Scaling Distributed Machine Learning with the Parameter Server

Key Takeaways

- The parameter server introduced here is specialized in machine learning model training.
- The parameter server here reduces the problem of network traffic for sending and receiving parameters between machines by caching and compression. The caching means when a range of keys (an example of the keys are the index of columns or rows) are used, they are often always used together. In other words, we often query key (1,2,4,6,8) together since they are the keys stored on a single worker node. Therefore, we do not need to send the keys explicitly. Instead, we can cache the keys and compute a hash of the range of keys, so each time we only need to send the hash. The compression mostly makes use of the sparcity of the data that are mostly zero, and the timestamp of a range of keys is often the same hence we only need to store a single time stamp for a range of keys.
- The parameter server achieves higher performance by relaxing the consistency requirement. A worker does not need to wait for the most up-to-date parameter for its own computation. Instead, it can use slightly outdated parameter of other machines to compute its own update to the parameter. In that way, the time spent on waiting for the network traffic and other machines is reduced, and we need less time for convergence at the cost of more iterations.
- The parameter server is fault tolerant by allowing the joining and leaving of a server node or a worker node.

Difficulty with Parameter Server

- Accessing the parameters requires an enormous amount of network bandwidth.
- Many machine learning algorithms are sequential.
- Learning tasks are often performed in a cloud environment where machines can be unreliable and jobs can be preempted, hence fault tolerance is critical.

Features of Parameter Server

- The parameter server of this paper has the following features:
- Efficient communication: The asynchronous communication model does not block computation. It is optimized for machine learning tasks to reduce network traffic and overhead.
- Flexible consistency models: Relaxed consistency requirement can reduce synchronization cost and latency at the cost of convergence rate. The designer can choose how to balance the above two requirements.
- Elastic scalability: New nodes can be added without restarting the running framework.
- Fault tolerance and durability: Recovery from and repair of non-catastrophic machine failures within 1s, without interrupting computation.
- Ease of Use.

Architecture

- Server nodes are grouped into server node groups.
- Each server node maintains a portion of globally shared parameters.
- Servers communicate with each other to replicate parameters for reliability.
- Each server group has a server manager. The server manager maintains a consistent view of the metadata of the servers, such as node liveness and the assignment of parameter partitions.
- The workers are also grouped into worker node groups. Each worker group runs an application.
- A worker stores locally a portion of training data.
- Worker only communicates with servers and does not communicate with other workers.

- Within each worker group, there is a scheduler node. The scheduler node assigns tasks to workers and monitors their progress. If workers are added or removed, it reschedules unfinished tasks.
- Different worker groups can also work together like a single worker group by using the same namespace of parameters.
- The model parameter can be represented as (key, value) pairs. For example, the linear regression parameter can be represented as (feature index, feature slope) pairs. In order to facilitate the common linear algebra in machine learning, the model parameters are represented as vector/matrix of (key, value) pairs, so that we can make better use of the linear algebra of these instances.
- The portion of parameters stored on the worker can be pushed to the server, while the parameters stored on the server can be pulled to the workers.
- The pull and push can be done on a subrange of (key, value) pairs, where the ranges are defined by the range of keys.
- A task such as pull or push is issued by RPC call. The RPC call can be asynchroneous for higher efficiency, or has a dependency structure by requiring an execute-after-finish dependency structure between tasks.
- The relaxed consistency means in the example of gradient descent, we can collect the contribution to the gradient from all workers then calculate the full gradient, or we can only collect from a subset of workers and leave some workers' contribution to the next iteration. In this way, the calculation is inconsistent, but the model can still converge with more iterations.
- Different consistency can be applied by requiring a task to wait for another task unless a certain timeout has passed. By changing the timeout, we are changing how strong the consistency requirement is.

Implementation

- A vector clock is used to store the time of each (key, value) pair on each node.
- In order to save memory, we cannot store a timestamp for each (key, value) pair on each machine. However, since each update is likely to applied to a range of (key, value) pairs, the vector clock needs to only store the timestamp of a range.
- When there are k communications, a range can be broken into $O(k)$ ranges with each range has a single timestamp for all pairs within the range. In this way, the memory requirement changes from $O(n)$ to $O(k)$.
- When communicating between nodes, the message contains all (key, value) pair for that range and a single timestamp for that range.
- In order save bandwidth, the message is compressed. In addition, since the same range can be sent multiple times and we actually only need the value not the keys, the range of keys can be hashed and cashed on the receiver, so that the next time only the hash of the range needs to be sent. The values can also be sparse, hence we can only send the non-zero values.
- The keys are partitioned into multiple parameter servers just like distributed hash table: the keys are partitioned by ranges, and the server it is stored is its primary server. Other servers also have replicate of the parameters for reliability, and these other servers are called the slave of the primary server. When data on a primary server is modified, the primary server pushes the change to the requesting worker and also its slave servers. The push is completed only after the copy to the slaves is complete.
- Quite often the update to parameter is equivalent to aggregating some statistics from multiple workers first then updating the data using the aggregation. In that case, the primary server waits for the aggregation to be done then does the update, instead of performing the update and transmitting to its slave each time it receives the update from a single worker.

Node Management

- There is a server management node that manages all the server nodes.
- When a new server joins, the server manager assigns the new node a key range, and the new server node serves as the primary server of this key range. This new server node then fetches this range of data to maintain itself as the primary node of this range and also other ranges that it serves as the

slave nodes. After the data replication is complete, the server manager broadcasts the node changes to all other servers. The other servers may need to shrink their own data based on the key ranges that they were assigned but now assigned to the new server, and they may need to resubmit the unfinished tasks to the new node.

- When copying the data from an original server to new server, the original server first prepares the data to be copied into a data structure containing the range of data to be copied and its vector clock. If the new server fails at this stage, then no changes is made to the original server. Then the original server no longer accepts the message affecting the range of keys to be copied by dropping the corresponding messages, and at the same time sends the prepared range of data to the new server node.
- When a server receives a node change message from the server manager, it checks whether it maintains part of the range. If so, since it no longer maintains this part of range, it simply deletes the coreresponding key/value pairs and the vector clock. Then it checks if any of its unreplied data are relevant to the range of data it no longer maintains. If there is such range, the message will be split so that it only replies the part that it now maintains.
- When a server node leaves, similar things happens. The server manager assigns a node by adding the range the failed node to this node.
- When a new worker node joins, the task scheduler assigns the new worker a range of data. This new worker node loads the range of training data from a file system or other workers, and pull the shared parameters from the servers. Then the task scheduler broadcasts the changes so that other workers can free some range of training data they are no longer responsible for.

Evaluation

- The sparse logistic regression is evaluated using 170 billion training samples, 65 billion features, and the data is 636TB when uncompressed. The training system contains 1000 machines, each with 16 core CPU 192G memory and 10Gb ethernet. The system contains 200 parameter servers and 800 workers.
- The training is performed using gradient descent with relaxed consistency. The data are partitioned by columns instead of by rows, and the range of columns are randomly assigned to workers. For each worker at each iteration, there is a tolerance of inconsistency τ so that each worker needs to wait for the iterations that are τ iterations ago to finish, then compute its gradient on its range of features and push to the parameter servers, then pull the aggregated parameters from the server.
- The system here outperforms existing systems in terms of time for convergence. This is because its advantage of allowing relaxed consistency so that more iterations can be done without waiting for all aggregation of statistics to be ccomplete. This is shown by the reduced percentage of time a worker spends on waiting instead of computing. On the other hand, this system also requires more iteration (although still less time) in convergence due to relaxed consistency requirement.
- The system here also reduces the network traffic. Due to caching of keys the system save 50% traffic. The data compression is also effective because there are large fraction of gradient and parameters to be zero in sparse logistic regression.
- By further relaxing the consistency requirement, the system first reduces the time needs then increases the time for convergence.
- The system is also tested on LDA machine learning for finding topics from articles.
- When increasing the number of machines from 1000 to 6000, the convergence speed is increased by 4X.