Summary of "Scaling Distributed Machine Learning with the Parameter Server"

Shaowen Wei(shawnwei), Chi Zhang(clzhang), Die Hu(hudie)

Problem and Motivation

Large scale machine learning systems require careful system design and distributed implementation due to the huge volume of data and model complexity. Sharing of parameters between worker nodes is necessary to deal with the large volume of training data. But the implementation of sharing parameters imposes challenges considering (1)communication problems including network bandwidth for accessing parameters, latency effects due to sequential learning tasks and (2)fault tolerance difficulty, which has strong effects on performance of large scale systems. However, existing parameter servers such as MPI, GraphLab and YahooLDA don't hold the property of continuous fault tolerance and scalability with too large scale. Most of the existing systems perform in a synchronous way. The GraphLab achieves asynchrony at the cost of structural limitation. For this paper, their work focus on providing a user-friendly parameter server with efficient asynchronous communication, elastic scalability and strong fault tolerance.

Hypothesis

- The goal of machine learning algorithm can be expressed via an "objective" function, and the training data is extremely large.
- Machine learning algorithms typically treated model as a linear algebra object.
- To support vector operation optimizations. Authors assume that the key is in order, which reduces the programming effort to implement optimization algorithm.

Solution Overview

Architecture

Parameter server nodes are grouped into a server group and several worker groups. Each server node in server group maintains a replica of globally shared parameters for reliability and scaling. Each worker group runs an application and has a task scheduler to assign task to each worker and monitor the whole process. Each worker typically stores a portion of training data locally. During the computation, each worker only computes local statistics and communicates with server group to update and retrieve the shared parameters.

- *(Key, Value) Vectors*
  The model shared among nodes can be represented as a set of (key, value). Each entry can be read/write remotely or locally. If the keys doesn't exist, system just replace it with zero. It is very useful in linear computation in machine learning. Author also assumes that the keys are ordered to support optimization.
- *Range Push and Pull*
  A technique called range push and pull is used by parameter server to optimize for computational and bandwidth efficiency. For the range push operation, it will send all

existing entries with key in a certain range to the destination, and for range pull operations, it will read all entries with key in a certain range from destination. This interface can also be extended to communicate any local data.

- *Asynchronous Tasks and Dependency*
  Tasks are executed asynchronously, that is the caller can perform further computation immediately after issuing a task, and also mark a task as finished only once it receives the callee's reply. By default, callees executes tasks in parallel for best performance, and caller can place an execute-after-finished dependency between two tasks, which helps implement algorithm logic and supports the flexible consistency models.
- *Flexible Consistency*
  The parameter server gives the algorithm designer flexibility in defining consistency models, which makes it different from other machine learning systems who force users to adopt one particular dependency. Three models can be implemented by task dependency: sequential model, eventual model and bounded delay model.
- *User Defined function and filter*
  Beyond aggregating data from worker, server group can also execute user-defined function which helps to deal with servers' up-to-date shared parameter information. User defined filter is also supported to selectively synchronize individual pairs.

Implementation

Parameters are stored in the format of key-value pairs using consistent hashing for consistency and load balancing. And entries are replicated using chain replication for achieving fault tolerance. The range-based pattern in this parameter server design largely contributes to the implementation for efficiency and fault tolerance.

- *Vector Clock*
  Vector clock is introduced for efficiently tracking aggregation status in level of nodes instead of key-value pairs. The range-based communication pattern helps to easily handle the vector clock implementation by compressing parameters within a range into the same vector clock for a specific node, where the number of ranges is much smaller than that of parameters. Thus the space introduced by vector clock is largely optimized.
- *Messages*
  Messages in the system transmitted between nodes and node groups are structured with a vector clock and a list of key-value pairs. Message compression is also needed for high-bandwidth requirement. Nodes can cache received key lists for reducing the size of messages transmitted in the future. Snappy is used for filtering out zero entries.
- *Replication and Consistency*
  Parameter server partitions keys using consistent hashing. Each server nodes stores a replica and it communicates with the master of a key range using push and pull. Any modification in the master will be copied to the node which holds that replica to maintain consistency.
- *Server Management*
  Addition and removal of nodes are supported to achieve tolerance and dynamic scaling. When a server joins, the server manager assigns the new node a key range to serve as

master, leading to the split or removal of another key range from a terminated node. Meanwhile, the range of data is fetched to maintain the node's status, and the server manager will broadcast the change of nodes.

- *Worker Management*
  The addition of a new worker is similar but simpler: a range of data will be assigned to the new worker; the range of training data from file systems or existing works will be loaded; the change will be broadcasted. Since recovering a worker could be expensive and the affection of losing a small amount of training data is small, options are given to the algorithm designers to control the recovery of workers.

Limitations and Possible Improvements

- By using the asynchronous communication in the system, a single update may take more times of iterations to meet the same objective value comparing with a synchronous one. This slows down the process of computation stage.
- Evaluation was done on specially designed Machine Learning algorithm. Maybe they should also evaluate on algorithms such as neural network.
- Replication strategy used in this system is optimized. But it may still bring problems such as increasing the network traffic and slow down the computation process.

Summary of Class Discussion

In class, we mainly discuss about how this parameter server succeeds in achieving asynchronous communication, fault tolerance and scalability dealing with large scale machine learning systems. Communication cost is largely optimized using range-based pattern and user-defined filters. And fault tolerance is relative better than other systems by doing using different replication strategy.

Q. Is bounded delay time a real time?

A. Author actually doesn't mention it in the paper, but according to figure 13, they didn't give unit of the delay, so maybe delay is not a real time.

Q. Who is in charge of distribute data at first place.

A. Task scheduler in worker group.

Q. Why we use consistent hashing

A. Consistent hashing makes adding and removing server better by improve the load balancing and reduce the intensity of data moving. Also it supports replication which is used in recovery process.

Q. Why using virtual nodes in consistent hashing

A. The reason why we use virtual nodes in consistent hashing is because by adding n nodes to the ring, the data will distribute more fairly on each server. That's because we actually divide the whole ring into more pieces. Without using virtual node, there is a large possibility that some part of the ring may have more data and thus one node gonna has more data than the others.