PD^2F: Running a Parameter Server within a Distributed Dataflow Framework

Nan Zhu
School of Computer Science
McGill University
Montreal, QC, Canada
nan.zhu@mail.mcgill.ca

Lei Rao
Huawei Technologies Co. Ltd.
Santa Clara, CA, US
ruby.rao@huawei.com

Xue Liu
School of Computer Science
McGill University
Montreal, QC, Canada
xueliu@cs.mcgill.ca

Abstract

A Parameter Server makes it easy to adapt machine learning algorithms to large-scale applications. It schedules the workload of storing and updating large volume of parameters to distributed servers. It also accommodates flexible consistency requirements to reduce the communication overhead. However, due to the design and deployment of parameter server, important issues arise. It brings additional operational cost, imposes unnecessary overheads to developers and has been inconvenient to the user when working with other data processing systems. To solve these problems, we propose PD^2F, a non-intrusive approach to run **P**arameter server as an application of the **D**istributed **D**ataflow **F**ramework. Preliminary experimental results demonstrate the effectiveness of our proposed solution.

1 Introduction

The growing scale and importance of big data have driven the development and deployment of distributed dataflow frameworks. General-purpose distributed dataflow frameworks, like Hadoop [3], Spark [14], have gained momentum by supporting data-intensive applications. In recent years, with the emerging deep learning technologies, machine computing has even surpassed human experts in certain applications [13]. However, in many scenarios, we do not only need to handle a significant amount of input data but also need to deal with the massive model size containing millions to billions of parameters. To handle these complexities in both the data dimension and the model dimension, the parameter server (PS) framework [6, 8, 2, 12, 7] has been considered as an efficient solution to scale machine learning algorithms. PS distributes the workload of storing and updating the model parameters across multiple servers so that it mitigates the overhead on a single node. Some PSs apply loose synchronization constraint on the parameter value to further reduce the communication overhead [6, 7].

While PS and dataflow frameworks enhance our ability to handle a variety of large-scale data processing problems, there are several critical issues as they move forward as two separate systems. (1) **Operational Cost**: the maintenance and operation of two separate systems evidently take more labor and hardware resources than operating a single system and hence

these lead to a higher operational cost; (2) **Unnecessary Overheads**: Existing dataflow frameworks have already developed mature solutions for multi-tenant management, authentication, etc., however having two separate development tracks still requires the developers of PSs to repeat the work done in the community and hence it results in unnecessary overheads; (3) **Inconvenient to use**: PS has been developed as a specific system for machine learning problems. Thus, it relies on dataflow framework for data preprocessing. When researchers and engineers are actively trying different combinations of features, they have to refer to two separate frameworks each time they need to change the input data format. This may lead to more operation time and errors.

To address the above issues, we design a system, PD^2F, running **P**S as an application within the **D**istributed **D**ataflow **F**ramework. We implement it as a third-party library of a widely used dataflow framework, Spark [14]. We expose the APIs that are identical to the original Spark machine learning library. The main contribution of this paper is threefold:

- **We separate the task scheduling logic like Stale Synchronous Parallel (SSP) [6] or Dependent Task [7] from the dataflow framework scheduler.** The scheduler of dataflow frameworks handles resource allocation and task scheduling as a whole. It assigns tasks wrapping user-defined logic to distributed servers when there are available computing resources. Upon the completion of logic, it reclaims the resources. The "Resource Manager and Scheduler as a whole" design prevents us from controlling the synchronization of parameters in a more flexible way. To solve this problem, we take the original framework scheduler as a resource manager and have a "Coordinator" thread running aside to it to decide which tasks shall run or stop.
- **We introduce the "Stateful Task" to implement PS node and local parameter cache.** Dataflow frameworks provide the high-level abstraction for data processing and do not expose any internal state of the task to the user. The "stateless" view of the task to the user makes it difficult to implement the PS instance that maintains the parameters as the internal state. We use the ordinary task in dataflow frameworks as the wrapper of PS instance, and expose the internal state, i.e. parameters, via a message-based channel.
- **We decouple the lifecycle of computing tasks in dataflow frameworks with the input data.** Considering about the lifecycle, a task in PS is with a more coarse granularity. The task in PS takes a relatively larger chunk of data as input and updates the parameters for several iterations before it stops. In contrast, dataflow framework tasks are trending to be with the finer and finer granularity [11]. They take much smaller chunk of data and stop after they apply the user-defined function to every element in the chunk for a single round. The binding between tasks' lifecycle and input data prevents us from implementing a more flexible parameter synchronization mechanism in PS, such as SSP. Our proposed solution addresses this issue by allowing the tasks in dataflow frameworks to delegate the scheduling of its lifecycle to the Coordinator, regardless of the size and the access pattern of its input data.

2 Background

The mini-batch Stochastic Gradient Descent (SGD), especially its parallel version, has been a popular gradient descent optimization method used in the large-scale machine learning model training. In this section, we give an overview of how the parallel mini-batch SGD is implemented in Spark. It serves as the background knowledge on why PS is an efficient solution to train massive machine learning models.

Figure 1 shows the workflow of the mini-batch SGD in Spark [5]. The application in Spark consists of two types of processes, a singleton *Driver*, which schedules tasks, and one or more *Executors* to execute the tasks. For each iteration, Spark samples the training dataset as a mini-batch, which is partitioned to multiple servers. A computing task is started for each partition to updates the gradient in parallel. The update of the gradients are aggregated following a tree structure consisting of Executors and finally flow back to the

Driver. The iteration ends after the Driver has aggregated all updates. At the beginning of each iteration, the driver broadcasts the updated gradients to all executors as the input for the coming iteration.

The major problems in this implementation are: 1) all parameters have to be collected to a single node (Driver) and broadcasted again, this brings large single-node memory pressure and communication overhead when the model size is very large; 2) the strict constraint on the iteration progress makes the application vulnerable to the straggler problem [4].

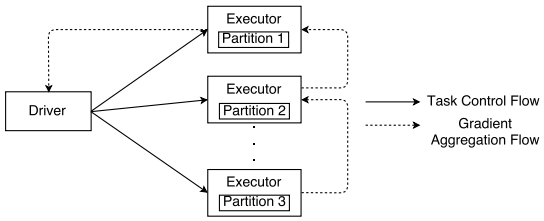

Figure 1: Workflow of SGD in Spark

3 Design of PD^2F

Figure 2 shows the components of PD^2F, a PS application within Spark framework[1]. PD^2F consists of a scheduling module, Coordinator, and two Spark jobs including Parameter Server job and Learning job.

3.1 Coordinator

As shown in Figure 2, the scheduling module in our system, "Coordinator", runs in parallel with the Driver (i.e. Spark's application task scheduler). The Coordinator is the key to implement flexible parameter synchronization mechanism in PS. We partition the training data in distributed servers and launch a task for each partition to iteratively update the model parameters. The Coordinator tracks the iteration number that the tasks work on for each partition. When the task finishes an iteration, it inquires the Coordinator with its just-finished iteration number (i.e. x). If $x - s \leq min(i)$, the Coordinator indicates the task to continue the processing of the next iteration, where s is the staleness bound and $min(i)$ is the least iteration number over all partitions. If $x - s > min(i)$, the Coordinator schedules the task to release the resources and stop. The Coordinator starts new Learning Jobs (Section 3.3) after $min(i)$ catches up, ensuring that the stopped processing of the partition resume. **With the design of Coordinator, we provide a more flexible parameter synchronization mechanism**.

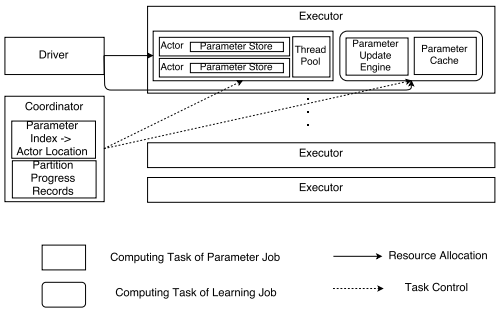

Figure 2: PD^2F Architecture

3.2 Parameter Server Job

Parameter Server job starts **"Stateful Task"** maintaining the parameters in a distributed fashion. Each "Stateful Task" acquires the resources (CPU and Memory) from Spark cluster and keeps running until the end of PS application. "Stateful Task" delegates the state maintenance to a memory-efficient entity, Actor [1] (as shown in Figure 2). When a stateful task starts, it associates several actors with the CPU resources it acquires. Each actor maintains a set of parameters and the only way to read/update parameters is to send messages to the actor. When there is a message in the queue, the actor is attached to CPU resources acquired by the task to execute the message processing logic. To ensure that the parameter server actors are always available for parameter synchronization, we

[1]There are proposals in the community about PS with Spark [9], however, it requires to modify the Spark's core implementation which is also the reason for its being rejected by the community.

guarantee that all tasks in Parameter Server Job must be running before the Coordinator starts Learning job to train the model.

The mapping relationship from the parameters to the actor location is tracked by the Coordinator and broadcasted to all servers so that the learning task can get the actor location for parameter synchronization. We assign a unique ID to each actor and persist its state (parameters) to the persistent storage system (e.g. HBase) for the fault-tolerance purpose.

3.3 Learning Job

The computing tasks in Learning Job update the parameters and synchronize with the actors in the Parameter Server job. To implement the flexible consistency control model, we **decouple the task lifecycle and input data**. Every task in Learning Job maintains the iteration number it works on and the local cache for the parameter. With the iteration number and the local parameter cache, we implement the flexible consistency model, Stale Synchronous Parallel. At the end of each iteration, the task inquires the Coordinator to decide whether it continues, i.e. whether its progress is faster than the slowest task for more than s iterations. We support the following operations in the computing tasks of Learning Job: 1) *read_para(parameter_ids)*: retrieve the latest value of the parameters identified by *parameter_ids*; 2) *inc(parameter_ids, val)*: increase the value of the parameters identified by *parameter_ids* with the value *val*, which can be negative; 3) *sync()*: synchronize the local updates of parameters with all actors in Parameter Job involved by the outstanding *inc()*.

When a task working on the iteration i requests a parameter p by calling read_para(), it first checks its local cache and compares the version of the parameter v with i. If $i - v \leq s$, it takes the cached value of the parameter, updates it and commits the updated parameter by calling sync() after it finishes the current iteration; otherwise, it reads parameters of interests from the remote PS actors.

4 Preliminary Evaluation

We implemented PD^2F and evaluated it using the URL reputation dataset [10]. We used logistic regression with SGD to classify 2,396,130 URLs into malicious and benign categories according to 3,231,961 features given in the dataset. We compare PS with SSP synchronization mechanism in PD^2F (with staleness boundary as 5 and 25) and Spark 1.4.1 in terms of the rate of convergence and training time. We ran the experiments in a 4-servers cluster. Three of them have Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz (4 physical cores, 8 logical cores with Hyper-Threading) and 16 GB RAM. The other one has Intel(R) Xeon(R) CPU E5-2687W v3 @ 3.10GHz (10 physical cores, 20 logical cores with Hyper-Threading) and 32GB RAM. Each Spark Executor process is with 8G memory. We started 100 PS actors, 10 per task for Parameter Server job and the Learning job parallelism is 10. In total, we run 50 iterations with the step size as 0.001.

Figure 3 shows the experimental results on the rate of convergence. The figure demonstrates the average loss value in the last ten iterations. We observe that with the same number of iterations, PD^2F converged to less loss value than Spark. In Table 1, we show two metrics describing the classification accuracy of three execution models. The dataset we used for evaluation contains 33% positive instances, so that we show the common evaluation metric, area under ROC, as well as the area under PR which has the better capability to capture the skewness in the dataset. From the table, we observe that PD^2F performs better

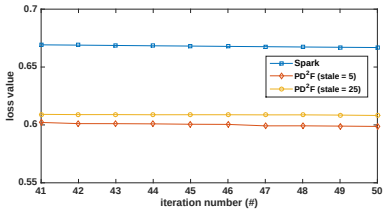

Figure 3: Evaluation Results: we compare the PS with SSP model implemented in PD^2F with LR algorithm in Spark in terms of the rate of convergence.

	Area under ROC	Area under PR
PD^2F (stale = 5)	0.9	0.81
PD^2F (stale = 25)	0.88	0.79
Spark	0.85	0.74

Table 1: Comparison of Classification Accuracy

than Spark in correctly classify an URL under both of these two metrics. We attribute the improvement of the accuracy to the feature of SSP. A computing task in SSP execution model may lose updates made by the other tasks which are slower than it for less than s iterations. Simultaneously, a computing task in SSP also gains the updates from the tasks which are faster than it for less than s iterations. With the dataset used in our evaluation, we obviously got more gains than loss.

To finish 50 iterations, these three systems spent nearly an identical amount of time, around 210s. To avoid the modification of the internal of Spark, we sacrificed several chances to further optimize the system performance, e.g. the tasks started in the same executor cannot have a shared and mutable memory space so that they cannot share the latest value of the parameters with "process cache" [6].

5 Summary

We implement PD^2F, a Parameter Server system within Distributed Dataflow Framework. The proposed work incorporates the benefits brought by both the PS and Dataflow framework computing paradigm: 1) By training a machine model with the PS paradigm, we eliminate the single point performance bottleneck when synchronizing parameters in the conventional dataflow framework; 2) With the support of flexible parameter synchronization mechanism, i.e. SSP, we achieve the faster rate of convergence than the dataflow-based machine learning system; 3) PD^2F is designed in a non-intrusive fashion so that we can easily build a complete data processing pipeline covering dataflow-based Extract-Transformation-Load and PS-based machine learning model training. In future, we will extend the work to support more optimization algorithms in PD^2F. At the same time, we will explore approaches to integrate existing PS libraries with the dataflow framework, using the design philosophy we exhibited in this paper, to further improve the deployment efficiency and the user experience of machine learning systems.

6 Acknowledgement

We are indebted to Tianqi Chen for insightful comments on several discussions about this work.

References

[1] Gul A. Agha, Ian A. Mason, Scott F. Smith, and Carolyn L. Talcott. A foundation for actor computation. *J. Funct. Program.*, 7(1):1–72, January 1997.

[2] Amr Ahmed, Moahmed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J. Smola. Scalable inference in latent variable models. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 123–132, New York, NY, USA, 2012. ACM.

[3] Apache. Hadoop. `http://hadoop.apache.org/`.

[4] James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Gregory R. Ganger, Garth Gibson, Kimberly Keeton, and Eric Xing. Solving the straggler problem with bounded

staleness. In *Presented as part of the 14th Workshop on Hot Topics in Operating Systems*, Berkeley, CA, 2013. USENIX.

[5] Spark Community. Spark stochastic gradient descent. `http://spark.apache.org/docs/latest/mllib-optimization.html#stochastic-gradient-descent-sgd`.

[6] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1223–1231. Curran Associates, Inc., 2013.

[7] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, Broomfield, CO, October 2014. USENIX Association.

[8] Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G Andersen, and Alexander Smola. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, 2013.

[9] Qiping Li. A prototype of parameter server. `https://issues.apache.org/jira/browse/SPARK-6932`.

[10] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 681–688, New York, NY, USA, 2009. ACM.

[11] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: Distributed, low latency scheduling. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 69–84, New York, NY, USA, 2013. ACM.

[12] Alexander Smola and Shravan Narayanamurthy. An architecture for parallel topic models. *Proc. VLDB Endow.*, 3(1-2):703–710, September 2010.

[13] Michael Thomsen. Microsoft's deep learning project outperforms humans in image recognition. `http://www.forbes.com/sites/michaelthomsen/2015/02/19/microsofts-deep-learning-project-outperforms-humans-in-image-recognition/`.

[14] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.