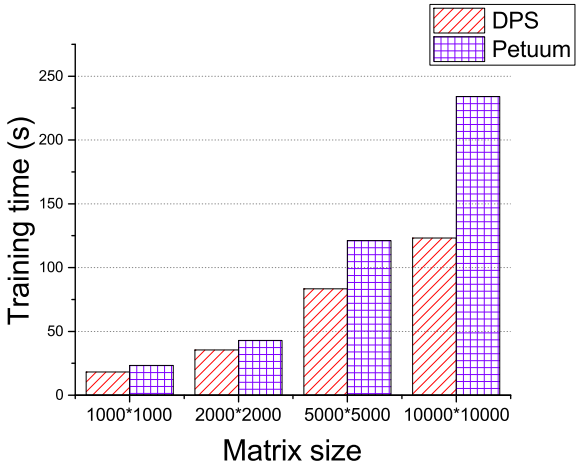


Fig. 12. NMF training time: DPS vs Petuum.

applications. As shown in Figure 13, in the MLR application, Petuum takes a total of 304 seconds, in which the networking time and the computing time are 126 seconds (41%) and 178 seconds (59%) respectively. In the DPS system, the total time is 201 seconds, in which the networking time and computing time are 72.6 seconds (36%) and 129.2 (64%) seconds. It can be observed that DPS reduced about 50% networking time and about 29% computing time compared to Petuum. The reason is that lightweight task scheduling mechanism is undertaken by DPS to reduce network scheduling time and make full use of the computing resources of the cluster.

4) performance with different staleness: An experiment is also conducted to evaluate system performance with different staleness of SSP protocol. As shown in Figure 14, it can be

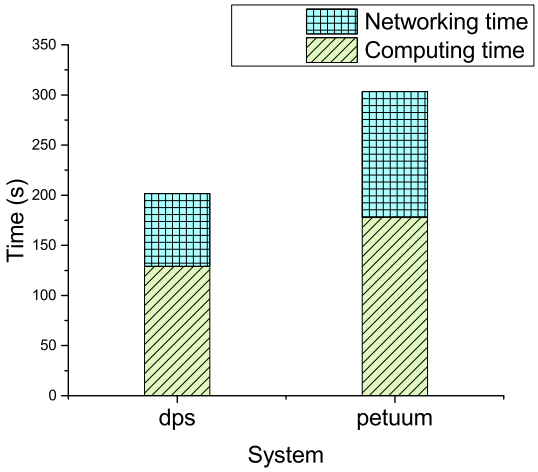


Fig. 13. Computing time VS networking time.

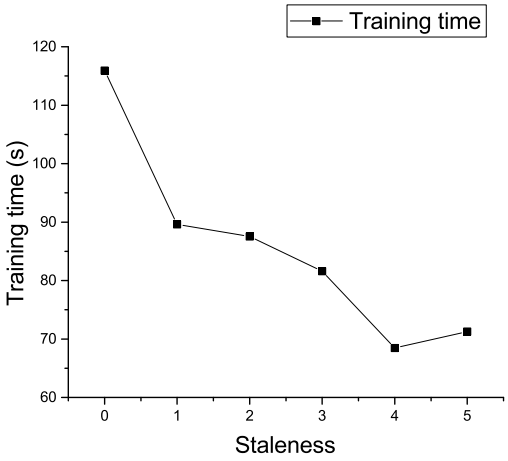


Fig. 14. Performance with different staleness.

seen that it takes the longest training time (115.93 seconds) when the staleness is set to 0, i.e. the consistency model degrades to BSP. When staleness is set to 1 or 2, the performance increase of more than 20%. The experiment results show that the SSP consistency model improves the performance of machine learning by saving network communication. In addition, it can also be noted that increasing staleness could improve system performance. However, when staleness is set to 5, the performance declines slightly. The reason is that the SSP protocol could only save network time but not the computing part compared to BSP. And increasing staleness may increase the inconsistency among nodes which may slow down the training process. It can be concluded that SSP protocol has a good effect between staleness and 2 to 4. And there is no need to take too large staleness.

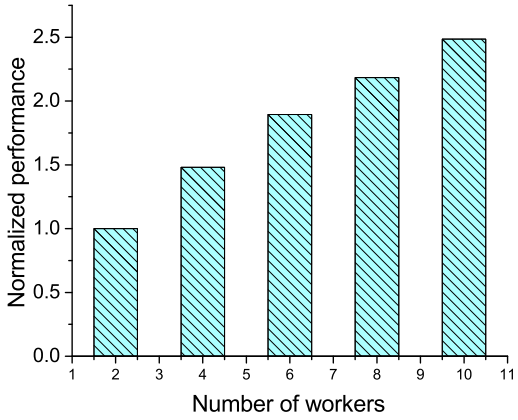


Fig. 15. Performance with different number of workers.

5) System scalability experiment: Another experiment is conducted to evaluate the scalability of the proposed system on the MLR application. The number of workers ranged from 2 to 10 while all other settings stayed the same. The experimental results are shown in Figure 15. It can be observed that the throughput of the system increases with the increase of the number of workers, but there is a certain gap between the performance of DPS and the perfect linear scalability. With workers ranging from 2 to 10, the number of worker increases 4x, and the system throughput only improves 2x. It may be the increased network access to global parameters brought by the increased number of workers that limits the scalability. There should be some potential improvements in the proposed DPS. This could be regarded as the future work.

V. Related Work

A. Big Data Machine Learning Systems

Current big data machine learning system can be classified into three types in the data abstraction and programming model aspect, matrix abstraction, graph abstraction and parameter abstraction.

1) matrix-based systems: Most machine learning algorithms can be expressed as matrix or vector algebra. Many researchers have constructed some large data machine learning systems based on matrix computing model, which allows developers to build data analysis and machine learning algorithms directly based on matrix calculation. Spark Mllib[13] is an attempt of the matrix-based machine learning system that is built on Spark[11], and provides many machine learning algorithms and tools.

2) graph-based systems: Some practical applications of social network analysis usually need to be based on graph-based machine learning systems for efficiency[14]. Many graph-based big data machine learning systems have been proposed to meet the real world needs. Graphlab[15] is a graph system for machine learning that achieves high performance and

provides high-level programming interface like MapReduce. GraphX[16] is a batch processing graph system built on Spark that transforms graph operations into RDD-based operations.

3) parameter-based systems: Matrix and graph abstraction lack flexibility and are hard to tune while parameter tuning is very important for machine learning, especially for deep learning to achieve better accuracy. To cope with these problems, parameter-based abstraction has been proposed in both industry and academia. The parameter-based system can also be called parameter server. Parameter server is originally introduced by [17] that leveraged Memcached as a key-value store but lacked flexibility and performance. YahooLDA [18] used a dedicated server as parameter store with set, get, and $update$ operations to improve the design. [18] and [17] can be classified as the first generation parameter server. The second generation parameter servers are application specific, such as Distbelief[19]. A more general parameter server is Petuum. It improves the design by combining parameter server with the SSP (stale synchronous parallel) consistency model that significantly reduces synchronous overhead and improves system performance compared to previous parameter servers. Parameter Server [4] is called the third generation parameter server for very large scale machine learning. It improves the parameter server design systematically by introducing elastic scalability, replica-based fault tolerance and some level of ease of use. However, it lacks high-level abstraction of data and programming models, making it complicated to implement machine learning algorithms on it.

VI. Conclusion and Future Work

We described a DSM-based parameter server system, DPS, to solve the problem of efficiently storing and updating of model parameters. DPS is easy to use: the high-level programming interface and data abstraction that allows data preprocessing, data transformation and training process to be pipelined in a small number of lines of codes; data management partitioned training data automatically when scaling to a different number of workers. Flexible consistency models are supported to suit different machine learning algorithms. DPS is efficient: a light-wight task scheduling system is adopted to reduce IO network time and take full advantage of computing resources by decomposing training process into small tasks. Experiments have been conducted to demonstrate its performance on MLR and NMF applications, SSP protocol, and scalability. Compared to Petuum, DPS further reduces the network time and achieves up to 1.9x performance. Besides, there exist potential improvements of DPS. In the future, we will implement more machine learning algorithms and improve the system design to further enhance the performance.

Acknowledgment

This work is supported by China 973 Fundamental R&D Program (No.2014CB340300), NSFC program (No.61472022, 61421003), SKLSDE-2016ZX-11, and partly by the Beijing Advanced Innovation Center for Big Data and Brain Computing. We'd also love to extend out gratitude to the anonymous reviewers for their valuable comments and suggestions that help improve the quality of this manuscript.

References

[1] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@ home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[2] Y. Wang, X. Zhao, Z. Sun, H. Yan, L. Wang, Z. Jin, L. Wang, Y. Gao, C. Law, and J. Zeng, "Peacock:learning long-tail topic features for industrial applications," *Acm Transactions on Intelligent Systems and Technology*, vol. 6, no. 4, pp. 1–23, 2014.

[3] J. Nelson, B. Holt, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin, "Latency-tolerant software distributed shared memory," in *Usenix Conference on Usenix Technical Conference*, 2015, pp. 291–305.

[4] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 583–598.

[5] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: A new platform for distributed machine learning on big data," *IEEE Transactions on Big Data*, vol. 1, no. 2, pp. 49–67, 2015.

[6] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 571–582.

[7] G. Alverson, R. Alverson, D. Callahan, B. Koblenz, A. Porterfield, and B. Smith, "Exploiting heterogeneous parallelism on a multithreaded multiprocessor," in *International Conference on Supercomputing*, 1992, pp. 188–197.

[8] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith, "The tera computer system," 1990, pp. 122–127.

[9] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters." in *Conference on Symposium on Opearting Systems Design and Implementation*, 2004, pp. 137–150.

[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *IEEE Symposium on Mass Storage Systems and Technologies*, 2010, pp. 1–10.

[11] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Usenix Conference on Hot Topics in Cloud Computing*, 2010, pp. 10–10.

[12] I. S. Dhillon and S. Sra, "Generalized nonnegative matrix approximations with bregman divergences," *Neural Information Proc Systems*, pp. 283–290, 2005.

[13] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, and S. Owen, "Mllib: machine learning in apache spark," *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2015.

[14] W. Yu, J. Li, M. Z. A. Bhuiyan, R. Zhang, and J. Huai, "Ring: Real-time emerging anomaly monitoring system over text streams," *IEEE Transactions on Big Data*, 2017.

[15] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[16] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework." in *OSDI*, vol. 14, 2014, pp. 599–613.

[17] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," *Proceedings of the Vldb Endowment*, vol. 3, no. 1, pp. 703–710, 2010.

[18] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola, "Scalable inference in latent variable models," in *International Conference on Web Search and Web Data Mining, WSDM 2012, Seattle, Wa, Usa, February*, 2012, pp. 123–132.

[19] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, and P. Tucker, "Large scale distributed deep networks," in *International Conference on Neural Information Processing Systems*, 2012, pp. 1223–1231.