Distributed Machine Learning Systems

Prifysgol Abertawe
Swansea University

Christopher Hopkins

Department of Computer Science
Swansea University

This dissertation is submitted for the degree of
Bachelor

October 24, 2020

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 40,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 100 figures.

Christopher Hopkins
October 2020

Declaration

This is where you write your abstract ...

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and Context

Machine Learning algorithms have become ubiquitous in modern life. Powering social media feeds, email spam filters, advertising personalisation and even identifying breast cancer more accurately than doctors. [1] To train these Machine Learning algorithms large datasets are needed. The more data available, the more nuanced and accurate model will be able to be produced

The more nuanced and complex the problem being solved, the more data is necessary. As the scale of problems we are trying to solve dramatically increase, the scale of datasets are too. Services like the Internet Archive as of 2020 contain over 70PB in its database. Labeled datasets such as AViD have 467k videos and 887 action classes, which is in the order of terabytes. [2] Whilst the data grows the as does the Machine Learning models in order to to obtain ever more accurate results. Since 2008 Google has been processing more 20PB of data a day using their Machine Learning MapReduce algorithm. [3].More recently the cutting edge GTP-3 Natural Language Processing model contains 175 Billion parameters. [4] And efforts are being made to create models with trillions of parameters. [5]

Deriving meaning from these vast quantities of data to obtaining nuanced insights from them is a difficult task. Not only because deeper insights into data require larger Machine Learning models. But because more data is needed to populate the parameters of these models. Both of these factors contribute to the need to distribute the computation of the model across multiple nodes otherwise known as Distributed Machine Learning. Distributed Machine Learning is often a pre-requisite for training models, now datasets and models are becoming so large. [6] Scalability is another challenge a Distributed Machine Learning system must face. As more machines are added to a system the communication overhead increases. Communication bandwidth is a limited resource which can produce bottlenecks when the network becomes saturated. With more independent systems working in conjunction and training that costs extensive time and expenses, resiliency has also become a important factor in Distributed Machine Learning. When a Machine fails to work a Distributed Machine Learning System must have contingency plans. These plans can ensure the minimum amount of model data is lost, and that the machine learning can continue swiftly after a major error.

The popular current solution is to have multiple machines compute the model together, communicating the improvements that they've made to each other. The model operates on a worker and parameter server paradigm. The server holds the global model including all the parameters. Workers then use the model in conjunction with the training data split between them to update the global model in the parameter server. [6]

However this model is not perfect and has 3 considerable drawbacks:

- Workers either spend too much time being idle or perform redundant computation, even when the system is under full load. [3]
- Every worker must communicate with a single parameter server, this limits scalability as eventually the network bandwidth becomes saturated severely impacting performance. [6]
- Many parameter server models require the whole model to be replicated within each node. [7] This means that very large models simply cannot run in machines with low system memory.

1.2 Aims

My solution to address these issues raised above and a number of others is to introduce a new model for Distributed Machine Learning: *RingTMP*. RingTMP (Ring Topological Model Parallel) is a Ring Topological Model Parallel Distributed Machine Learning framework focusing on optimising Distributed Stochastic Gradient Descent. This is a novel design drawing in inspiration much research but particularly from the STRADS and Distbelief machine learning frameworks, [8, 9] in addition to identifying issues with existing systems and addressing them. These are the benefits my system will have over existing parameter server architecture:

- Workers will have less idle time, because the work will be distributed more proportionally between nodes. And computing resources will be used more efficiently.
- There will less communication across nodes as the parameter server so not need to send their local weights to the global parameter server each iteration.

- There will also be a potential for larger communication bandwidth, as with RingTMP ring topology only adjacent nodes need to communicate. While with a parameter Server model every worker needs to communicate with the Parameter server at the end of each iteration, which can lead to bottlenecking.
- The nodes require no scheduling as in some model parallel systems. Instead scheduling is managed in a decentralised manner via communication with adjacent nodes. This also makes the system more resilient, in the case of the scheduler crashing.

My aims more specifically for this project are to:

- Create a prototype RingTMP framework and run it multiple machines simultaneously.
- Create a parameter server model framework using the same tools and run that over multiple machines simultaneously.
- Show that RingTMP reduces the time workers are idle in comparison to the parameter server model.
- Exhibit that per iteration RingTMP makes more progress than the parameter server.
- Display that RingTMP is more scalable than a generic parameter server.
- Show that this solution can be as resilient as a parameter server model distributed neural network.
- Demonstrate that RingTMP can hold larger models on comparison to a standard parameter server, where each worker hold the whole model.

1.3 Related Work

In order to understand the benefit this system will have over other Distributed Machine Learning Systems, one must first understand their different variations. While also understanding the benefits and drawbacks.

There are two main variations with respect to the operation of the workers in parameter server model: 1) The parameter server has to wait for the last worker to be finished before

it can calculate the new global parameters. much like the MapReduce algorithm. [3] This is known as *BSP* which stands for Bulk Synchronous Parallel Design. 2) The workers operate asynchronously constantly updating the parameter server, the parameter server calculating new global parameters periodically. [10] This is known as *ASP* which stands for Asynchronous Parallel Design. Whilst this method is the most common method of machine learning with many benefits, there are 3 key drawbacks:

- The model sacrifices efficiency in either time or computation. Either it must wait for all workers to be done each round, or redundant computations must be made. [11]
- when the parameter server is calculating the new global parameters the workers are idle or otherwise computing on stale data. [12]
- Each time the parameter server calculates a new global parameters, these parameters must be broadcast to each worker simultaneously, consuming vast network bandwidth. [6]

As has already been addressed frequently models can get so large that they can no longer feasibly be held within one worker. Therefore there are also variations with respect to how much of the model each worker operates on: 1) The model is segmented and split between worker. This is known as *Model Parallelism* 2) The data is split between the workers which have their own full local models, but are synced with each other at periodic intervals. This is known as *Data Parallelism* [13] Though model parallelism shows promise it has its own set of drawbacks:

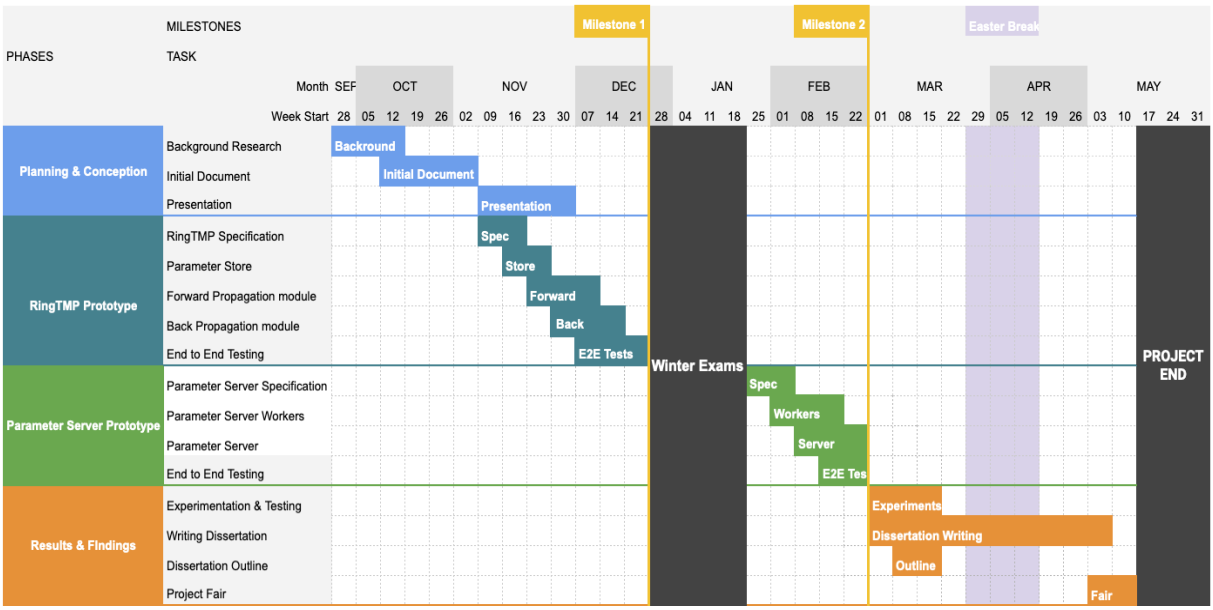
- Often in model parallelism nodes do not communicate with each other, this makes it performing algorithms such as Stochastic Gradient Decent difficult as clusters of nodes are isolated.
- some model parameters take more algorithmic iterations to converge than others, so that they converges at the same rate, this means that some nodes may be idle, not spreading the load equally or efficiently. [9]
- Because some parameters converge at different rates, a scheduler is needed. However this in turn require more computational overhead and communication and reduces iteration throughput. [8]

Taking all of these into consideration, I believe that I could design a system that avoids these limitations, therefore creating a efficient, scalable and resilient system.

2 Background Research

Hello there

3 Project Plan and Time Management



The gant chart above outlines the order and time frame each task should take in order complete this dissertation on time and to a high quality. After researching the background and submitting the initial document. Building a RingTMP prototype of my system is the most important task to embark on. It is better to do this first as it will take the longest time to produce and has the most 'unknown unknowns', moreover this is the centrepiece of my project if I have not finished this there will be no dissertation to write. I then take a month break to revise for my exams. I acknowledge that this is a long time, however semester 1 modules contribute more to my final grade than my dissertation does, therefore its important this project isn't detrimental to my grades in other modules. After Christmas I'll start work on a basic prototype of a parameter server. This will use the same tools and languages as my framework. In this way it will be easy to compare and contrast the benefits and shortcomings of each system on a level playing field. Once Both pieces of software have been complete I will run various tests to see how well my aims have been achieved. While I'm conducting these tests I shall also be writing up my results. Once my experiments have finished I shall start working on my dissertation document in earnest. I'll hand in my outline a week before easter break and shall keep working till it is complete, hopefully sometime before the project deadline (TBC) sometime in May.

3.1 Applied Methodology

My project lends itself to using an experimental research methodology to collect my findings, with this in mind is paramount to ensure that my results are taken is as controlled conditions as possible. I will achieve this by creating two software prototypes, one is my new novel machine learning framework, the other is the established parameter server design. I will make these using the same languages and tools as each other. Only the architecture will differ, meaning comparisons of the two systems will only reflect the performance of the respective architectures. On these systems I will conduct a series of tests. Each of these tests will compare each system to one another and will correspond to the aims I outlined in the introduction:

- To compare the efficiency of the systems we can measure the idle time of each worker in each of the systems and divide that by the time each system took to train a model on some basic task. This will give us a percentage of how much time the machine spend processing and how much time was spent on communication between nodes. This same experiment can be repeated with a different amount of nodes to see how the results change.
- To compare the progress made per iteration, the loss function can be measured for each iteration on both of the systems, then it is trivial to see which ones makes the most progress.
- To measure scalability both systems can be run with a varying number of different nodes. In all experiments they are given the same dataset to be trained on. We can measure how long it takes for them to complete the task, and how well both systems scale when more nodes are added.
- To measure resilience a node can be permanently or temporarily removed from each system while a model is being trained. Then the success of its mitigation and recovery strategies can be assessed.
- To discover the limits of how large a Neural Network each system can hold. Tests which incrementally increase the amount of layers and number of layers until the system crashes can be run.

a	Hello there henuy	what yo doing	c	v

By doing all this I can measure the performance between these two systems and accurately assess my architecture's performance relative to the parameter server.

3.2 Risk Assessment

A dissertation is a long arduous process in which many things could go wrong. Hence it is best to identify possible hazards and decide how to mitigate them before they happen.

Risk	Likelihood	Potential Impact	Mitigation
Completed	The requirement is implemented or fulfilled		
Partial	The requirement is partially implemented or fulfilled		
Not Completed	The requirement is not implemented or fulfilled		
FR	Functional Requirement		
NFR	Non Functional Requirement		
FS	Functional Specification		
NFS	Non Functional Specification		

References

[1] S. M. McKinney, M. Sieniek, V. Godbole, J. Godwin, N. Antropova, H. Ashrafian, T. Back, M. Chesus, G. S. Corrado, A. Darzi, M. Etemadi, F. Garcia-Vicente, F. J. Gilbert, M. Halling-Brown, D. Hassabis, S. Jansen, A. Karthikesalingam, C. J. Kelly, D. King, J. R. Ledsam, D. Melnick, H. Mostofi, L. Peng, J. J. Reicher, B. Romera-Paredes, R. Sidebottom, M. Suleyman, D. Tse, K. C. Young, J. De Fauw, and S. Shetty, "International evaluation of an ai system for breast cancer screening," *Nature*, vol. 577, pp. 89–94, Jan 2020.

[2] A. Piergiovanni and M. S. Ryoo, "Avid dataset: Anonymized videos from diverse countries," 2020.

[3] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, p. 107–113, Jan. 2008.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[5] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," 2020.

[6] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, (Broomfield, CO), pp. 583–598, USENIX Association, Oct. 2014.

[7] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," 2018.

[8] J. K. Kim, Q. Ho, S. Lee, X. Zheng, W. Dai, G. A. Gibson, and E. P. Xing, "Strads: A distributed framework for scheduled model parallel machine learning," in *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, (New York, NY, USA), Association for Computing Machinery, 2016.

[9] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1223–1231, Curran Associates, Inc., 2012.

[10] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 1223–1231, Curran Associates, Inc., 2013.

[11] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, (Broomfield, CO), pp. 571–582, USENIX Association, Oct. 2014.

[12] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Comput. Surv.*, vol. 53, Mar. 2020.

[13] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: A new platform for distributed machine learning on big data," *IEEE Transactions on Big Data*, vol. 1, no. 2, pp. 49–67, 2015.

Appendices

A one

B two