Asynchronous Distributed Deep Neural Network Training at Scale

The y^2 Cluster, Carnegie Mellon University

Introduction

The Rise of the Machine

Recently, **Machine Learning** has seen a **massive increase in popularity**. A big part of this success came from the progresses made on neural networks and their possible applications in highly difficult to model fields.

2x

The increase in job offering in the AI field between 2016 and 2018 [1].

Problem

102

The *sequential* training time in *hours* for a state-of-the-art speech recognition NN (10'000 neurons).

Neural network can produce amazing results but they **take very long to train**. There exist some strategies to increase the training speed for some kind of neural networks but this is not the case for densely connected NN.

[1] *Demand for AI Talent on the Rise*. 1er mar. 2018. URL : https://www.hiringlab.org/2018/03/01/demand-ai-talent-rise/ (visité le 05/05/2019).

Approach

We decided to implement a solution that would allow **distributing the training** of NN in the cloud. To achieve this, we based our work on [2] which uses **1-bit quantization** to minimize communication which is the bottleneck when training at scale.

[2] Nikko Strom. « Scalable Distributed DNN Training Using Commodity GPU Cloud Computing ». In : (), p. 5. URL : http://www.nikkostrom.com/publications/interspeech2015/strom_interspeech2015.pdf.

Theory

1-bit Quantization

Only send weight changes equal to $\pm\tau$. This alleviates the need to encode the value of the weight change. You then only need to encode :

- Positive or negative change (1 bit)
- Index of the weight to change

$$\underbrace{\quad}_{\pm}\;\underbrace{\underbrace{\quad}\underbrace{\quad}\cdots\underbrace{\quad}}_{\text{index}}$$

Using 32 bits, each delta can encode weight indexes from 0 to $2^{31} - 1 = 4\,294\,967\,295$.

Achieving Convergence ?

1-bit quantization means that updates to the NN have a **coarser granularity**. The big question is to know if this will have a negative impact on the convergence of the training.

In order for the nodes to keep similar copies of the NN, they do not apply the whole computed gradient at the end of the batch. Instead they only apply the changes that they send to the cluster. The not-yet-applied changes are kept in a so-called **residual** array.

About y^2

Web https://yveszumbachcmu.github.io/y2_documentation/
Doc https://y2-documentation.readthedocs.io/en/latest/
Authors

- Yannick Bloem
- Yves Zumbach

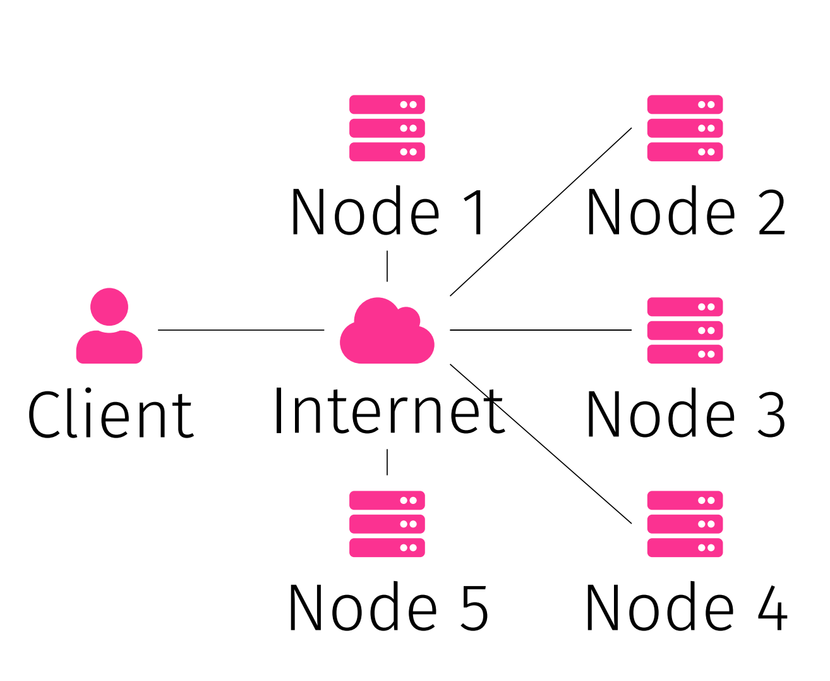
Implementation

y^2 Cluster

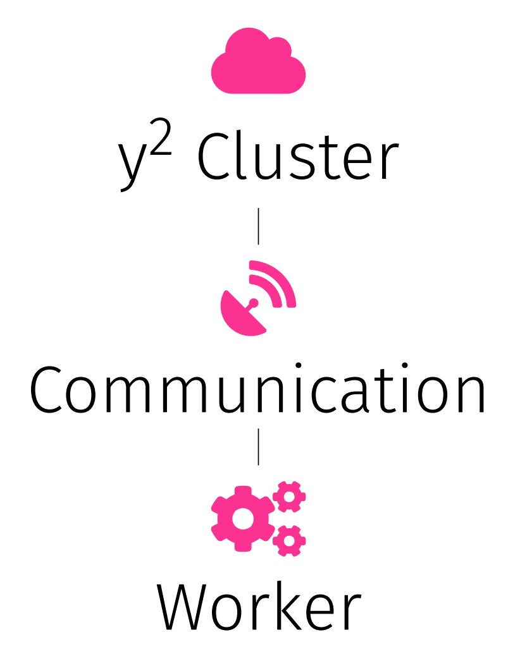
The y^2 cluster is implemented as three services :

Client service Cluster supervisor, starts the training process, assigns data, collects statistics, etc.
Communication service Handles communications between the nodes of the server.
Worker service Performs the actual training of the neural network.

Architecture



Cluster Architecture

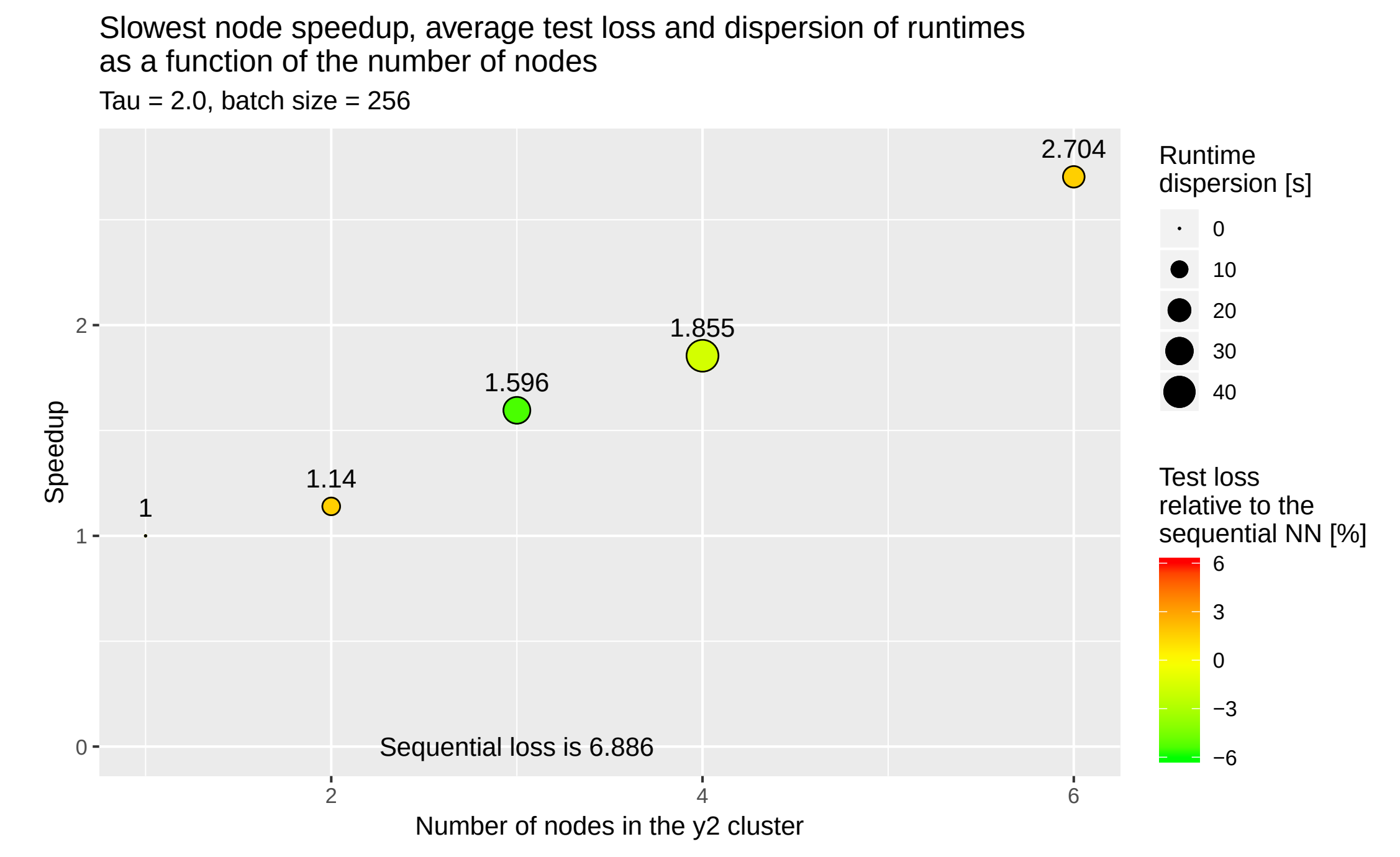


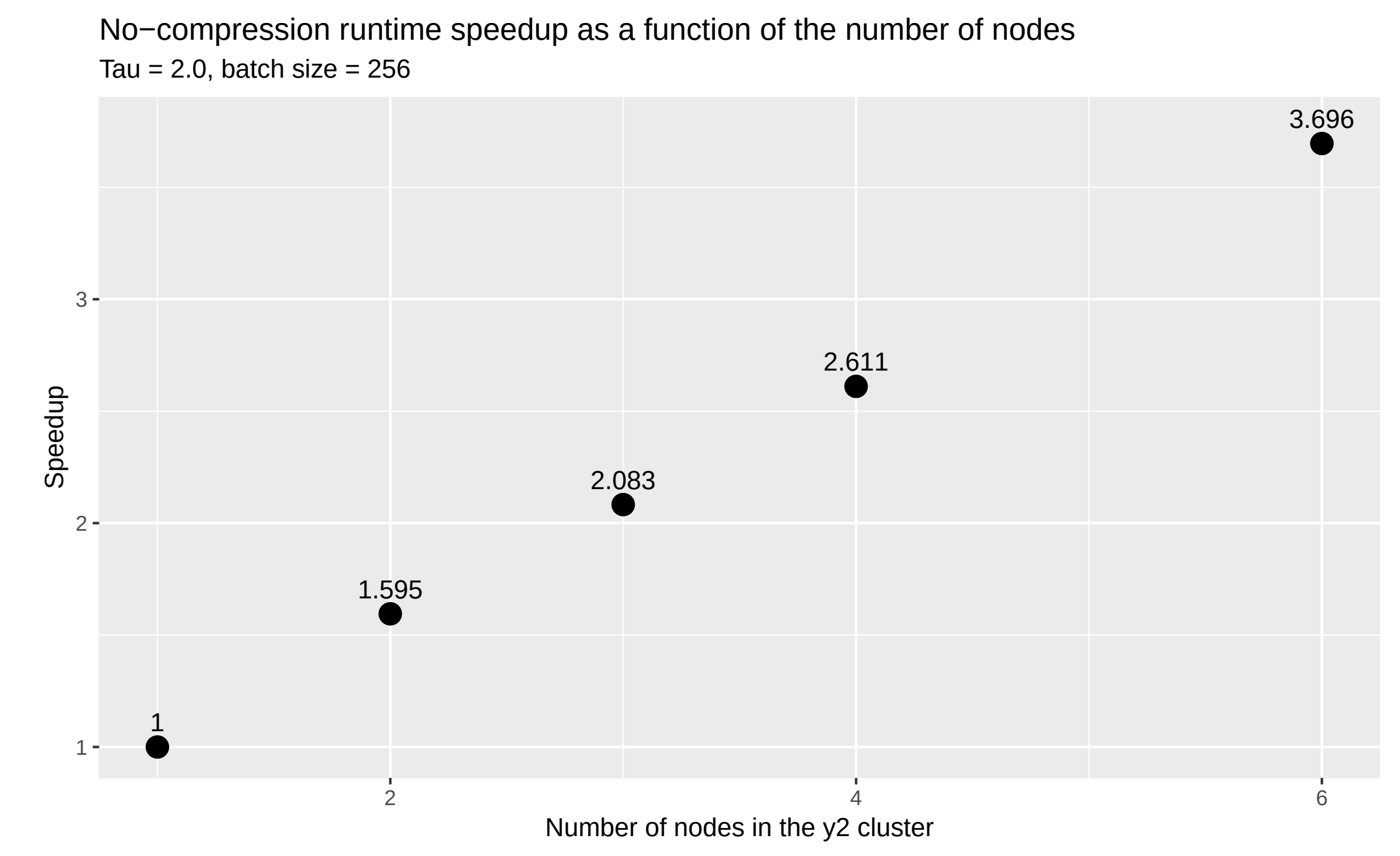
Node Architecture

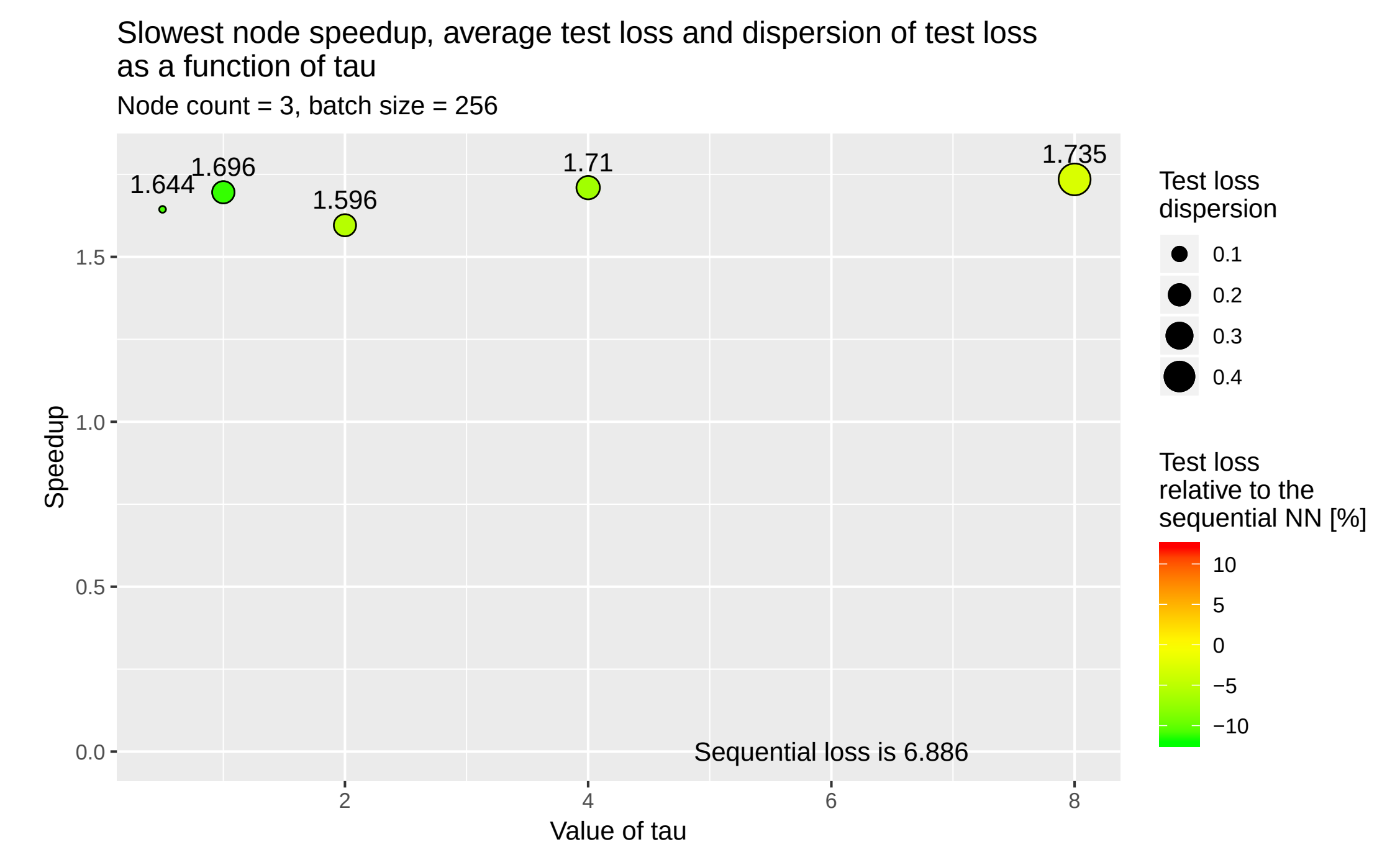
Technology Stack

- Client and communication services : Implemented in Scala, using the Akka library for the clustering and `AsynchronousSocketChannel` for the communications with the worker service.
- Worker service : Implemented in Python, using `pytorch` for the NN, `asyncio` for the asynchronous communication with the communication service.

Results







Conclusion

The graphs above demonstrate that the NN reaches **convergence on all the nodes**. Speedup is good and further implementation improvements could yield much greater results.

2.7x

Achieved speedup with a 6 node y^2 cluster.