**Review - Scaling Distributed Machine Learning with the Parameter Server**

**Brief summary:** The paper proposes a Parameter Server framework to solve distributed machine learning problems. The framework features asynchronous data communication, flexible consistency models, scalability and continuous fault tolerance. The author verifies the scalability of the proposed approach through experimental results on petabytes of real data and parameters on problems ranging from Sparse Logistic Regression to Latent Dirichlet Allocation and Distributed Sketching.

**Introduction:** In the recent years, machine learning field has grown in scale. Single machine is insufficient to handle today’s machine learning problems due to the growth of data and model complexity. In order to solve such large-scale machine learning problems, distributed storage and computing system is necessary. In particular, the system should satisfy the following properties – Efficient communication, consistency models, scalability, fault tolerance, durability and ease of use.

**Literature review:** In terms of the implementation, there are lots of familiar ideas. For example, consistent hashing is used to partition parameters among severs and vector clocks are used for fast recovery in case of system failure. However, there are some new ideas such as chain replication which is used to tolerate sever node failure.Previous works had used a key-value model as an abstraction, where entities could be read or written using a key. This newer version takes advantage of the fact that machine learning algorithms typically use linear algebra, and treat the parameters as a vector, assuming that keys are ordered. Workers can push data (for instance a local gradient update) to servers, and then request data back using a pull operation. The parameters are partitioned using consistent hashing.

**Methods:** The architecture makes use of the following components: a) *Vector Clocks*: They use a specialized version of vector clocks that reduces space overhead. This is because many parameters share the same timestamps so fewer things need to be stored. b) *Messages*: Messages can be sent between nodes as key-value pairs. Since ML algorithms require high bandwidth, there much be caching and compression. c) *Replication*: Replication only occurs after aggregation to reduce bandwidth. Thus, we have copies of previous states after core steps to roll back to in the case of failure. d) *Server/Worker Management*: Master and worker nodes communicate with each other to distribute workload and identify and handle failures.

**Results:** The system that the paper proposes has these properties: a) *Key-Value Vectors:* Key-value pairs enable for abstract assignment in different workloads. Specifically, these pairs are treated as sparse linear algebra objects and optimize for sophisticated operations. b) *Push/Pulls:* Each worker pushes its entire local gradient into the servers, and then pulls the updated weight back. In the case of a more advanced algorithm, a range of keys is communicated each time instead. c) *Asynchronous Tasks*: Tasks are executed asynchronously. This means that the caller can perform other computations immediately after issuing a task. d) *Flexible Consistency*: Three different models can be implemented by task dependency: Sequential consistency, Eventual consistency, and Bounded Delay. These different models cover a variety of use cases and does not force the user to adopt a dependency that might impose some consequences for a particular problem.

**Discussion:** One good point about this paper is that it describes the current machine learning goals and methods that made the rest of the paper easier to understand. However, there are some drawbacks too. First, authors mention this design as a third-generation parameter server but did not do any performance analysis against the previous two generations. Second, the evaluation part only focusses on different use cases and is not comparing with other frameworks.

**Conclusion:** The authors clearly propose a novel parameter server framework to solve issues like very large-scale machine learning workloads with high flexibility, consistency, scalability and failure tolerance. However, on one hand it provides a way to scale machine learning computations, but on the other hand, it doesn't provide a flexible enough programming model for building different machine learning applications.