QPECS Project Report
Dynamic Batch Size Scaling for Reduced Wait Time

Jackson Campolattaro
Faculty EWI
TU Delft
Delft, Netherlands

Sian Hallsworth
Faculty EWI
TU Delft
Delft, Netherlands

Yiwei Tao
Faculty EWI
TU Delft
Delft, Netherlands

***Abstract*—Neural network training is increasingly being done using rented time on large distributed clusters. These clusters are expensive to operate, so making efficient use of their hardware is a subject of great importance. This is a difficult challenge, because the usage of these clusters are not coordinated, so training jobs can arrive stochastically. The goal of this paper is to apply dynamic batch size switching during the training process to help training jobs make effective use of a fixed pool of resources.**

***Index Terms*—Deep Learning, Distributed Computing, Hyper-parameter Optimization, Batch-size Switching, Scheduling**

I. Introduction

As neural networks and the datasets they are trained on grow larger, distributed systems have become increasingly necessary for their training. This is done using Data-Distributed learning, where individual cluster nodes train the complete model on subsets of the complete dataset, with data shared in between steps to keep the nodes synchronized. The large distributed training clusters used for this are unlikely to be monopolized by a single user, so it is common to split time and hardware between multiple training jobs.

Modern neural network architectures have found use in industry as general-purpose learners which can be trained on different datasets. Alongside improvements in hyperparameter tuning techniques, this has reduced the labor involved in producing a trained network, and shifted the balance of time towards training. As a result, it has become more valuable for networks to be trained quickly, and to make effective use of the available hardware.

In practice, neural networks are often used to augment human labor, and improve efficiency. In these cases, network accuracy may not be especially high, and small differences in network accuracy are less important. Because of this, there are scenarios where it may make sense to sacrifice a small amount of accuracy for much improved network training times and hardware costs.

In this work, we study how the parameters such as batch size, number of epochs, and number of nodes influence the DL training process. We build on these results to examine how accuracy can be traded for performance by dynamically modulating batch size during training. We propose a solution to maximize the utilization of nodes in a cluster of fixed size when training deep neural networks using variable batch sizes.

The article is outlined as follows:

- Section II covers existing research in the domain of batch-size-switching, and discuss how it can be applied to our use case.
- Section III reports on the experimental setup used to collect data, and visualizes some of the results. This section also discusses the analytical techniques we used to confirm the statistical significance of our parameters.
- Informed by the results of our analysis in section III, section IV builds a set of predictive models which relate our parameters to the service time of a long training job. It also proposes a model which relates these parameters to average response times, waiting times and utilization of smaller training jobs with stochastic arrival times.
- Section V applies the models developed in section IV to guide the design of an optimized system which uses variable batch sizes in order to maximize the utilization of a fixed number of nodes when training deep neural networks.

Finally, in section VI we reflect on the challenges we experienced during our research, and discuss possibilities for future extensions.

II. Background

A. Cluster Utilisation

In a 2018 report, Jeon et. al. investigated three different factors that affect cluster utilisation when training DNN jobs. They argued that there are two possibilities for queuing delays, either waiting for the sharing of resources to become available or waiting for locality constraints to be satisfied. [5]

B. Dynamic Batch-size

To optimise cluster utilisation, we found several works on the topic of dynamic batch-size.

A 2020 article by Ye et. al. considers dynamic batch size can be a solution for the poor cluster utilisation problem since it can balance the load. Rather than changing the synchronous or asynchronous approaches, they dynamically split the dataset based on the previous epoch performance, which they called Dynamic Batch Size (DBS) strategy. [7]

Similarly, a 2019 paper by Hu et. al. also proposed a variable batch-size strategy by dividing the training process into two stages, using different batch sizes. Their approach aimed to

alleviate poor test accuracy caused by large batch-size training. Defining of larger than 1k, large batch-size can reduce iterative steps so as to speed up the convergence of training process, while in practical cases, sacrificing accuracy to achieve this goal is not applicable. To reduce the impact of large batch-size, small batch-size was used at first and stepping up with auto-tuning hyper-parameters after a certain number of epochs. Both their methodology inspired us to optimize our system by adapting batch-size to job arrival rate. [4]

In another 2019 paper, Liu et. al. propose a method of dynamically choosing the batch size with multiple levels. It selects between a set of reasonable batch size values in order to maximize training efficiency without sacrificing accuracy. [6] An earlier 2017 report by De et. al. proposes another method where batch sizes are automatically selected, but uses a more elaborate strategy in which the batch size is smoothly increased with each training step. In this case, batch sizes are selected automatically, and the goal is to grow the batch size to ensure a constant signal-to-noise ratio during training. [1]

III. Experimental Evaluation

Motivated by the existing works mentioned in Section II, we selected the following 3 factors in our experiments:

1) **Batch-size**: The size of the random data slice that is used for each iteration of training.
2) **Number of epochs**: The number of times that the learning algorithm will be applied to the entire training dataset, helping the DNN learn the structure of the data.
3) **Number of nodes**: The number of independent Google Cloud Platform (GCP) nodes used by the distributed learning cluster.

A. System Configuration

We run the experiments separately in two types of configuration:

1) The batch mode deploys all jobs at once and runs experiments one by one, which allow us to get ideas about service time of the system since it has nothing to do with queuing. The levels chosen for each of these factors can be seen in table I.
2) The simulated mode sends jobs to the system following the Poisson distribution, which allows us to make a queuing theory model of the system. As a result, the effect of the inter-arrival time on the waiting time of jobs can be investigated. The levels of factors used in these experiments are shown in table II.

Aside from those levels, all other parameters remained fixed. The fixed properties of all jobs are described in table III.

TABLE I: Levels of Factors used for batch-mode experiments

Factors	**Levels**		
Batch-size	32	128	
Num. of epochs	40	80	
Num. of nodes	1	2	4

TABLE II: Levels of Factors used for simulation experiments

Factors	**Levels**	
Batch-size	32	128
Num. of epochs	5	10
Num. of nodes	1	2
Inter-arrival time	90	120

TABLE III: Values of Fixed Parameters for all experiments

Factors	**Levels**
Cores	3
Memory	2GB
Network	FashionMNISTCNN
Dataset	MNIST
Optimizer	Adam
Learning Rate	0.001

Tests were run on '`e2-highcpu-8`' nodes in the Google Cloud using a modified version of the kubernetes-based distributed training tool FLTK [3]. The tests run in 'batch' mode were carried out in order to examine the performance of each job independently. For each configuration, data from 3 replications were collected and the resulting service time recorded. In the case of the simulated mode, the response time was also recorded. This data makes up the test dataset that is used later on to create models.

The average of all 3 replications is shown for each of the configurations as a set of bar charts. The effects of different epoch counts can be seen in fig. 1. The effects of different batch sizes can be seen in fig. 2. At first glance, all effects appear linear, implying that we should be able to achieve high accuracy with a linear regression model. This will be further explored in section IV.

B. Full Factorial Analysis

Based on the selected factors, we performed full factorial analysis to ascertain the relative importance of each of these factors and determine which levels of the factors would result in an optimal service time. Having 2 levels for batch-size, 2 levels for the number of epochs and 3 levels for the number of data parallelism and replication of each setting 3 runs, 36 experiments are conducted in total. The experiments use all possible factor combinations to fully examine all the possibilities; with respect to the chosen factors and levels.

In table IV, the results of the ANOVA analysis are presented. These results show that all three factors, $batch\_size$, num_nodes and num_epochs, as well as the combination $nodes : batch\_size$ and $nodes : epochs$, have p-values less than the significance level 0.05. This means that they have

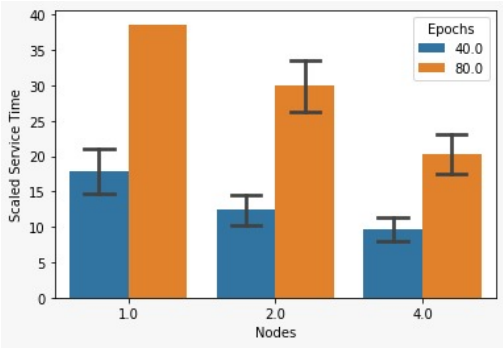

Fig. 1: The effects of increasing epoch count, for each node count

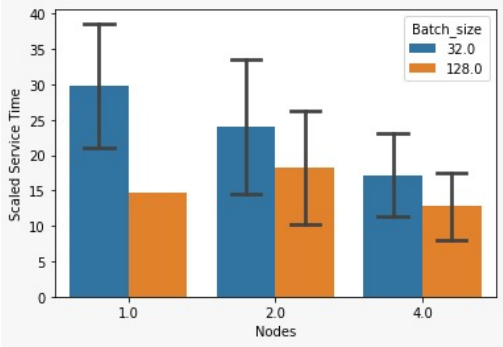

Fig. 2: The effects of increasing batch size, for each node count

a high contribution to the variation in the response variable, compared to that of the errors, at 95% level of confidence. As a result, we confirm that there are only statistically significant associations between the aforementioned factors/interactions and the service time. Ergo, the other factors and interactions are not relevant for modelling the service time.

Subsequently, t-tests were used to determine whether the factors show significant differences in their levels. The results for node count, epoch count, and batch size can be seen in table V, table VI, and table VII, respectively. None of these confidence intervals include zero, confirming that there is significant variation between levels with 90% confidence.

IV. Predictive Models

A. Multiple Linear Regression Model

A linear regression model was used to make predictions of the system service time. It is notable to mention that in this case, the service time of the system is equal to the response time, since no queuing is involved.

Two different regression models were constructed:

1) based on all three factors ($batch\_size$, num_nodes and num_epochs) as well as their combined interactions;
2) based on only the factors and interactions which passed our ANOVA F-test and t-test as described in the previous section ($batch\_size$, num_nodes, num_epochs, $nodes : batch\_size$ and $nodes : epochs$).

These two models were used to predict the service time for the test dataset. Additionally, the significance of their predictions at 90% confidence interval as well as the coefficient of determination (R^2) of each model were calculated. The results are summarised in table VIII.

TABLE VIII: Regression Model Prediction Results

Prediction of Response Time	R^2	T-test Result
Without ANOVA analysis	93%	True
With ANOVA analysis	93%	True

Both of these models achieved an R^2 score of 93%, which indicates that the regression model is a good fit. However, this result is interesting because even though the two models give the same coefficient of determination, they generate different when predictions for the response time. A comparison of their predictions can be found in fig. 3. Here it can be seen that the model which ANOVA analysis into account (model 2), results in smaller confidence intervals. This shows that model 2 gives a better estimation of the service time.

The full parametrization of our final regression model (model 2), can be found in table IX. These coefficients result in the following function for service time:

$$\mathbf{E[S]} = \mathbf{X} \begin{bmatrix} 2.5815 \\ -9.0432 \\ 0.653 \\ -0.5977 \\ 0.8777 \\ -0.0757 \end{bmatrix} \tag{1}$$

where **E[S]** is the matrix of expected service times, **X** is the matrix of corresponding factor values and the coefficients are in the order of [b_0, $b_{\text{batch\_size}}$, b_{epochs}, b_{nodes},$b_{\text{nodes:batch\_size}}$, $b_{\text{nodes:epochs}}$]. The results of this regression model can be found in Appendix A.

In order to prove the validity of this model its initial assumptions are checked against the its results. There are four assumptions which have to be met. These assumptions and their validity with relation to model 2 are discussed below.

1) **Linear relationship**: *The response variable scales linearly with the model factors.*

   From fig. 1 and fig. 2 the relationship between the results of the predicted response time and factors can be observed. For Node Count, we see that the response time is linear with respect to the levels, and because of the way in which levels were selected, this indicates that it is linear with respect to the square root of the number of nodes. However, for the Batch Size and Epochs factors, these results are not informative because these factors were investigated at two levels, which can only show a linear relationship. This prevents us from confirming how well this model can generalize to other levels for these factors. In spite of this, we confirm that the factors have a linear relationship with the response variable

TABLE IV: ANOVA for the levels shown in table I

	df	sum_sq	mean_sq	F	PR(>F)	significance(0.05)	% variation
Batch_size	1.0	565.075596	565.075596	66.346790	1.689551e-08	True	19.0
Nodes	1.0	417.367314	417.367314	49.004030	2.448059e-07	True	14.0
Epochs	1.0	1688.231103	1688.231103	198.218990	2.175328e-13	True	55.0
Nodes:Batch_size	1.0	91.565048	91.565048	10.750857	3.060993e-03	True	3.0
Epochs:Batch_size	1.0	22.889840	22.889840	2.687547	1.136604e-01	False	1.0
Nodes:Epochs	1.0	55.509854	55.509854	6.517536	1.716635e-02	True	2.0
Nodes:Epochs:Batch_size	1.0	0.022650	0.022650	0.002659	9.592821e-01	False	0.0
Residual	25.0	212.924996	8.517000	NaN	NaN	False	7.0

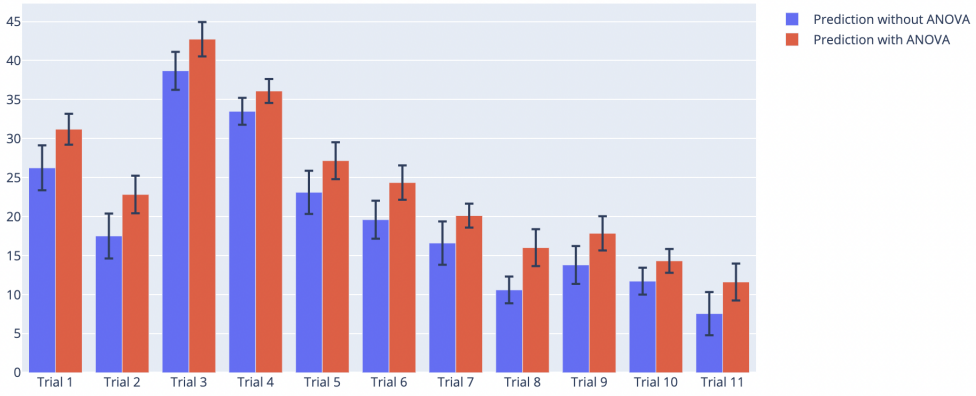

Fig. 3: Response time predictions using linear regression, with and without ANOVA adjustment

TABLE V: Confidence interval for each node count level

	lower_bound	upper_bound
1.0	4.091693	5.708164
2.0	0.429842	2.046313
4.0	-5.721259	-4.104788

TABLE VI: Confidence interval for each epochs level

	lower_bound	upper_bound
40	-6.99011732	-6.18188206
80	7.499082	8.30731726

TABLE VII: Confidence interval for each batch size level

	lower_bound	upper_bound
32	3.3733929	4.18162815
128	-4.93713026	-4.128895

since a linear regression model can be used to explain 93% of the variation in the response variable.

2) **Independence**: *The residuals are independent and there is no association between consecutive residuals.*

   As shown in fig. 4, there is not a clear relationship between the residuals and predicted service time. This also holds true for the experiment number, as can be seen in fig. 5. In both of these plots, the points are randomly scattered which strongly suggests that our errors are independent.

3) **Homoscedasticity:** *At each level, the variance of the residuals is constant.*

   As shown in fig. 4 and fig. 5, rather than becoming spread out as values get larger, standardized residuals generally remain at the same level. There are however, a few noticeable deviations from the general trend. Nevertheless, the homoscedasticity condition is still met since the distribution of standardized residuals fit the expected ranges as stated by Dean et al. [2]. The expected distributions are as follows: 68% of standardized residuals in the range $[-1, +1]$, 95% in the range $[-2, +2]$ and 99.7% in the range $[-3, +3]$.

4) **Normality:** *The residuals must be approximately normally distributed with respect to response time.*

   The Quantile-Quantile plot shown in fig. 6 nicely follows the 45-degree best-fit line. While there are some clear outliers, the outliers are balanced on either side of the best-fit line. Therefore, the normality assumption is considered met.

TABLE IX: Regression Model Parameters

	estimate	lower_bound	upper_bound	width CI/2	t_test_pass
Intercept	2.054575	1.626849	2.482301	0.427726	True
Batch_size[T.128.0]	-7.197333	-7.625059	-6.769608	0.427726	True
Nodes	-0.475719	-0.903445	-0.047993	0.427726	True
Nodes:Batch_size[T.128.0]	0.698534	0.270808	1.126259	0.427726	True
Epochs	0.519707	0.091982	0.947433	0.427726	True
Nodes:Epochs	-0.060307	-0.488033	0.367418	0.427726	False

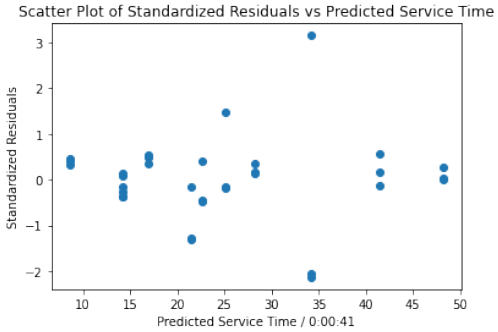

Fig. 4: Residual errors plotted against service times

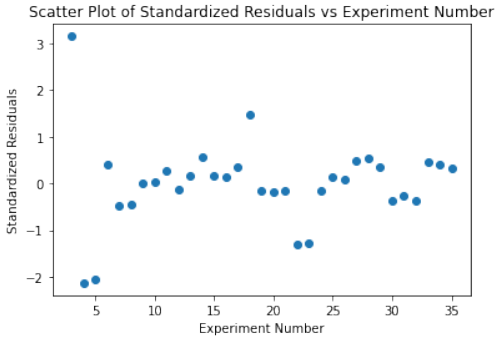

Fig. 5: Residual errors plotted against experiment number

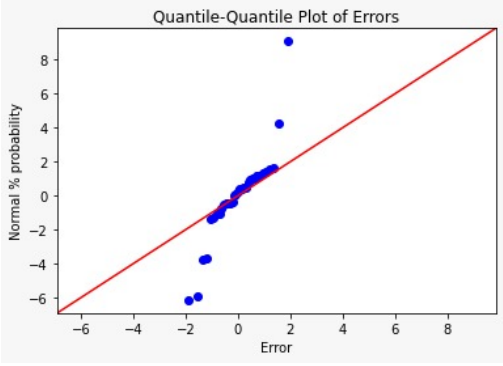

Fig. 6: Quantile-Quantile plot of errors with 45-degree best-fit line in red

B. Operational Laws & Queuing Theory Models

1) Expected Utilization: According to the utilization law (equation 2), the utilization is related to the service rate and arrival rate.

$$\rho = \frac{\lambda}{\mu} = \lambda \cdot E[S] \tag{2}$$

Therefore, the model to predict the service time in Section IV-A can be expanded upon to build a prediction model for utilization. This model is given by:

$$\mathbf{E[U]} = \lambda \mathbf{X} \begin{bmatrix} 2.5815 \\ -9.0432 \\ 0.653 \\ -0.5977 \\ 0.8777 \\ -0.0757 \end{bmatrix} \tag{3}$$

where **E[U]** is the matrix of predicted utilizations, **X** is the matrix of corresponding factor values and the coefficients are in the order of [b_0, $b_{\text{batch\_size}}$, b_{epochs}, b_{nodes},$b_{\text{nodes:batch\_size}}$, $b_{\text{nodes:epochs}}$].

Since the service rate is an input parameter of the experiments, it has no error. Therefore by error propagation, the confidence intervals found in for $E[S]$ also apply to this utilization model.

2) Expected Queuing time & Response time: In this section models for predicting the queuing time and response time are described. This model uses the test data from the simulated experiments and considers the response time as:

$$E[T] = E[S] + E[W] \tag{4}$$

where $E[T]$ is the expected response time of a given job, $E[S]$ is the expected service time and $E[W]$ is the expected time spent in the queue.

Figure 7 shows the relationship between service time and response time for each experiment setting. When the arrival interval is small, jobs can go directly to the server without waiting, so there is no queuing problem. When the arrival interval becomes large, on the other hand, the difference between service time and response time becomes significant; especially if the service time is also large. These results correspond with behaviour expected from equation 4.

An M/M/1 queuing model is used to describe the experiment set up. This model is suitable because both the arrival rate and service rate distributions belong to the exponential distribution family. From fig. 8 it can be observed that the arrival rates of

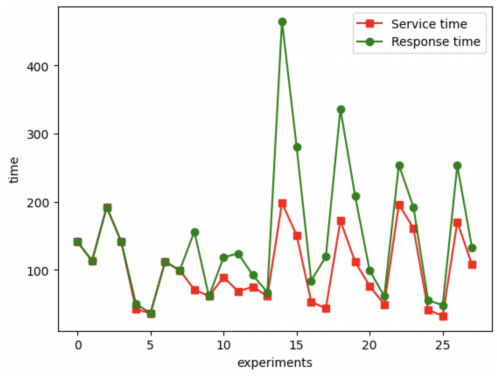

Fig. 7: Service time and Response time

all jobs are distributed according to a Bernoulli distribution. While fig. 9 shows that the service rates are roughly exponentially distributed.

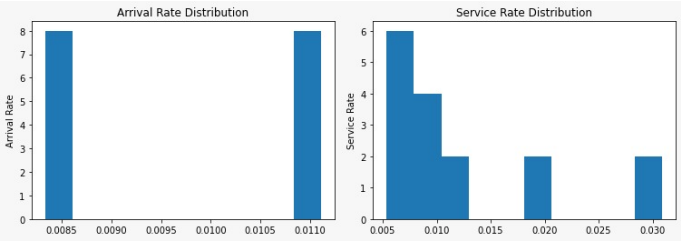

Fig. 8: Histogram of the arrival rates

Fig. 9: Histogram of the service rates

The expected response time can be found using Little's Law (equation 5) and a Continuous Time Markov Chain (CTMC) to determine the number of jobs in the system from the limiting probability of N jobs being in the system.

$$E[T] = \frac{E[N]}{\lambda} \tag{5}$$

We find that expected number of jobs is defined as:

$$E[N] = \frac{\rho}{1-\rho} \tag{6}$$

where ρ, is the ratio of the arrival rate (λ) to the service rate($\mu = \frac{1}{E[S]}$).

Therefore, the response time is modelled by equation 7. Notably, the regression model from Section IV-A did not work with the simulated dataset. This is likely due to the large discrepancies in the factor levels explored. As a result, a new regression model (model 3) was fitted to this dataset, using the same methods as described in Section IV-A. The coefficients for model 3 can be found in Appendix F.

$$\begin{gathered} E[T] = \frac{1}{\frac{1}{E[S]} - \lambda} \\ E[S] = 109.63 - 47.75\text{bs}_{128} - 67.62\text{n} + 12.75\text{n:bs}_{128} \\ +13.15e + 1.45\text{n:e} \end{gathered} \tag{7}$$

where bs_{128} is 1 when batch size 128 is used and 0 otherwise, n is the node level and r is the epoch level.

Using the model for response time and service time, the model for waiting time can be stated as:

$$E[W] = \frac{1}{\frac{1}{E[S]} - \lambda} - E[S] \tag{8}$$

Expected values for service time, time in queue, number of jobs in queue and response time can all be found in the Appendices B,D,C and E respectively.

C. Decision Tree Model

A third model was created by training a decision tree regressor to directly predict the response times of simulated runs. Response time was chosen as the target of prediction because it should contain less non-linearity than wait time, as it doesn't "bottom-out" at 0 for configurations with low utilization. The decision tree was created using Sklearn's `DecisionTreeRegressor` package, and trained on the same set of all collected simulation runs that was used for the operational-law based model.

Sklearn's `GridSearchCV` package was used to optimize the hyperparameters of the decision tree, in order to produce the best possible accuracy. A total of 64 different configurations were tried, and the final parameter selection is shown in table X.

TABLE X: Optimal Decision Tree Hyper-parameters

Parameter	Optimal Value
Criterion	`absolute_error`
Max Depth	None
Max Features	`log2`
Splitter	`random`

Performance was validated using Sklearn's cross-validation tooling, and scores represent the average result of all "leave-one-out" tests. Because results occupy a range of magnitudes, it made sense to use mean absolute percentage error as our accuracy metric in place of R^2 error, this naturally corrects for scale.

Unfortunately, with such a small dataset, a decision-tree based method was unable to achieve very high levels of accuracy. The average cross-validation score was 54%, indicating that when trained on every other data point, the model would typically under- or over-shoot the final data point by more than 50%.

It is worth noting that a decision tree trained on the entire dataset can produce 100% accuracy on the dataset, but our cross-validation tests demonstrate well that this can only be an example of overfitting, and would produce a model that is very unlikely to generalize to un-tested configurations.

V. Optimization Strategy

Performance can be improved by a simple variable batch size scheme which splits training into two parts: phase-1 uses small batch sizes during the most critical part of training, and phase-2 uses larger batch sizes to speed up epochs in the long-tail of the training process. Previous research switched batch

sizes based on optimal values for particular hardware, this could be expanded to more varied hardware, and in a cluster setting [4], [6]. Our models indicate that increasing batch size provides a significant benefit to performance.

Performance can be further improved by dedicating separate resources to phase 1 and phase 2 of training. This takes advantage of the fact that certain hardware sees a bigger improvement in performance when batch sizes are increased. This structure is shown in fig. 10. In this diagram, μ_1 is the service rate of the hardware responsible for phase 1 of training, and μ_2 is the is the service rate of the hardware responsible for phase 2. The job arrival rate λ is the same as the arrival rate to phase 2, because jobs should exit phase 1 at a the same average rate with which they arrive.

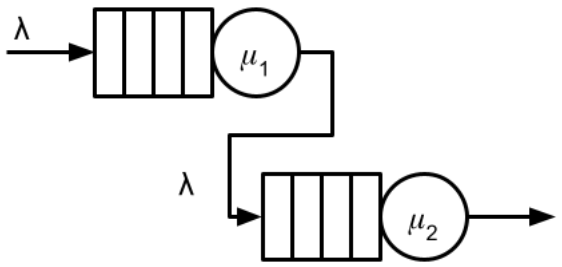

Fig. 10: System design with separate resources for Phase 1 and Phase 2

Note that if phase 1 and phase 2 are instantiated on equivalent hardware, the increased batch size in phase 2 will result in the service rate μ_2 being much larger than μ_1. This would mean that the second queue would seldom grow, and μ_1 would become a significant bottleneck to performance. This bottleneck can be ameliorated by dedicating more hardware to phase 1 than phase 2.

In some cases, it may make more sense to provide phase 2 with different resources than phase 1. For example, GPUs see a much larger speedup from larger batch sizes than CPUs, so if a limited number of GPUs are available it would make sense to allocate them for the second phase. Moreover, the systems do not actually need to be instantiated on separate devices – because a GPU training task makes little use of the CPU and vice-versa, they can share nodes in a cluster which have access to both types of resources.

VI. Conclusions

In section III, we demonstrated that all of our selected factors were significant, as well as many of the interactions between the features. We used this information to build a model based on linear regression in section IV, which was able to predict the service times of individual jobs with a high degree of accuracy. This also informed our design of a model based on operational laws and queuing theory, which allowed us to make predictions about the response times, waiting times and utilization of a more elaborate simulation. Based on the same data we also built a decision-tree based model, but because of the small data-set its accuracy was limited. Finally, we speculated about ways in which these models could inform the design of a system which uses dynamic batch size to make more efficient use of the hardware.

In the course of this project we encountered challenges collecting data. Issues with our use of FLTK prevented us from simulating stochastically arriving tasks for all but a select few configurations. This made the first stage of our analysis difficult, limited our ability to validate our second (operational-law based) model, and prevented us from fully exploring the avenues for optimization we laid out.

In the future, we would like to explore the following optimizations, and determine how they affect performance:

- Switch batch size mid-training during each individual training job.
- Split phase 1 and 2 of training across separate training job types with independent queues, as discussed in section V.
- Perform phase 2 on more appropriate hardware, to fully take advantage of the split structure.

In addition to this, we would like to perform a separate statistical analysis of the accuracy achieved by the configurations in our experiments. This is necessary in order to fully understand the trade-offs being made when we increase batch size or decrease the number of epochs.

References

[1] Soham De, Abhay Yadav, David Jacobs, and Tom Goldstein. Automated inference with adaptive batches. 04 2017.

[2] Angela Dean, Daniel Voss, and Danel Draguljić. *Design and Analysis of Experiments*. 01 2017.

[3] J.M Galjaard. Kubernetes - federation learning toolkit ((k)FLTK).

[4] Zhongzhe Hu, Junmin Xiao, Zhongbo Tian, Xiaoyang Zhang, Hongrui Zhu, Chengji Yao, Ninghui Sun, and Guangming Tan. A variable batch size strategy for large scale distributed dnn training. In *2019 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 476–485, 2019.

[5] Myeongjae Jeon, Shivaram Venkataraman, Junjie Qian, Amar Phanishayee, Wencong Xiao, and Fan Yang. Multi-tenant gpu clusters for deep learning workloads: Analysis and implications. *Technical report, Microsoft Research*, 2018.

[6] Baohua Liu, Wenfeng Shen, Peng Li, and Xin Zhu. Accelerate mini-batch machine learning training with dynamic batch size fitting. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[7] Qing Ye, Yuhao Zhou, Mingjia Shi, Yanan Sun, and Jiancheng Lv. Dbs: Dynamic batch size for distributed deep neural network training. *ArXiv*, abs/2007.11831, 2020.

APPENDIX A

TABLE XI: Expected Service Time as predicted by Regression Model 2

	Nodes	Epochs	Batch_size	estimate	lower_bound	upper_bound	width CI/2
4	2.0	80.0	128.0	34.214068	31.723236	36.704900	2.490832
7	4.0	80.0	128.0	22.650051	19.629474	25.670629	3.020577
10	1.0	80.0	32.0	48.161637	45.396373	50.926901	2.765264
13	2.0	80.0	32.0	41.501941	39.577963	43.425919	1.923978
16	4.0	80.0	32.0	28.182548	25.208355	31.156742	2.974193
19	1.0	40.0	32.0	25.072713	22.307448	27.837977	2.765264
22	2.0	40.0	32.0	21.444000	19.520022	23.367979	1.923978
25	4.0	40.0	32.0	14.186576	11.212383	17.160770	2.974193
28	1.0	40.0	128.0	16.907152	14.141887	19.672416	2.765264
31	2.0	40.0	128.0	14.156128	12.232150	16.080106	1.923978
34	4.0	40.0	128.0	8.654079	5.679886	11.628273	2.974193

APPENDIX B

TABLE XII: Expected Service Time as predicted by Queuing Theory Model

Nodes	Epochs	Batch_size	lambda	E[S] (s)	std(E[S]) (s)
1.0	5.0	32.0	0.008333	115.000	15.450273
1.0	5.0	128.0	0.008333	80.000	15.450273
1.0	10.0	32.0	0.008333	188.000	15.450273
1.0	10.0	128.0	0.008333	153.000	15.450273
1.0	5.0	32.0	0.011111	115.000	15.450273
1.0	5.0	128.0	0.011111	80.000	15.450273
1.0	10.0	32.0	0.011111	188.000	15.450273
1.0	10.0	128.0	0.011111	153.000	15.450273
2.0	5.0	32.0	0.008333	54.625	15.450273
2.0	5.0	128.0	0.008333	32.375	15.450273
2.0	10.0	32.0	0.008333	134.875	15.450273
2.0	10.0	128.0	0.008333	112.625	15.450273
2.0	5.0	32.0	0.011111	54.625	15.450273
2.0	5.0	128.0	0.011111	32.375	15.450273
2.0	10.0	32.0	0.011111	134.875	15.450273
2.0	10.0	128.0	0.011111	112.625	15.450273

APPENDIX C

TABLE XIII: Expected Number of Jobs in Queue as predicted by Queuing Theory Model

Nodes	Epochs	Batch_size	lambda	E[N] (s)	std(E[N]) (s)
1.0	5.0	32.0	0.008333	23.000000	4.369997
1.0	5.0	128.0	0.008333	2.000000	0.546250
1.0	10.0	32.0	0.008333	999.000000	116.107105
1.0	10.0	128.0	0.008333	999.000000	142.667554
1.0	5.0	32.0	0.011111	999.000000	189.809876
1.0	5.0	128.0	0.011111	8.000000	2.184999
1.0	10.0	32.0	0.011111	999.000000	116.107105
1.0	10.0	128.0	0.011111	999.000000	142.667554
2.0	5.0	32.0	0.008333	0.835564	0.334225
2.0	5.0	128.0	0.008333	0.369472	0.249358
2.0	10.0	32.0	0.008333	999.000000	161.839746
2.0	10.0	128.0	0.008333	15.271186	2.962710
2.0	5.0	32.0	0.011111	1.544170	0.617667
2.0	5.0	128.0	0.011111	0.561822	0.379175
2.0	10.0	32.0	0.011111	999.000000	161.839746
2.0	10.0	128.0	0.011111	999.000000	193.812526

APPENDIX D

TABLE XIV: Expected Waiting Time in Queue as predicted by Queuing Theory Model with lower bound and upper bound given at 90% CI

	Nodes	Epochs	Batch_size	lambda	E[W](s)	std(E[W]) (s)	lb E[W] (s)	ub E[W] (s)
53	1.0	5.0	32.0	0.008333	2645.000000	524.627211	1694.133575	3595.866425
54	1.0	5.0	128.0	0.008333	160.000000	67.346179	37.937669	282.062331
55	1.0	10.0	32.0	0.008333	119692.000000	13932.861149	94439.230838	144944.769162
56	1.0	10.0	128.0	0.008333	119727.000000	17120.113413	88697.460022	150756.539978
65	1.0	5.0	32.0	0.011111	89795.000000	17082.895805	58832.915488	120757.084512
66	1.0	5.0	128.0	0.011111	640.000000	197.255882	282.481383	997.518617
67	1.0	10.0	32.0	0.011111	89722.000000	10449.650859	70782.414072	108661.585928
68	1.0	10.0	128.0	0.011111	89757.000000	12840.089126	66484.837645	113029.162355
57	2.0	5.0	32.0	0.008333	45.642686	42.980068	-32.257016	123.542389
58	2.0	5.0	128.0	0.008333	11.961662	33.676308	-49.075337	72.998661
59	2.0	10.0	32.0	0.008333	119745.125000	19420.775640	84545.724179	154944.525821
60	2.0	10.0	128.0	0.008333	1719.917373	355.860749	1074.933601	2364.901145
69	2.0	5.0	32.0	0.011111	84.350265	57.697199	-20.223665	188.924195
70	2.0	5.0	128.0	0.011111	18.188991	37.460386	-49.706503	86.084485
71	2.0	10.0	32.0	0.011111	89775.125000	14565.585315	63375.567886	116174.682114
72	2.0	10.0	128.0	0.011111	89797.375000	17443.134160	58182.372475	121412.377525

APPENDIX E

TABLE XV: Expected Total Response Time as predicted by Queuing Theory Model with lower bound and upper bound given at 90% CI

	Nodes	Epochs	Batch_size	lambda	E[T] (s)	std(E[T]) (s)	lb E[T] (s)	ub E[T] (s)
53	1.0	5.0	32.0	0.008333	2760.000000	524.399657	1809.546009	3710.453991
54	1.0	5.0	128.0	0.008333	240.000000	65.549957	121.193251	358.806749
55	1.0	10.0	32.0	0.008333	119880.000000	13932.852582	94627.246365	145132.753635
56	1.0	10.0	128.0	0.008333	119880.000000	17120.106441	88850.472657	150909.527343
65	1.0	5.0	32.0	0.011111	89910.000000	17082.888818	58947.928152	120872.071848
66	1.0	5.0	128.0	0.011111	720.000000	196.649871	363.579753	1076.420247
67	1.0	10.0	32.0	0.011111	89910.000000	10449.639437	70970.434774	108849.565226
68	1.0	10.0	128.0	0.011111	89910.000000	12840.079831	66637.854493	113182.145507
57	2.0	5.0	32.0	0.008333	100.267686	40.107048	27.575221	172.960152
58	2.0	5.0	128.0	0.008333	44.336662	29.922948	-9.897517	98.570841
59	2.0	10.0	32.0	0.008333	119880.000000	19420.769494	84680.610318	155079.389682
60	2.0	10.0	128.0	0.008333	1832.542373	355.525191	1188.166786	2476.917960
69	2.0	5.0	32.0	0.011111	138.975265	55.590070	38.220425	239.730105
70	2.0	5.0	128.0	0.011111	50.563991	34.125791	-11.287678	112.415661
71	2.0	10.0	32.0	0.011111	89910.000000	14565.577120	63510.457738	116309.542262
72	2.0	10.0	128.0	0.011111	89910.000000	17443.127317	58295.009877	121524.990123

APPENDIX F

TABLE XVI: Coefficients ("b-values") used by Queuing Theory Model

	b estimate	std
Intercept	109.625	66.154117
C(Batch_size)[T.128.0]	-47.750	39.892433
Nodes	-67.625	41.839537
Nodes:C(Batch_size)[T.128.0]	12.750	25.230190
Epochs	13.150	7.978487
Nodes:Epochs	1.450	5.046038