Summary of Scaling Distributed Machine Learning with the Parameter Server

Jinyang Li (jinyli), Yin Lin (irenelin), Jie Liu (jiezzliu)

Problem and Motivation

Solving large scale machine learning problems requires distributed training and inference. Due to the growth of data and the resulting model complexity, an increasing number of parameters are involved in the training process, which are maintained by server nodes and globally shared by all worker nodes.

There are three main challenges of implementing the sharing:

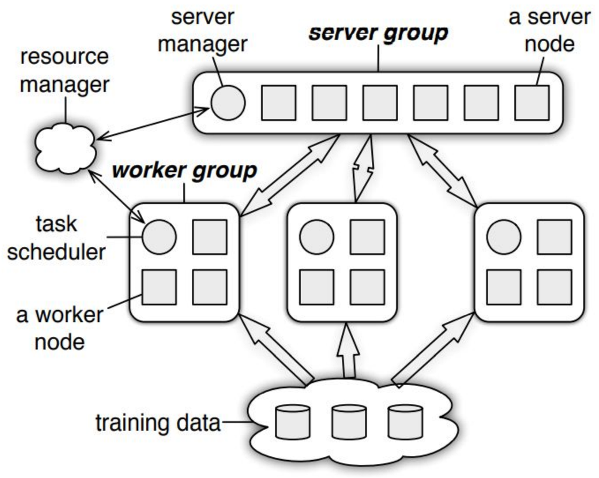
- Accessing parameters requires lots of network bandwidth
- Training is sequential and synchronization is hard to scale
- Fault tolerance at scale

Machine learning systems are vital as they are widely used in web search, recommendation systems, document analysis, and many other applications that are closely related to our daily lives. And many machine learning problems rely on large amounts of data for training and inference. When solving distributed machine learning problems, the issue of reading and updating parameters shared between different worker nodes is ubiquitous. Therefore, it is very important to design a parameter server framework to handle this issue. This is the first general purpose ML system capable of scaling to industrial scale sizes.

Hypothesis

At large scale, no single machine can solve machine learning problems sufficiently rapidly, due to the growth of data and the resulting model complexity. Both data and workloads are distributed over worker nodes, while the server nodes maintain globally shared parameters. Realistic quantities of training data can range between 1TB and 1PB, and quantities of model parameters ranges between 10^9 and 10^{12}.

Another hypothesis is that fault tolerance is critical at scale since machines can be unreliable and jobs can be preempted. Unlike in many research settings where jobs run exclusively on a cluster without contention, fault tolerance is a necessity in real world deployments.



Solution Overview

They propose a parameter server framework for distributed machine learning problems. Parameter server nodes are grouped into a server group and several worker groups as shown above. A server node in the server group maintains a partition of the globally shared parameters. Server nodes communicate with each other to replicate and/or to migrate parameters for reliability and scaling. It provides five key features:

1. **Efficient communication**
   Data is sent between nodes using push and pull operations. The parameter server optimizes these updates for programmer convenience as well as computational and network bandwidth efficiency by supporting range-based push and pull. The asynchronous communication model does not block computation (unless requested). Tasks are generally asynchronous in nature and programs/applications can continue executing after issuing the task. A task can be marked as completed only when all subtasks that the given task depends on returns. This communication model is optimized for machine learning tasks to reduce network traffic and overhead.
2. **Flexible consistency models**
   Relaxed consistency further hides synchronization cost and latency. By default tasks run in parallel. Dependencies can be placed between tasks to serialize task execution. Three different models (**Sequential, Eventual, Bounded Delay**) can be implemented by task dependency. They allow the algorithm designer to balance algorithmic convergence rate and system efficiency. Especially the maximal delay time τ allows more flexible control. The best trade-off depends on data, algorithm, and hardware. Moreover, user-defined filters selectively communicate (key, value) pairs, which allows for fine-grained consistency control within a task.
3. **Elastic scalability**
   New nodes can be added without restarting the running framework. Consistent Hashing allows for easy addition and removal of new nodes.
4. **Continuous fault tolerance**
   Recovery from and repair of non-catastrophic machine failures within 1s, without interrupting computation. Entries are replicated using chain replication. Vector clocks ensure well-defined behavior after network partition and failure.
5. **Ease of Use**
   The globally shared parameters are represented as (potentially sparse) vectors and matrices to facilitate development of machine learning applications. The linear algebra data types come with high-performance multi-threaded libraries.

Limitations and possible improvements

1. The training setup requires users to be able to write distributed algorithms for each machine learning strategy; there's no clear indication that there are some premade "building blocks" users can build upon. It's not very clear how generalizable this setup is for other algorithms and use cases. In other words, it does not provide a flexible enough programming model for building different machine learning applications.
2. Though users are given the flexibility to design synchronization policies, it is hard to find the optimal policies merely based experience. Later, there are some works applying a

reinforcement learning (RL) based approach to learning optimal synchronization policies in this setting, which enables users to learn synchronization policies that generalize to different cluster environments.
3. The evaluation part only focuses on different use cases but is not compared with other frameworks.
4. The paper mentions two types of node failures – server node and worker node. Yet, it does not mention the fault tolerance of the scheduler node.
5. Sometimes the parameter server as subsystems might not be needed. An alternative solution in the Tensorflow system is allowing the expression of stateful parameter nodes as variables, and use additional nodes in the graph to represent variable update operations.

Summary of Class Discussion

Q: Would you say a parameter server is like a database or is it doing some computations also?
A: In some way, it is a database. The highest-level point is to provide a point where all these different workers are working on those parameters. They have to be somewhere, they have to come together to be aggregated.

Q: Is it true that for any model that I design, I have to do this experiment to figure out how much bounded delay I want?
A: From the paper I didn't necessarily see them give like a rule of thumb. In the experiments they vary the parameters. I guess in the real execution, it requires user defined parameters.

Q: Could workers also filter what they send to the parameter server? Or wait until they get a big update?
A: Yes, the users can define what they want the workers to filter. In practice, system designs usually expose all of parameter configurations to the users and provide some default values.
It depends on the consistency model. It can be turned on/off.

A: Model parallelism vs Data parallelism?
Q: Data parallelism: from the workers perspective, every worker has the entire mode when it is running. Entire model is being trained on the local data that you have. To aggregate those things you are relying on parameter servers, each of which are holding some parameters of the entire model.
Model parallelism: you can think of that there is no parameter server per se, the entire model is now so large that it doesn't fit. You split it into many pieces and divide the model across all of these workers.

Summary of Project Adam: Building an Efficient and Scalable Deep Learning Training System

Jinyang Li (jinyli), Yin Lin (irenelin), Jie Liu (jiezzliu)

Problem and motivation

This paper presents the design and implementation of a distributed ML system named Adam for scalable and efficient Deep Learning training. With the success of complex tasks such as the visual object recognition and the speech recognition, deep learning models usually require a significant amount of data as well as a huge model to be trained. The existing work of large-scale distributed systems still faces some problems in terms of scalability and efficiency. Therefore, Adam has the following merits:

1. Optimizing and balancing on the computation and communication.
2. High performance and scalability based on the ability of ML training models. The authors make use of techniques such as multi-thread model parameter updates, asynchronous batched parameter updates and so on, which not only improves the efficiency but also the training accuracy.
3. Demonstration of the system efficiency. The authors show that Adam improves the computation speed as well as the prediction accuracy on two well-known benchmarks.

Hypothesis

Project Adam is designed for highly controlled environments (e.g. datacenters) where data is distributed among the machines and high-throughput networks are available. In this paper, the authors focus on the vision tasks. The Adam system is a general-purpose training algorithm that can train any DNN via back-propagation. It supports training any combination of stacked convolutional and fully-connected network layers and can be used for tasks such as speech recognition and text processing. In the experiments, the authors evaluated Adam using a 120 machine cluster, the scaling results are inducted to be scalable given the scaling result of the data.

Solution Overview

Adam architecture is based on Multi-Spert. And all of the machines are divided into three sets: Data serving machines, model training machines and global parameter servers.

Data serving machines: do data transformations. Image transformations are done in advance and cached in data servers.

Model training machines:

- Training is multi-threaded with different images assigned to threads.
- To accelerate training, shared model weights are updated locally without locks.
- Since the model is partitioned across machines, data values need to be communicated. For local communication, we pass a pointer rather than copy data. For non-local, we build a network library which is compatible with pointer communication mechanisms.
- Memory system optimizations: Models are partitioned across machines so that the working sets for model layers fit in the L3 cache which has a higher bandwidth. Create

two custom hand-tuned assembly kernels to address competing cache locality requirements (row/column).

- Mitigating the impact of slow machines: First, allow threads to process multiple images in parallel to avoid stalling threads waiting for slow machines. Second, end an epoch wherever 75% of images are completely processed, rather than all images.
- Parameter server communication: For convolutional layers, servers locally compute weight updates in a buffer and periodically send them to parameter server machines. For fully connected layers, they send activation and error gradient vectors to parameter server machines to do weight updates.

Global parameter server: communicate with model training machines to update weights. But the update is too high to use a key value store. It has a special architecture.

- Throughput optimizations: model parameters are divided into 1M shards and hashed to storage buckets equally distributed among parameter machines. Updates are processed in batches. Data structures and memory allocations are lock free.
- Delayed persistence: Persistence is decoupled from the update process to allow high throughput. Parameter storage is a write back cache, with dirty chunks flushed asynchronously.
- Fault tolerant operation: There are 3 copies of each parameter shard and they are stored on different servers. A set of parameter server (PS) controller machines are responsible to manage parameter machines, like storing the mapping, receiving heart beats and selecting a new primary server when there is a failure.
- Communication isolation: Parameter update processing is isolated from persistence by using two different NICs.

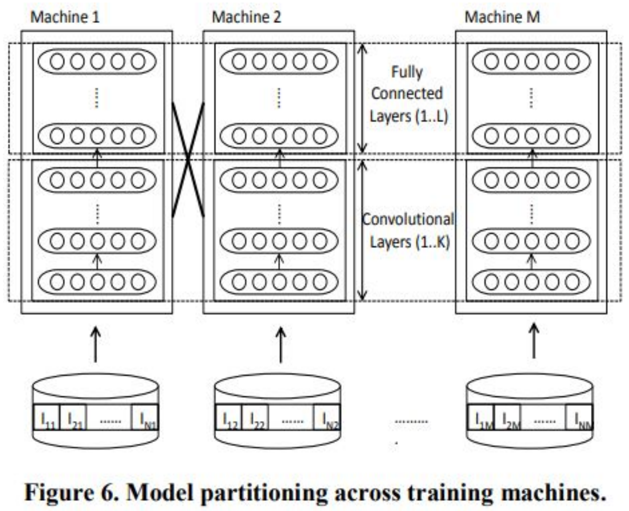


Figure 6. Model partitioning across training machines.

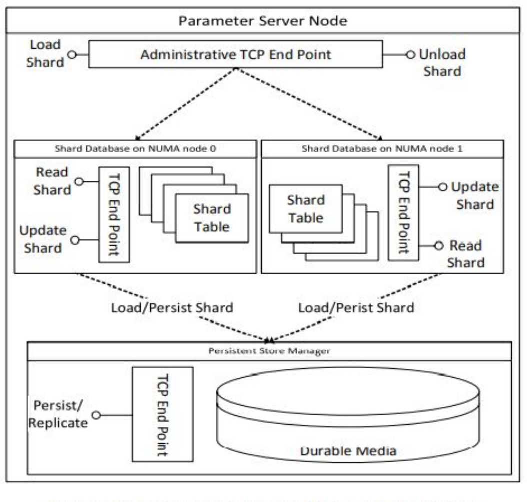


Figure 7. Parameter Server Node Architecture.

Limitations and possible improvements

1. The project Adam only focuses on the training of DNN, no evaluation for other tasks. Although DNN has a wide range of applications, it may need more flexible system design to handle other machine learning models and optimization algorithms.
2. This paper had the assumption that the model is running in the centralized system where data are distributed in a balanced manner across the machines. In the face of skewed

data distribution or relatively poor network connection, new systems or algorithms might also be needed.

3. This paper only discusses the time and accuracy performance of the Adam system. However, the authors do not provide discussions about the other aspects, for example, the availability of the system and whether the model has a convenient interface to debug the models.

Summary of Class Discussion

Q: I think it's cool that they found some heuristic which improves performance while not sacrificing accuracy but there is no indication as to WHY this happens! How can we write future papers trying to build off this concept if we can't give an explanation as to why this occurs? (For Section 3.2.5, Mitigating the Impact of Slow Machines)
A: It seems like the only thing we can do in the future is just randomly "guess" at other optimizations with no real intuition behind our decisions and hope they don't affect accuracy.

Q: By model partitioning, data isn't being stored in the disk at all? Or just most of it is in the L3 cache?
A: It is stored in L3 cache. The model is much larger. If data is stored in disk, you need to wait a long time for data, longer than training time.

Q: Can you really decouple the application from the system completely? Part of designing the system efficiently must include an intuition for what the general application being used on it might do or how it ought to perform on average. Would you agree?
A: I'd agree. You can't decouple if you want to optimize. But I want to make it easier for users to do what they want to do. I don't care about whatever they are trying to do. Sort of like, there are tons of games users play and waste time on. Do we care if that's going to destroy their time and study schedule when building them? We build what they want. It's their problem to manage their time. Having said that, there are calls for designing more ethical AI and ethical systems etc. So we should probably care a bit more than we do. But systems designers are a bit farther from the action. It's a difficult problem but many are starting to work on it

Q: I have seen some papers about ethical AI/ML algorithms or models. I am curious about how system design can handle ethical problems?
A: It's a difficult problem but probably should start from the application level. So AI/ML researchers have to say what they want from the lower layers to be able to do what they want to do.

Q: Would it be easy/feasible for an outside source to test the results from the Adam paper with different datasets?
A: There is a trend push for open source so that people can reproduce. But it's optional some people don't open source or only open source system/data. It is often doable to read paper, implement the missing part and compare.