Improving the Performance of Low Bandwidth Distributed Data Parallelism

Christopher Rae
email: 110641877@ea.edin.sch.uk

Abstract

There is no doubt that networking is a bottleneck when training neural networks across multiple nodes. Having a low bandwidth or a high delay in time it takes for data to be transferred between nodes can drastically increase training time of a neural network. It has become industry standard to use fast networking with bandwidths of up to 100Gbps this makes the requirements for training across multiple nodes very expensive.
2. Research methods
3. is viable

1 Introduction

Distributed data parallelism is a widely used technique for training artificial intelligence (AI) models, especially when working with large datasets. It involves dividing the data across multiple nodes and training multiple copies of the model in parallel, which can significantly reduce the time required for training. However, the efficiency of this technique can be affected by several factors, including the bandwidth of the communication channels between the nodes.

Low WiFi bandwidths can have a significant impact on the training time of AI models using distributed data parallelism. When the communication channels between nodes have low bandwidth, it can lead to slower data transfer and synchronization between the nodes, which can result in slower training times. In addition, low WiFi bandwidths may also lead to increased latency in the communication between nodes, further diminishing the efficiency of the training process.

In this paper, we aim to explore the relationship between low WiFi bandwidths and the training time of AI models using distributed data parallelism. We will discuss the various ways in which low WiFi bandwidths can impact the communication between nodes and the overall efficiency of the training process. We will also present potential solutions for mitigating the negative effects of low WiFi bandwidths on model training time.

To provide a better understanding of the role that WiFi bandwidth plays in the training of AI models using distributed data parallelism, we will conduct experiments using a variety of datasets and model architectures. We will compare the training time of models trained using distributed data parallelism in high and low WiFi bandwidth environments to evaluate the impact of low WiFi bandwidths on the training process.

Overall, our goal is to shed light on the importance of WiFi bandwidth in the training of AI models using distributed data parallelism and to identify strategies for optimizing the training process in low bandwidth environments. We hope that the findings of this research will be useful for practitioners seeking to improve the efficiency of their model training process and for researchers studying the optimization of distributed data parallelism for AI model training.

2 Methodology

Distributed Data Parallelism

The goal of distributed data parallelism is to split the data across the GPUs and computers in the cluster, so that the training process can be completed faster by leveraging the additional computational resources.

Here is an overview of the process:

1. First, the data and model are partitioned or ”split” across the GPUs and computers in the cluster. This is typically done by dividing the data into smaller chunks and assigning each chunk to a GPU or computer in the cluster.
2. Next, the training process begins. On each GPU or computer in the cluster, the model processes the data assigned to it and computes the gradients of the loss function with respect to the model parameters.
3. The gradients computed on each GPU or computer are then averaged together, and the model parameters are updated using this average gradient. This process is known as ”gradient averaging” or ”gradient synchronization.”
4. The process is then repeated, with each GPU or computer processing its assigned data and computing gradients, until the model has been trained.

By training the model in this way, the computation is distributed across multiple GPUs and computers, which can significantly reduce the time it takes to train the model. However, it also adds complexity to the training process, as the gradients must be averaged across the GPUs and computers and the model parameters must be kept consistent across the cluster. There are multiple ways of preforming this gradient averaging depending on the architecture of your GPUS and computers. In this paper we work with a synchronous architecture(all gpus have roughly the same comute power) for simplicity, but there is also the argument of a centralised versus decentralised topology.

In a centralized topology, the process of gradient averaging or gradient synchronization in step 3 of the distributed data parallelism process is typically performed on a central server or ”parameter server.” This is done by sending the gradients computed on each GPU or computer in the cluster to the central server, where they are averaged together and the model parameters are updated.

In a decentralized topology, the process of gradient averaging or gradient synchronization is performed in a decentralized manner, without the use of a central server or parameter server. This can be done using a number of different techniques, such as peer-to-peer communication or gossip protocols.

We find in this paper —INSERT LINK— that decentralized topologys are effected less by low bandwidths, and that is why we implment a decentralized architecture in our papers. To be more specific, we use our own implementation of the Baidu Ring AllReduce algorithm to average our gradients.

Gradient Quantization

Model gradients are typically stored using 32-bit or 64-bit floating point values, in order to reduce the amount of data sent over the network we quantize our gradients.

When gradients under go out implementaion of quantization thier physical size is halved for example a 32-bit float the gradient will become 16-bit. This will reduce the accuracy of the model but will theoritcally half the communication time.

Gradient Sparcification

Gradient sparsification is a technique that involves identifying and removing ”zero” or nearly zero gradients from the gradients being transmitted between GPUs or computers in a distributed data parallelism setting. The goal is to reduce the size of the gradients without significantly affecting their quality, so that they can be transmitted more efficiently over the network.

In order to perform gradient sparsification we have to decide on a threshold to determine whether or not a gradient is a zero gradient or not, if the gradient is less than the threshold then it it deemed a zero gradient. In our implementation of gradient sparcification we accumulate zero gradients locally and sum them with gradients from the next iteration.

Larger Batch Sizes

Increasing the batch size in a distributed training setup can also improve training efficiency by reducing the frequency of the gradient averaging step. This is because each device will process a larger amount of data before the gradients are averaged and applied to the model weights. This can reduce the overhead associated with performing the gradient averaging step, and can allow the model to make more efficient use of the available computation resources.

However, it is important to carefully tune the batch size, as increasing it too much can negatively impact the convergence and accuracy of the model. This is because a larger batch size can lead to a more noisy gradient estimate, which can make it more difficult for the model to learn effectively.

3 Results

4 Conclusion