 doi:10.20944/preprints202011.0017.v1

Differentiate Containers Scheduling for DeepLearning Applications

BY

Yun Song

BS, Sichuan University

 doi:10.20944/preprints202011.0017.v1

Contents

List of Tables iv

List of Figures v

Abstract i

1 INTRODUCTION 1

1.1 TRADL 4

1.2 DQoES 5

2 RELATED WORK 6

3 BACKGROUND AND MOTIVATION 10

4 SYSTEM DESIGN 15

4.1 TRADL System Design 15

4.1.1 TRADL Framework 15

4.1.2 TRADL Main Modules 16

4.2 DQoES System Design 18

4.2.1 DQoES framework 18

4.2.2 DQoES Modules 18

4.2.3 System Optimization Problem 20

5 ALGORITHM DESIGN **22**

5.1 TRADL : Algorithm for Dynamic Resource Allocation 22

5.1.1 Containerized Deep Learning Applications 22

5.1.2 Target-based Resource Allocation 23

5.2 DQoES Solution Design 25

5.2.1 DQoES Performance Management 25

5.2.2 Adaptive Listener on DQoES 27

6 Performance Evaluation **29**

6.1 TRADL Performance Evaluation 29

6.1.1 Implementation and Tested Models 29

6.1.2 Acceptable Only 30

6.1.3 Full two-tier Target 31

6.1.4 Scalability (Mixed) 34

6.2 DQoES Performance Evaluation 36

6.2.1 Experimental Framework and Evaluation Metrics 36

6.2.2 Single Model 37

6.2.3 Multiple Models 43

7 Conclusion **47**

REFERENCES **49**

 doi:10.20944/preprints202011.0017.v1

List of Tables

4.1 Notation Table 21

6.1 TRADL -Tested Deep Learning Models 30

6.2 DQoES -Tested Deep Learning Models 36

 doi:10.20944/preprints202011.0017.v1

List of Figures

3.1 Normalized Value of Loss Function 10

3.2 Cost Time of Batch processing 12

3.3 QoE of Diverse Models 13

3.4 CPU Usage for 5 models 13

4.1 TRADL System Architecture 16

4.2 DQoES System Architecture 19

6.1 5 Jobs (AO v.s NA) 31

6.2 5 Jobs (FT v.s. NA) 32

6.3 CPU Usage of 5-Job NA 33

6.4 CPU Usage of 5-Job FT 33

6.5 10 Jobs (TRADL v.s. NA) 34

6.6 CPU Usage of 10-Job NA 35

6.7 CPU Usage of 10-Job TRADL 35

6.8 Delivered QoE: 10 jobs with the same unachievable objectives (Burst schedule) . . 38

6.9 CPU distribution: 10 jobs with the same unachievable objectives (Burst schedule) . 39

6.10 CPU distribution: 10 jobs with the same achievable objectives (Burst schedule) . . . 40

6.11 Delivered QoE: 10 jobs with the same achievable objectives (Burst schedule) 40

6.12 CPU distribution: 10 jobs with varied objectives (Burst schedule) 41

6.13 Delivered QoE: 10 jobs with varied objectives (Burst schedule) 41

6.14 Delivered QoE: 10 jobs with varied objectives (Fix schedule) 42

6.15 CPU distribution: 10 jobs with varied objectives (Fix schedule) 42

6.16 Delivered QoE: 10 jobs with varied objectives (Random schedule) 43

6.17 CPU distribution: 10 jobs with varied objectives (Random schedule) 44

6.18 Delivered QoE in the cluster with 40 models and varied objective with DQoES . . . 45

6.19 Delivered QoE in the cluster with default resource management algorithms 46

6.20 CPU distribution in the cluster with 40 models and varied objective with DQoES . . 46

6.21 CPU distribution in the cluster with default resource management algorithms 46

 doi:10.20944/preprints202011.0017.v1

Abstract

Yun Song

MS, Fordham University

Differentiate Containers Scheduling for DeepLearning Applications

Dissertation directed by Ying Mao, Ph.D.

The advent of deep learning has completely reshaped our world. Now, our daily life is fulfilled with many well-known applications that adopt deep learning techniques, such as self-driving cars and face recognition. Furthermore, robotics developed more forms of technology which share the same principle with face recognition, such as hand pose recognition and fingerprint recognition. Image recognition technology requires a huge database and various learning algorithms, such as convolutional neural network and recurrent neural network, that requires lots of computational power, such as CPUs and GPUs. Thus, clients could not be satisfied with the computational resource of the local machine. The cloud resource platform emerged at the historic moment. Docker containers play a significant role of microservices-based applications in the next generation. However, it could not guarantee the quality of service. From clients' perspective, they have to balance the budget and quality of experiences (e.g. response time). The budget leans on individual business owners and the required Quality of Experience (QoE) depends on usage scenarios of different applications,for instance, an autonomous vehicle requires real-time response, but, unlocking your smartphone can tolerate delays. Plenty of on-going projects developed user-oriented optimization resource allocation to improve quality of the service. Considering the users' specifications, including accelerating the training process and specifying the quality of experience, this thesis proposes two differentiate containers scheduling for deep learning applications: `TRADL` and `DQoES` .

 doi:10.20944/preprints202011.0017.v1

In TRADL , developers have options to specify a two-tier target. If the accuracy of the model reaches a target, it can be delivered to clients while the training is still going on to continue improving the quality. If the accuracy of the model reaches a target, it can be delivered to clients while the training is still going on to continue improving the quality. The experiments show that TRADL is able to significantly reduce the time cost, as much as 48.2%, for reaching the target.

In addition, DQoES is a differentiating quality of experience scheduler for deep learning applications. DQoES accepts client's specification on targeted QoEs, and dynamically adjust resources to approach their targets. Through extensive, cloud-based experiments, DQoES demonstrates that it is able to schedule multiple concurrent jobs with respect to various QoEs and achieve up to 8x times more satisfied models when compared to the existing system.

 doi:10.20944/preprints202011.0017.v1

INTRODUCTION

In recent years, dramatic growth of big data has been witnessed from different sources, such as webs, cameras, smartphones, and sensors. To utilize the data, many businesses start to powered by big data analytics. For example, Google AdSense [1] recommends clients the advertisements by studying data that they generated through Google applications and Apple FaceID [2] learns from user's face images with TrueDepth camera for a secure authentication solution. This technology is capable of extracting the unique features of the given face and comparing these features with face characteristics from the database, so that it could achieve face recognition. Furthermore, robotics developed more forms of technology which share the same principle with face recognition, such as hand pose recognition and fingerprint recognition. Consequently, various learning algorithms and models are proposed to facilitate the analytical processes. In this domain, deep learning technologies, such as artificial neural networks (e.g. convolutional neural network [3] and recurrent neural network [4]) are popular solutions to enhance the learning and improve the results. For instance, generative adversarial networks [5] are deep neural net architectures that use two neural networks, pitting one against the other in order to generate new, synthetic instances of data that can pass for real data. These applications are all powered by deep learning, the developers have to train a model iteratively with plenty of training data, and apply the model on evaluation dataset to evaluate the quality of model. After making sure the model is satisfied with clients, it can be introduced to the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

public.

From an overall perspective, a typical training process of deep learning can be divided into three sections: model training, model evaluating and parameter tuning. Our dataset could divide into two parts, training dataset and evaluation dataset. In the training process, the algorithm is fed with training data, so that the model can learn as examples and improve its quality iteratively. The evaluation section is followed, it allows the developers to test quality of the trained model through applying the model on the evaluation dataset. Whenever the evaluation is done, if the result does not achieve the expectation, it is most likely for clients to wonder if the model can be further improved with different parameter settings. For the evaluation section, there are plenty of methods that predefined by clients to evaluate the trained model. For example, we usually use accuracy and f1 score as measurement criteria to determine the predictive result. The model with the highest accuracy is what we want to achieve. In this thesis, we use Loss Function to evaluate the model. For instance, Variational Autoencoders(VAE) [6] use reconstruction loss and Convolution Neural Network (CNN) usually use cross entropy. Our measurement standard is the minimum value of Loss Function. To be more specific, in each iteration of training phrase, the model could reach the local minimum of the loss function. Then, as the algorithm keeps running, more iterations are examined for the parameter. Among all iterations, after comparing all values of Loss Function, we are able to find the global minimum of the Loss Function.

However, among so many different deep learning applications, not all of them require perfect accuracy. Users do not necessarily need to wait until each application achieves its maximum value and finishes. For example, Yelp [7] employed deep convolutional neural networks(CNNs), which involved multiple convolutional layers, ReLU(Rectified Linear Units) [8] layers, pooling layers and local response normalization layers to classify uploaded photos. Nonetheless, after capturing photos, the photo classifier could divide the photos into different class without perfect accuracy. What's more, Yelp provided the option for users to report misclassified photos, that means users could tolerate a certain level of error rate.

Obtaining a satisfied deep learning model is both time-consuming and resource-intensive,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

which requires lots of computational power, such as CPUs and GPUs. Thus, the training process on a local machine is usually super slow due to the large amount of training data and the limited resource. Although, there are more resources on the cloud than on a local machine, the Cloud is a multi-tendency system, it is shared by more users, such as storage, mobile and database. Therefore, considering large amount of computing jobs, if we run plenty of deep learning training jobs concurrently, they will still need to compete for the resource. To be more specific, we implemented CNN, Bidrnn [9], VAE, VAET and RNN on the docker platform with 16 CPUs concurrently, then monitored the CPU usage for each job in the meanwhile. The result shows that Bidrnn only utilizes less than 12.5% of CPU. Moreover, the completion time for individual job becomes longer and longer, which significantly slow down the deployment and deployment process.

For companies with limited budgets, it is impractical and expensive to collect the data and train the model all by themselves. Therefore, big players, such as Amazon SageMaker [10], provide pre-trained machine learning services of ready-made intelligence for targeted applications and workflows. For example, if an application wants to automatically identify targeted objects in user-uploaded images, the developer can utilize pre-trained computer vision models to uncover the insights and relationships in the unstructured data, e.g. images. In practice, how good a model performs relays on the learning algorithms as well as training data and how fast the system reacts depends on the deployment plans as well as computational resources. While many research projects focus on improving the accuracy and accelerating the training, deploying the models and manage the resources according to specific requirements is still underinvested. Different applications have varied working environments, which lead to different requirements for the quality of services. For instance, in an autonomous vehicle (e.g. Waymo [11]), the system has to, immediately, react to an object that suddenly appears in the front; however, when unlocking the smartphones,users can tolerate a short delay, up to 1 second [12]. Without any constraints, systems should react users as soon as possible. However, taking budget into consideration, the quality of services could be set to the level, in which user can tolerate and, at the same time, service providers can achieve a break-even state.

Thus, due to insufficient cloud resources which is a multi-tendency system, Docker may not

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

guarantee the quality of the service. Plenty of on-going projects developed user-oriented optimization resource allocation to improve quality of the service. Considering the users' specifications, including accelerating the training process and specifying the quality of experience, this thesis proposes two differentiate containers scheduling for deep learning applications:TRADL and DQoES .

1.1 TRADL

TRADL [13] is a target-based resource allocation scheme to accelerate the training process for deep learning applications in a multi-tenancy system. In TRADL , clients can specify a two-tier target in the format of <Acceptable, Objective> when start training the model. When its loss function reaches the acceptable value, the best parameter set is saved and the model then is ready to use by its users. At this stage, TRADL reduces the resources that are occupied by this specific model so that others have more access to use. When it achieves the objective value, the training processing for the model completes and model is ready to ship to current users. The TRADL system terminates the model, who reaches its objective, and release newly freed resources for other jobs. Both Acceptable and Objective can be omitted. With <_, _>, TRADL will execute the job just like a regular system; While the option <Acceptable, _> means that the model has to obtain its global minimum, but ready to use when achieves Acceptable value. Finally, <_, Objective> represents that this application has a specific objective. The main contributions of this thesis are summarized as follows:

- We introduce a new scheduler, TRADL which utilizes a two-tier target to improve resource allocation of deep learning applications.
- TRADL is designed to reallocate resources when active training models are reaching a predefined Acceptable/ Objective value on the loss function.
- We implement TRADL on a commercial platform, Docker and evaluate it with intensive deep learning applications. The experiments validate that TRADL achieves as much as 48.2%

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

reduction of time cost for reaching the target.

1.2 DQoES

DQoES is a scheduler that supports developer specified Quality of Experience (QoE). When launching a back-end service, a value can be given to DQoES as the system's targeted response delay. DQoES keeps monitoring running models as well as their associated QoE targets and tries to approach the targets through efficient resource management. The main contributions of this paper are summarized as follows.

- We introduce differentiate quality of experience to the deployment of various deep learning applications. With the respect to various budget limitations and usage scenarios, the system reacts differently.
- Without modifying the existing cloud system, we propose DQoES with a suite of algorithms that accepts targeted QoE specifications from clients and dynamically adjust resources to approach their individual targets to achieve satisfied performance at user's end.
- DQoES is implemented into Docker Swarm [14]. Based on extensive cloud-executed experiments, it demonstrates its capability to react to different workload and achieves up to 8x times more satisfied models when compared to the existing system.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

RELATED WORK

In past decades, we have witnessed the blooming development of data-driven applications and their fast-growing infrastructural supports. For example, people from all over the world use mobile social network to connect with each other [15]–[21], where applications collect the data and process the information to enable various services such as personal recommendations and location-based searches. At back-end side, those applications could not be able to host locally due to limited availability of computational resources on the mobile devices.

The cloud resource providers emerged at the historic moment. Lots of projects have been done to improve the performance of the learning models from various perspectives [22]–[24]. In addition, based on ImageNet, a large visual database designed for use in visual object recognition research, many innovative learning algorithms have been proposed as a tournament to improve the accuracy of the classification. Among the solutions, [25], which currently ranks on the top of the list, shows that it is able to use unlabeled images to significantly advance both accuracy and robustness of the learning model. Focusing on the transfer learning, authors [26] revisit classical transfer learning and propose BiT that uses a clean training and fine-tuning setup, with a small number of carefully selected components, to balance complexity and performance. [27] aimed at dealing with the problem of cost-minimization and deadline-constrained workflow scheduling (CMDCWS) model which is proposed

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

by Rodriguez and Buyya in 2014. Because the value of each dimension is an irrelevant feature for the resource attribution, therefore, based on the computational power, such as the cost per unit time, RNPSO reordered and reallocated the resource to make the difference between the well-performed and poorly-performed particles meaningful. Compared to the CMDCWS, RNPSO achieved better performance on test experiments with different scales, including small, middle and large scales.

Because of the large amount of data and limited resource, the process is always super slow. The computer field is full filled with on-going research project which has been tried to reduce the consumed time. Docker containers play a significant role in microservices-based applications in the next generation. Starting a container on a node with locally available images is fast, but due to insufficient resources, it may not guarantee the quality of the service. However, in the case where the selected node does not have the required image, downloading the image from a different registry will increase the configuration time. [28] proposed the system for collaboratively managing Docker images among a group of nodes. This system focus on storing or deleting images partially. What is notable is that it allows distributed pull of images when starting the container. Compared to the state-of-the-art full-image-based approach, CoMICon can reduce application delivery time by an average of 28% when applied the model on 142 publicly available different images from the Docker hub. In addition, [29] is proposed to quantizes gradients to ternary levels to reduce the overhead of gradient synchronization, which results in overall communication reduction. Focusing on mobile devices, [30] is proposed to accelerate the execution deep neural network in a resource constraint environment. Moreover, [31] is developed to measure the vulnerability of deep learning models as well as to conduct comparative studies on attacks/defenses in a comprehensive and informative manner. Considering a cloud system, [32] is designed to schedule the resources with the respect to growth efficiency of the loss function. On the other hand, [33] and [34] optimize the system by efficient container placement schemes. Focusing on accelerating neural network training, FreezeOut [35] proposes training only part of hidden layers instead of all layers in traditional computation, freezing them out one-by-one and excluding them from the backward pass. TicTac [36] studies the communication scheduling between computing nodes in distributed deep learning systems. With parameter servers (defined in the paper), TicTac improves the iteration time by up to

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

37.7% in inference and 19.2% in training of state-of-the-art deep neural networks. Besides human's efforts, DeepRM [37] shows that machine learning itself can provide a viable alternative to human-generated heuristics for resource management.

As an application, various learning techniques have been applied, from clients' points of view, to assure the quality of experience at end users. Authors in [38] present the first attempt to predict video quality of experience based on information directly extracted from the network packets, which can detect anomalies at the current time instant and predict them at the next immediate instant. Deep learning is a subfield of machine learning that concerned with algorithms inspired by the structure and function of the brain called artificial neural networks [39]. In the training process, neural networks learn a mapping from inputs to outputs that can be summarized as solving the problem of function approximation. As shown in [40], discovering the best approximation of a neural network is time-consuming and varies significantly based on the type of hardware, e.g. CPU and GPU, which are used for computation. The cloud service providers, such as Amazon AWS [41] and Microsoft Azure [42], offer the infrastructure support for training the deep learning models. From the service provider's point of view, the cloud computing platform is a multi-tenancy system, where limited resources are sharing among the users. Therefore, to boost the performance, efficient scheduling of the training jobs is required. A quality-driven cluster scheduler named SLAQ [43] is designed for approximate large-scale Machine Learning(ML) training jobs. Base on the iterative nature of algorithms, SLAQ collects quality and resource usage information from ML jobs, and automatically infer the models' loss reduction rate from past iterations to predict future resource consumption and loss improvements for subsequent allocation decisions.

Utilizing a novel deep neural network, [44] is able to learn the relationships between the network parameters and the subjective scores that indicate quality of experience when a user is viewing videos transmitted over the mobile internet in a practical environment. Additionally, CVART [45] studies application response times by using image recognition to capture visible changes in the display of the device running the application to compute the application response time of an operation that triggers these visual changes. However, CVART is a toolkit and fails to improve response time.

 doi:10.20944/preprints202011.0017.v1

Although, there are more resources on the cloud than on a local machine, the Cloud is a multi-tendency system, it is shared by more users for the resources, such as storage, mobile and database. Thus, many experts developed different algorithm to reallocate resource, such as CPUs, to accelerate the whole process and save computational cost [24], [46]. For example, Novel[47] optimization model could allocate cloud computing resources for a cloud-based information and communication technology flexibly. With the modified priority list algorithm, Novel cost-oriented optimization model saved the operation cost of cloud platform in demand side management.[48] resource allocation was applied on Infrastructure-as-a-Service (IaaS) providers like Amazon EC2 because of dynamically fluctuating resource demands. To maximize total revenue and minimize the energy cost, the work modeled the problem with Model Predictive Control(MPC). The experimental result reveals that compared to the static allocation strategies, the model achieved higher net income and minimized the average request waiting time. The work [49] roposed a task-oriented model for resource allocation in cloud computing environment which focused on improving the ranking system that calculates the weights of different tasks by the Analytic Hierarchy Process (AHP) and the pairwise comparison matrix. The cloud computing resources can be assigned for the tasks based on the rank determined by reciprocal pairwise comparison matrix, however, the reciprocal pairwise comparison matrix might have inconsistent elements since different tasks could have conflicting preferences. Therefore, this work proposed the induced bias matrix to identify the inconsistent elements in order to reduce the consistency issue of a reciprocal comparison matrix. Hybrid [50] optimal cloud computing resource scheduling contributed to accelerate the scheduling speed and improve the efficiency of scheduling when applied traditional heuristic algorithm under the abnormal network. Genetic algorithm is good at improving the search efficiency. And as for colony algorithm, it plays an important role in increasing the accuracy of the optimal solution. Hybrid optimal cloud computing resource scheduling merged the advantages of genetic algorithm and ant colony algorithm. As the results, it only takes 10s for Hybrid optimal cloud computing resource scheduling to allocate resource, which is much faster than single genetic algorithm or colony algorithm.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

BACKGROUND AND MOTIVATION

Generally, the process for training a deep learning model is the parameter tuning process to find the global minimum of the predefined loss function.

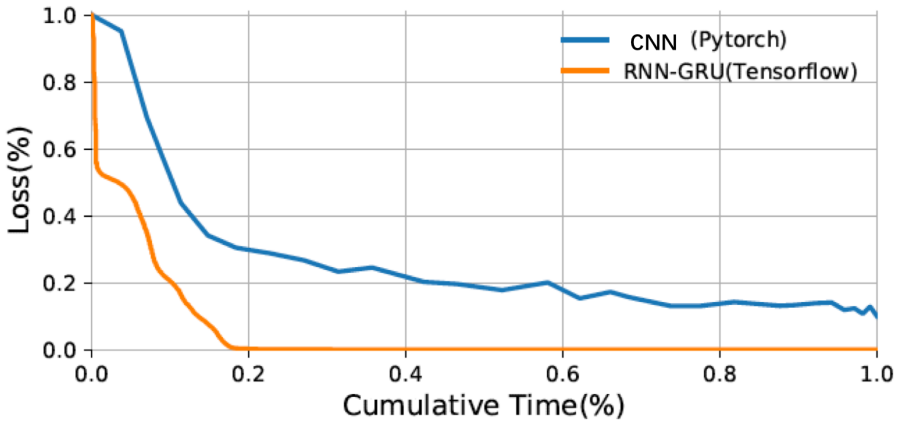


Fig. 3.1: Normalized Value of Loss Function

Fig. 3.1 shows two training process of a Recurrent Neural Network (RNN-GRU) and convolutional neural network (CNN-MNIST). The figure plots the cumulative time v.s. the loss reduction. As we can see that RNN-GRU achieves it's the largest reduction at 20% of the time, which means that clients need to wait the rest 80% time until they can start using the model. A similar trend can be found for the CNN-MNIST training processing. When it completes 60% of the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

time, CNN-MNIST records loss reduction 0.18 and it further reduces to 0.16 when completes. This means that with the last 40% of the total completion time, accuracy improves only 2%. With the assumption that a user can tolerate error rate at a certain level, the learning model does not have to be perfect before shipping to the users.

Suppose many other learning tasks are running in parallel on the same computing system and seeking computational resources. When RNN-GRU achieves an acceptable value of the loss reduction, it is reasonable to shift parts of the computational resource occupied by RNN-GRU training to the other learning tasks (i.e., real-time task-resource scheduling), so as to improve the overall performance of all tasks.

Also, there's a trade-off between the client and developer. From the view of clients, their target is to achieve higher accuracy of training model. However, during deep learning training process, higher accuracy means more iterations, which is both time-consuming and resource-consuming. That contradicts to the desire of developers. They always want to train models at low cost. We proposed the assumption that users can tolerate a certain level of error rate. Therefore, based on this assumption, we develop two-tier target to balance the relationship between the clients and developers.

Thus, Considering the users' specification, including accelerating the training process, this thesis proposes differentiate containers scheduling for deep learning applications:`TRADL` .

In `TRADL` , we set two values: acceptable and objective. Acceptable stands for the maximum error rate of loss which client can accept. Before training job, developers can do a survey on clients and decide which error rate of loss value they can accept. And considering the cost, Objective means the value of loss which developer satisfy. In our system, when training job reaches Acceptable, we allocate less resource and shift resource to other learning tasks. And when it reaches Objective, we terminate the task and release resource.

In the meantime, the clients not only want to accelerate the training process, but also desire to specify the quality of experience.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

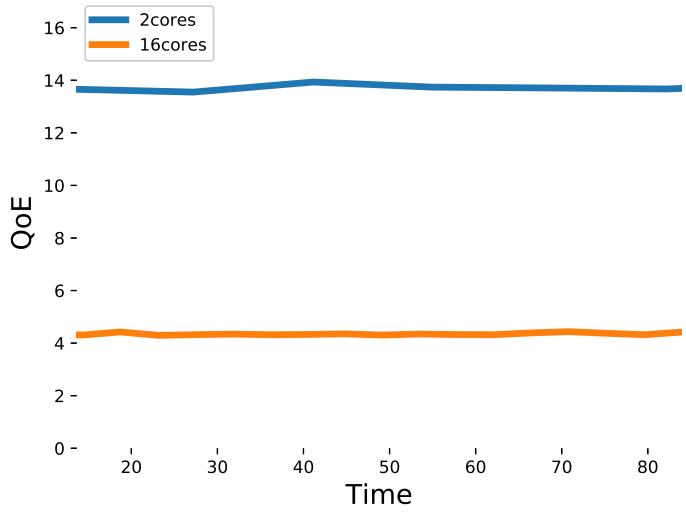


Fig. 3.2: Cost Time of Batch processing

The quality of experience we defined is the response time for using deep learning applications. Fig. 3.2 shows the quality of experience when running Resnet50 on different machines. Computer with 16 cores CPUs could provide more resource, so that the response time is much shorter, it achieved 4 seconds. On the contrary, the QoE for machine with 2 cores CPUs is more than triple performance for computer with 16 cores CPUs. In Fig. 3.3, we can see that if we run five Resnet50 jobs concurrently, the performance is as high as 17.5. This is because each training job only utilizes a part of CPUs (shown in Fig. 3.4). To be more specific, Model -1 is assigned almost 20% of CPUs. On the opposite, the Resnet50 job utilizes nearly 100% CPUs when we run only one job on the machine. As a consequence, the quality of experience is reduced to 5 seconds. It is a fact that the training job could show different performance according to the diversity of distributed resource, which provides the high possibility that we can achieve the desired cost time of each epoch through reallocating CPU on the machine.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

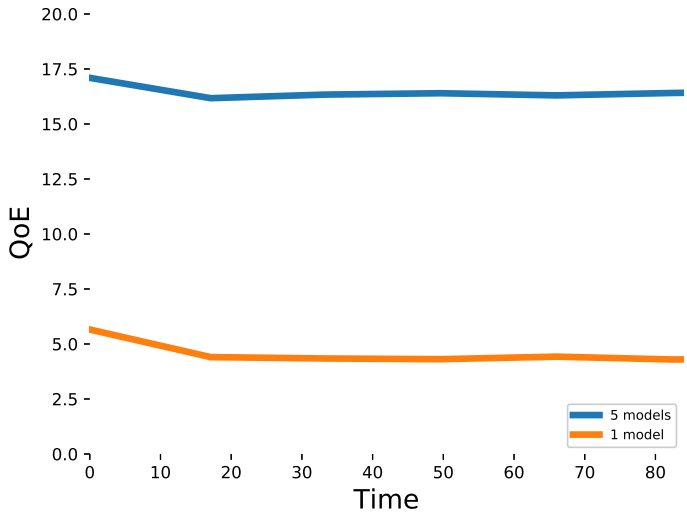


Fig. 3.3: QoE of Diverse Models

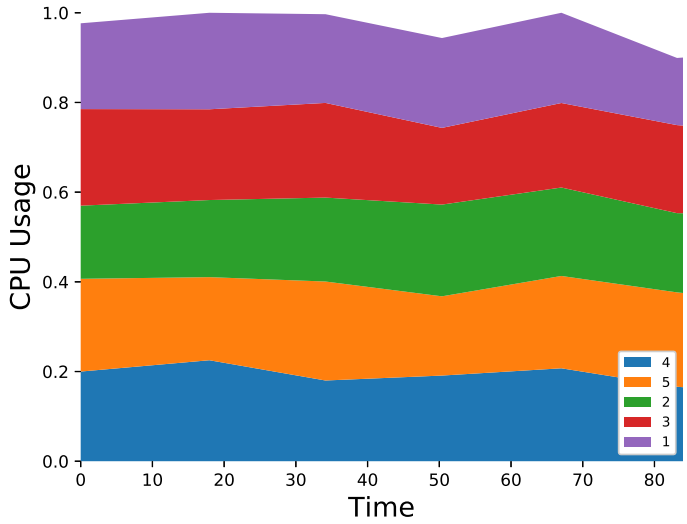


Fig. 3.4: CPU Usage for 5 models

 doi:10.20944/preprints202011.0017.v1

Plenty of users not only desire to accelerate the training process, but also want to specifying quality of experience. Consequently, this thesis proposes another differentiate containers scheduling for deep learning applications:DQoES .

DQoES supports developer specified Quality of Experience (QoE). When launching a back-end service, a value can be given to DQoES as the system's targeted response delay. DQoES keeps monitoring running models as well as their associated QoE targets and tries to approach the targets through efficient resource management.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

SYSTEM DESIGN

Considering the users' specification, including accelerating the training process and specifying the quality of experience, this thesis proposes two differentiate containers scheduling for deep learning applications:TRADL and DQoES . This chapter shows the system design for them

4.1 TRADL System Design

In this section, we present the system design of TRADL in detail, including its design logics and functionalities of key modules.

4.1.1 TRADL Framework

In a typical multi-tenancy system, a cluster of cloud servers, is configured to a manager-worker mode and works in a distributed manner; Managers interact with clients and assign tasks to the workers, and workers execute the tasks and generate the results. In this system, managers maintain the status of workers and make sure jobs are efficiently executed.

The system scheduler, who controls the resource allocation, usually resides on the manager

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

side. They have a global view of the nodes inside the cluster. Workers are the physical machines that provide the resources, such as CPU, GPU and memory. In general, the number of workers is way greater than the number of managers. To prevent managers from overwhelm the system, TRADL adopts a resource allocation scheme on the worker side.

In TRADL , the system scheduler of the cluster remains untouched, instead, an allocator is implemented as a plugin, on the worker, to dynamically update the configuration for each job.

4.1.2 TRADL Main Modules

As demonstrated in Fig. 4.2, TRADL consists of three modules: a Request Analyst, a Training Tracker and a TRADL Allocator. The Request Analyst is working on the manager and the other two modules are running on the worker nodes. Each module runs independently and exchanges information about active jobs in the pool. Their functionality is detailed below.

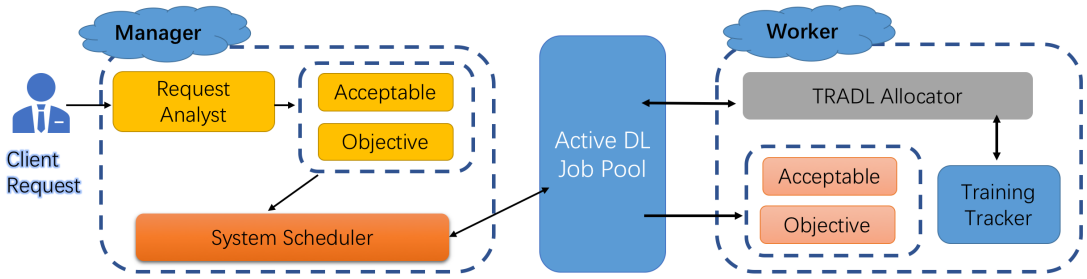


Fig. 4.1: TRADL System Architecture

Request Analyst

In TRADL system, users have options to initiate deep learning jobs with a two-tiered target, in the format of <val_1, val_2>, where val_1 represents a minimum value of the loss function that the model considers to be acceptable and val_2 is the accuracy at which the model obtained satisfies the users. For example, the command, Face_Recognition, <0.93, 0.98>, starts the training of a face recognition model with a 93% acceptable accuracy threshold and a 98% target

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

accuracy threshold. To set meaningful values in the two-tiered target, one must be familiar with the loss function, which varies for different models, e.g. reconstruction loss for Variational Autoencoders, across entropy for MNIST, and inception score for Generative Adversarial Networks.

Request Analyst collects the two-tiered target from the clients, extracts `Flag_A` (acceptable value) / `Flage_O` (Objective value) and passes the information to the system scheduler, which then labels the training job with `Flage_A` and `Flag_O`.

Training Tracker

As described above, training applications are tagged with a two-tiered target and execute on worker nodes. A Training Tracker, first, extracts the corresponding Acceptable and Objective values from its local active job pool, and then, creates a table to map jobs, J_{id}, with their targets. With the table, Training Tracker runs periodically to monitor the progress of the jobs. If any of them reaches the Acceptable or Objective values, the `TRADL` allocator will be triggered for further processing. Training Tracker has to iterate all running jobs on this worker and will introduce overheads to the system. To reduce the overhead, the frequency of the tracking is designed as a configurable parameter for the system.

`TRADL` Allocator

The `TRADL` allocator serves as the key module of the system. It obtains the progress data and resource usages of each running job. With the information, it calculates the required parameters for analysis and executes the algorithm (described in Section 5.1) to update the resource configuration for each of the jobs.

`TRADL` Allocator runs purely on the worker side to reduce the workload of managers and does not exchange any information outside this worker, which further reduces the network traffic and avoids communication delays. In addition, it is triggered by the Training Tracker and runs on

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

demands.

4.2 DQoES System Design

This section presents the system architecture of DQoES in detail, including its framework, design logics and functionalities of key modules.

4.2.1 DQoES framework

A typical management system for a cluster of containers, such as Docker Swarm [51] and CNCF Kubernetes [52], involves managers and workers. It works in a distributed manner, in which managers, on the one hand, interact with the users, analyze their requests as well as manage the workers that associated with this cluster and workers, on the other hand, are the workhorses that provide the computing resources, such as CPU, GPU and memory, to execute the tasks, store the data as well as report their status to managers.

DQoES employs the manager-worker architecture. In DQoES, the manager collects QoE targets from clients, pass them the workers and maintains an overall performance assessment table to keep monitoring the real-time performance of all the active workloads in the system. The workers, who host the deep learning applications, track the performance of each individual model and its real-time resource usages. According to the QoE targets, the worker dynamically adjust resource limits for each container to achieve best overall performance among all models that resident in it.

4.2.2 DQoES Modules

As demonstrated in Figure 4.2, DQoES consists of four major modules, an Application Monitor, a Worker Monitor and an Executor on the worker side. A QoE Analyst on the manager side. Each module runs independently and exchanges information about models inside the containers as well as

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

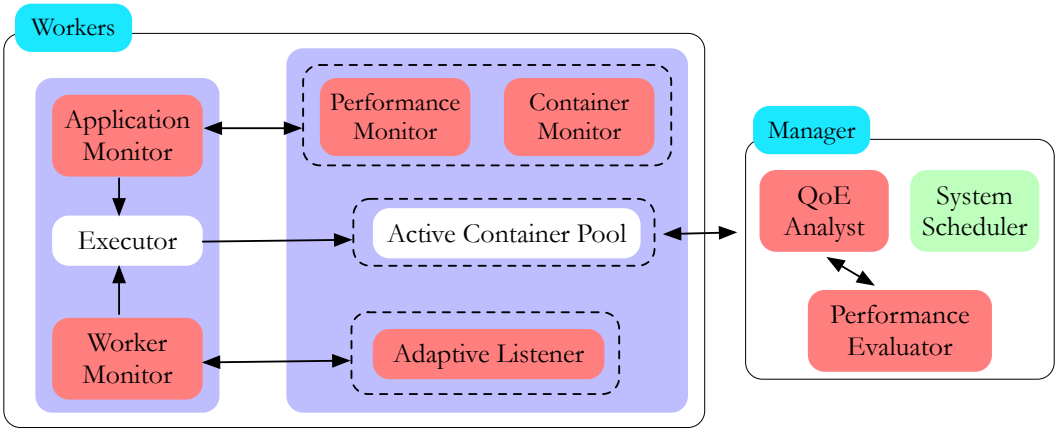


Fig. 4.2: DQoES System Architecture

the worker status. Their functionality is detailed below.

Application Monitor: it maintains performance metrics for each deep learning model runs this worker. For example, when users queries the model, the response time would be recorded along with the resource usage that associated with the time cost. Comparing with the QoE targets, The DQoES executes an algorithm to adjust resource assignments for active containers and approaches the targeted values iteratively.

Worker Monitor: it measures the active container pool. When a new model is assigned by a manager, the worker monitor adds it to the pool and keep tracking performance difference in each iteration by using an adaptive listener. The listener hosts an algorithm that regulates the frequency of updating the resource assignment.

Executor: it is a key module that, accepts the workload and collects data on the worker. Based on data, it calculates the parameters that required by the algorithms (described in Section 5.2) in Application and Worker Monitor. Upon receiving a new plan for resource configuration for each container, the Executor will interrupt the current limits and update containers with newly calculated values.

QoE Analyst: it resides on the manager side. A QoE analyst interacts with clients and collects the QoE targets for each of the deep learning application. The targeted values are sent to workers

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

through the System Scheduler. When the status reports arrive from workers, it utilizes a Performance Evaluator to preserve data and monitor the overall system performance.

4.2.3 System Optimization Problem

In `DQoES` , we consider deep learning applications host on the cloud with a cluster of containers and each learning model resides in a container. `DQoES` aims to provide users the service with a predefined QoE target.

In the system, we denote $c_i \in C$ to be a container that serves one deep learning application. For each c_i, we have $r_i \in R$ is the resource usage of c_i that can be monitored through the system APIs, such as `docker stats`, where function $R(c_i, t) = r_i$ is defined to retrieve the resource usages in real-time.

For the model that runs in each container, we keep tracking its performance $P(c_i, t) = p_i$, which is a value to represent a predefined quality metric, e.g. response time, of a particular service. Additionally, there is an pre-defined QoE targeted value for each model, $o_i \in O$. Then, the quality of a given container c_i (a learning model runs inside) is $q_i = o_i - p_i$. Therefore, the system performance on a specific worker node W_i is,

$$Q_{W_i} = \sum_{c_i \in W_i} q_i \tag{4.1}$$

`DQoES` anticipates a system that minimizes the performance difference between the predefined QoE target and current container outputs. Considering the real-time container outputs, `DQoES` classifies running models into three different categories.

- Class G: The models in this category perform better than the preset QoE value, e.g. response faster than the target.
- Class S: The models in this category achieve the predefined QoE targets and they are satisfied containers.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

- Class B: The model in this category is underperformed due to lacking of resources or unrealistic targets.

Assume that there are n containers, each runs one learning model in the system. Then, our performance optimization problem, $\mathcal{P}$, can be formalized as

$$\mathcal{P}: \text{Min} \sum_{c_i \in B \ OR \ c_i \in G}^{n} q_i \tag{4.2}$$
$$\textbf{s.t.} \sum_{c_i \in W_i}^{n} r_i T_R$$

The following table summarizes the parameters that utilize in DQoES.

Table 4.1: Notation Table

W_i	Worker i
$c_i \in C$	The container i in the set C
G	Set G contains the c_i with a better performance than o_i
S	Set S contains the c_i with targeted performance o_i
B	Set B contains the c_i with a worse performance than o_i
$o_i \in O$	Objective value o_i for c_i in the set O
$r_i \in R$	Resource usage r_i for c_i in the set R
$R(c_i, t)$	The function that uses to calculate r_i at time t
p_i	The performance indicator of c_i
$P(c_i, t)$	The performance function that calculate p_i at time t
q_i	The quality of c_i that can be calculated by using o_i and p_i
T_R	The total resource usage of containers on a worker i
R_G, R_S, R_B	The resource usage of containers in set G, S and B
Q_G, Q_S, Q_B	The sum of qualities for containers in set G, S and B
$L(c_i, t)$	The limit of the resource usage for container c_i at time t

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

ALGORITHM DESIGN

Considering the users' specification, including accelerating the training process and specifying the quality of experience, this thesis proposes two differentiate containers scheduling for deep learning applications:TRADL and DQoES . This chapter shows the algorithm design for them.

5.1 TRADL : Algorithm for Dynamic Resource Allocation

In this section, we present the design of the algorithm for dynamic resource allocation in TRADL , which can adjust resource assignments for active jobs at run time.

5.1.1 Containerized Deep Learning Applications

Running a deep learning application usually involves multiple different dependencies, such as Tensorflow [53] and Pytorch [54], which makes resource configuration at run time a challenging task.

Given the system requirements, we utilize the Docker containers [51], where all dependencies of a deep learning application is enclosed in a virtual container, which is an emerging lightweight

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

virtualization technology. From the operating system's point of view, containers are no different to regular processes. With the Docker platform, we adopt the existing toolkit for resource management of containerized applications. For example, users can initiate a container without specifying a resource limit by using the command, `docker run image_name`, and let it compete for resources with other processes freely in the system. When a container is running, the command, `docker update --cpu-share 512 -m 300M container_id`, can set an upper limit of a specific container on CPU and memory usage.

5.1.2 Target-based Resource Allocation

In a cluster of containers, the manager accepts the commands from users and selects a worker to host the containers. Containers, then, would compete for resources such as memory and CPU when they are running in the same worker.

By default, each container is assigned with the same priority resulting in uniform resource distribution and maintaining fairness among all containers in the worker. For example, when there are n containerized applications running with the default setting, theoretically, each of them will be assigned $1/n$ of the total available resources (e.g. the number of CPU cores). In reality, however, not all applications can reach the upper bound. The real resource usage depends not only on the contention in the system, but also the program itself and other factors, which are out of the control of the system.

`TRADL` utilizes both theoretical fairness allocation and real resource consumption by dynamically updating configuration with respect to the two-tier target, Acceptable and Objective.

As shown in Line 1 of Algorithm 2, the worker node (W_i) maintains its local active containers c_i. It needs to obtain the Acceptable and Objective values for each of them (c_i^A and c_i^O). Additionally, worker initializes the following parameters,

- AS: Acceptable Set that includes all containers, who have reached acceptable states.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

Algorithm 1 Target based Resource Allocation on Worker W_i

1: Initialization: W_i, $c_i \in \{c_1, c_2, ..., c_n\}$, Acceptable value c_i^A, Objective value c_i^O, Set AS, Resource Usage Function $RU(c_i)$, Loss Function $LF(c_i)$,

2: $n = |W_i|$
3: **if** $\sum_{i=1}^{n} RU(c_i) = 1 = \alpha \times$ Total_Resource **then**
4: **for** $c_i \in W_i$ **do**
5: c_i.remove_limits

6: **else if** $|AS| = n$ **then**
7: **for** $c_i \in W_i$ **do**
8: c_i.remove_limits

9: **else**
10: **for** $c_i \in W_i$ **do**
11: **if** $c_i^O \leq LF(c_i)$ **then**
12: c_i.terminate
13: **else if** $c_i^O > LF(c_i)$ & $c_i^A \leq LF(c_i)$ **then**
14: AS.insert(c_i)
15: **if** $RU(c_i) > \frac{1}{n}$ **then**
16: c_i.set_limits($\frac{1}{n \times 2}$)
17: **else if** $RU(c_i) \leq \frac{1}{n}$ **then**
18: c_i.set_limits($\frac{1}{n+1}$)

- $RU(c_i)$: Resource Usage function that takes the container id as the input and output its current resource consumption.
- $LF(c_i)$: Loss Function that defined by the clients, who submits the training job. It calculates the current value of Loss Function for each particular container.

After initialization, the algorithm calculates the number of active containers, and then, compares the sum of resource usages to the maximum available resources in the system times a factor α, where α represents the system overhead (resources that consumed by other processes). If the sum is less than $\alpha \times$ Total_Resource, it means that containers of training jobs fail to fully utilize the resources. Therefore, we can safely remove the limits from all of them (Line 2 - 5). When all containers are in the AS, TRADL removes all the limits to let them compete freely and maintain fairness in the system (Line 6 - 8). Whenever a container reaches its preset objective, we terminate it and release the resource for others (Line 9 - 12). In a scenario that the training job achieves acceptable value but not

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

objective, TRADL first inserts it into the AS set and then, limits the resources occupied by this job (Line 13 - 14). Based on its current resource consumption ($RU(c_i)$), there are two values of limits,

- When the portion is taken by the container that is larger than fair distribution, TRADL significantly reduce the usages to $\frac{1}{n\times 2}$ and move resources to others (Line 15 - 16).
- When it is less than the fair distribution, TRADL limits it to $\frac{1}{n+1}$ (Lin 17 - 18).

5.2 DQoES Solution Design

In this section, we present the detailed design of the performance management algorithms in DQoES , which can adjust resource assignment for containers at run time. In addition, we discuss the algorithm of an adaptive listener to reduce the overhead of DQoES .

5.2.1 DQoES Performance Management

DQoES manages the QoE of learning models through updating the resource distribution for their host containers. The administrators of a container management system, such as Docker Swarm, are able to configure a "soft limit" for a running container. The limit specifies an upper bound of the resources that a container can be assigned. For example, the command `docker update --cpus="1.5" Container-ID` can set the container is guaranteed at most one and a half of the CPUs assuming the host has at least 2 CPUs.

DQoES utilizes Algorithm 2 to manage the performance, in terms of QoE targets, of each deep learning application. Firstly, it initializes the parameters that required by the algorithm (Line 1). For every active container, it fetches the runningtime resource usage, r_i, the current performance value, p_i and, together with the predefined target, it calculates the quality of the container, q_i, at this moment (Line 2-5).

DQoES , then, classifies all learning models into three different categories, G, B and S. Based

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

Algorithm 2 Performance Management on W_i

1: Initialization: W_i, $c_i \in C$, $o_i \in O$, $p_i \in P$,

2: **for** $c_i \in W_i$ **do**
3: $R(c_i, t) = r_i$
4: $P(c_i, t) = p_i$
5: $q_i = o_i - p_i$
6: **if** $q_i > \alpha \times o_i$ **then**
7: G.insert(c_i)
8: $Q_G = q_i + Q_G$
9: $R_G = r_i + R_G$
10: **else if** $q_i < -\alpha \times o_i$ **then**
11: B.insert(c_i)
12: $Q_B = q_i + Q_B$
13: $R_B = r_i + R_B$
14: **else**
15: S.insert(c_i)

16: **for** $c_i \in W_i$ **do**
17: **if** $c_i \in G$ **then**
18: $L(c_i, t+1) = L(c_i, t) * (1 - \frac{q_i}{Q_G} \times R_G \times \beta)$
19: **if** $L(c_i, t+1) < \frac{1}{2 \times |C|}$ **then**
20: $L(c_i, t+1) = \frac{1}{2 \times |C|}$
21: **else if** $c_i \in B$ **then**
22: $L(c_i, t+1) = L(c_i, t) * (1 + \frac{q_i}{Q_B} \times R_G \times \beta)$
23: **if** $L(c_i, t+1) > T_R$ **then**
24: $L(c_i, t+1) = T_R$

on the current quality of c_i, if it is larger than $\alpha \times o_i$, which means the quality level is higher than the target, c_i is marked as G. The α is a percentage value that used to represent developer's tolerance of the target; on the other hand, if q_i is smaller than $-\alpha \times o_i$ that indicates the quality level is lower than the target, c_i is set to be B; finally, in the case that q_i falls within the interval $[-\alpha \times o_i, \alpha \times o_i]$, it suggests that the service provided by c_i satisfies the predefined QoE target and c_i joins S. When classifying the containers, `DQoES` calculates a total quality value of learning models in G and B as well as the sum of resource usages, respectively (Line 6-15).

For the containers in G, which perform better than their QoE targets, Algorithm 2 cuts their resource usages to reduce the performance and approach their targets. For underperformed containers

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

in B, DQoES tries to increase their resource allocation to help them move toward the QoE targets. Specifically, the following two branches are executed.

- For $c_i \in G$, the total resource limits are reduce from the previous values by $R_G \times \beta$, where β is a parameter that system administrator can configure to gradually update the containers. The degree of the reduction depends on the how far away a c_i is from its QoE target. For example, when a learning model is performing slightly better than the targeted value, the value of $\frac{q_i}{Q_G}$ is very small and the resource reduction is limited (Line 16-18). When updating the limits, DQoES set $\frac{1}{2\times|C|}$ to be the lower bound of the model to prevent abnormal behaviors due to lacking of resources (Line 19-20).
- For containers in B, the saved resources from G will be reallocated to them. Similar to G, the degree of increase depends on how bad a container performs when comparing to its QoE target. For example, if a given container, c_i, is underperforming a lot, which results in a large value of $\frac{q_i}{Q_B}$ and leads to a hike on the resource limit (Line 21-22). However, there is an upper limit (e.g. hardware limits) of the resource allocation, T_R, to each of the container (Line 23-24).

5.2.2 Adaptive Listener on DQoES

The Algorithm 2 aims to dynamically update the resource allocation for active containers in order to make the learning models approach their predefined QoE targets iteratively. To achieve the goal, DQoES has to keep measuring performance of each container and their resource usages, which create overhead to the system. In addition, with a fixed number of active containers and untouched QoE targets, DQoES , after several iterations, converges to the state with a stable resource distribution. Therefore, it is unnecessary to frequently collect information and execute Algorithm 2.

In DQoES , it implements an adaptive listener with an an exponential back-off scheme to control the frequency. Algorithm 3 presents the details in the listener. At the very beginning, it initializes the parameters, such as the sets G, S, B, that require for execution (Line 1). Then, the sum of qualities, Q_G, Q_B, Q_S for each of the container set is calculated. Please note that, for $c_i \in S$, the learning

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

Algorithm 3 Adaptive Listener on W_i

1: Initialization: W_i, $c_i \in W_i$, $o_i \in O$, $p_i \in P, q_i, G, S, B$

2: **for** $c_i \in W_i$ **do**
3: **if** $c_i \in G$ **then**
4: $Q_G = Q_G + q_i$
5: $Q_G(t) = Q_G$
6: **else if** $c_i \in B$ **then**
7: $Q_B = Q_B + q_i$
8: $Q_B(t) = Q_B$
9: **else if** $c_i \in S$ **then**
10: $Q_S = |S|$
11: $Q_S(t) = Q_S$

12: **if** $|Q_G(t+1) < Q_G(t)|$ & $|Q_B(t+1) > Q_B(t)|$ **then**
13: $i++$
14: **if** $i2$ **then**
15: $IV = IV \times 2$
16: $i = 0$
17: **else if** $Q_S(t+1) < Q_S(t)$ **then**
18: $IV = IV \div 2$
19: $i = 0$
20: Execute Algorithm 1
21: **else**
22: $IV = IV$
23: $i = 0$

models have satisfied their QoE targets. Thus, their $q_i = 0$ and we use the number of containers in S to represent Q_S. While Q_G, Q_B, Q_S are per iteration values, DQoES uses $Q_G(t), Q_B(t), Q_S(t)$ to record the time serial of these values (Line 2-11).

Then, DQoES compares the most recent two values of Q_G and Q_B. If both Q_G and Q_B are approaching to 0, which indicates by $Q_G(t+1) < Q_G(t)$ and $Q_B(t+1) > Q_B(t)$, it means they are all converging to S. When the trend maintains for 3 consecutive iterations, Algorithm 3 doubles the interval, which reduce the frequency of updating the resource allocation (Line 12-16). However, when $Q_S(t)$ reduces, it suggests the stable state is broken. It could be abnormal usages of one particular container or a new one joins the system. In this case, DQoES halves the interval and execute Algorithm 2 immediately (Line 17-20). When the system performance is still bouncing, DQoES maintains the original interval (Line 21-23).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

Performance Evaluation

Considering the users' specification, including accelerating the training process and specifying the quality of experience, this thesis proposes two differentiate containers scheduling for deep learning applications:TRADL and DQoES . This chapter shows the performance evaluation for them.

6.1 TRADL Performance Evaluation

In this section, we present the evaluation performance of TRADL. The evaluation result proves that TRADL accelerates the deep learning training process based on the two-tier target.

6.1.1 Implementation and Tested Models

Our experiments are based on Docker Community Edition (CE) 17.09.0-ce, and built on the Cloudlab [55]. The tested is build on the R320 physical nodes, which belongs to the Cloudlab Apt cluster, housed in the University of Utah's Downtown Data Center in Salt Lake City, Utah. The node configures 16 CPUs (Intel(R) Xeon(R) CPU E5-2450 0 @ 2.10GHz), 16GB Memory with Ubuntu 16.04.3 LTS.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

We evaluate TRADL system with various deep learning models. The models use both the Tensorflow [53] and the Pytorch [54] platforms. Table 6.2 lists testing models.

Table 6.1: TRADL -Tested Deep Learning Models

Model	Loss Function	Two-tier Targ.
Variational Autoencoders (VAE)	Recon. Loss	120, 110
Modified-NIST (MNIST)	Cross Entropy	0.3, 0.25
Gated Recurrent Unit (GRU)	Quadratic Loss	0.01, 0.008
Long Short-Term Memory (LSTM-CFC)	Softmax	0.02, 0.015
Long Short-Term Memory (LSTM-CRF)	Squared Loss	10.5, 10.2
Bidirectional-RNN	Softmax	0.6, 0.5
Recurrent Network (RNN)	Softmax	0.55, 0.5
Dynamic RNN	Softmax	0.15, 0.1

Experiment Setup and Evaluation Metrics

To assess TRADL, we mainly focus on the following evaluation metrics:

- Acceptable and Objective time: the time cost that each job achieve the two-tier target.
- CPU Usage: the CPU usage for the deep learning applications

As described before, one of the values can be empty in the two-tier target. We evaluate the performance of TRADL system with three different scenarios:

- Acceptable Only: <value_1, _> (denoted as AO)
- Full two-tier Target: <value_1, value_2> (denoted as FT)
- No Algorithm: <_, _> (denoted as NA)

6.1.2 Acceptable Only

With the respect to other none TRADL tasks in the operating system, we set α = 0.9 (α as denoted in Algorithm 2).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

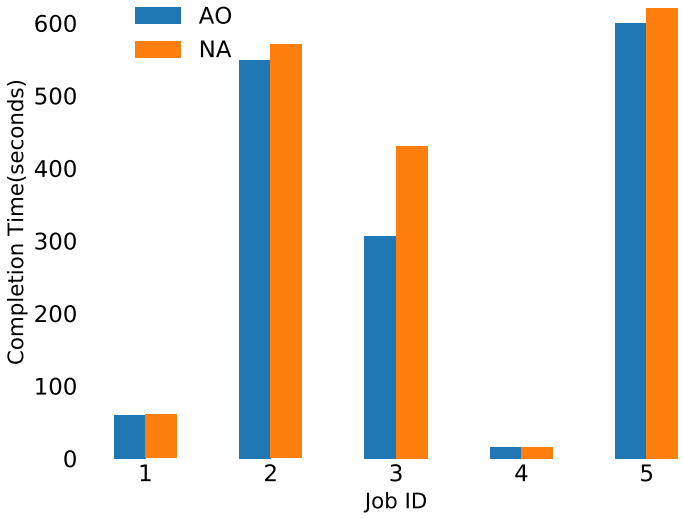


Fig. 6.1: 5 Jobs (AO v.s NA)

Fig. 6.1 plots the time cost for achieving the acceptable states when 5 jobs submit to the system randomly with 0-200s. Please note that we read the log to record the time for NA. As we can see that Job-4 reaches the preset value almost at the same time (15.31s v.s. 16.32s) for AO and NA. This is due to the fact that, at the very beginning, `TRADL` does not do anything other than monitoring active jobs periodically. When the first job (Job-4), who reaches the acceptable state, `TRADL` starts working to update the resource allocation for it. In this experiment, the time cost for Job-1, which is the second one that reaches acceptable state, is also very similar for AO (59.69s) and NA (60.61s) because of the remaining time after Job-4 becomes acceptable is very limited. For the other longer jobs, however, `TRADL` is able to improve the time cost by 29.1% (Job-3), 5.2% (Job-2) and 4.1% (Job-5). The degree of improvement varies is because the jobs are submitted randomly from 0 - 200s.

6.1.3 Full two-tier Target

This experiment uses the same jobs and starting time as the previous one. However, each job is equipped with a full two-tier target, in the format of `<value_1, value_2>`.

We can find the same trend on the Fig. 6.2, where Job-1 and Job-4 achieve the acceptable state at a similar time cost. However, the other jobs receive a significant reduction on the time cost.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

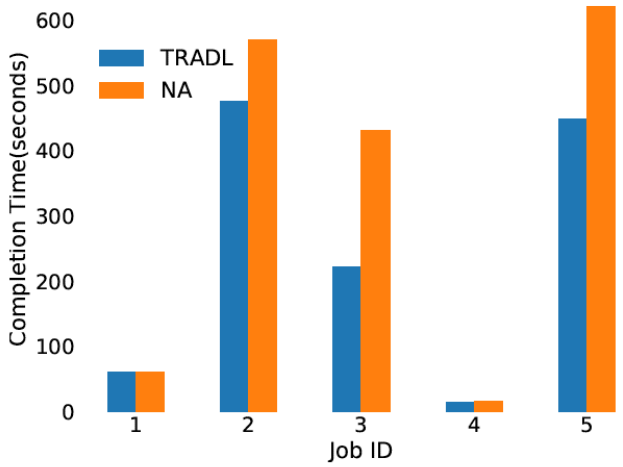


Fig. 6.2: 5 Jobs (FT v.s. NA)

- Job-3 reduces 48.2%: 428.35s to 221.94s
- Job-2 reduces 18.0%: 578.92s to 474.85s
- Job-5 reduces 17.4%: 541.13s to 447.93s

The reason for the dramatic improvement is reducing the resources being held by the DL application when it reaches its acceptable target so that other applications can use more resources. Upon reaching the predefined objective, the clients assume that the model is ready to use so that `TRADL` terminates the container and releases the resource for others.

Fig. 6.3 and Fig. 6.4 illustrate the detailed CPU usages of NA and FT. We can clearly find on the Fig. 6.3 that the resource allocation is under a fair distribution, where jobs compete for resources freely without any prioritization. With `TRADL`, on the other hand, the trend is completely different. For Job-1, who enters acceptable state at 60.0s, the CPU usage decrease significantly by `TRADL`. Furthermore, when Job-1 and Job-4 achieve the objective value at 100.5s and 31.2s, the containers are terminated so that the released resource can be utilized by the other three jobs.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

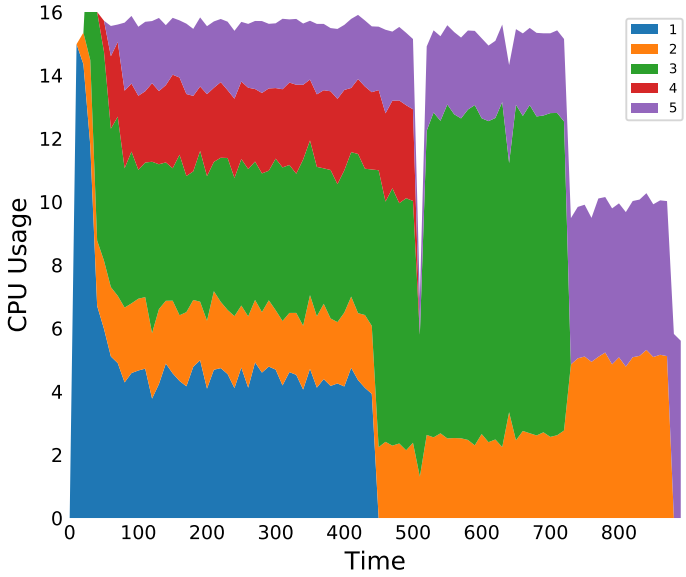


Fig. 6.3: CPU Usage of 5-Job NA

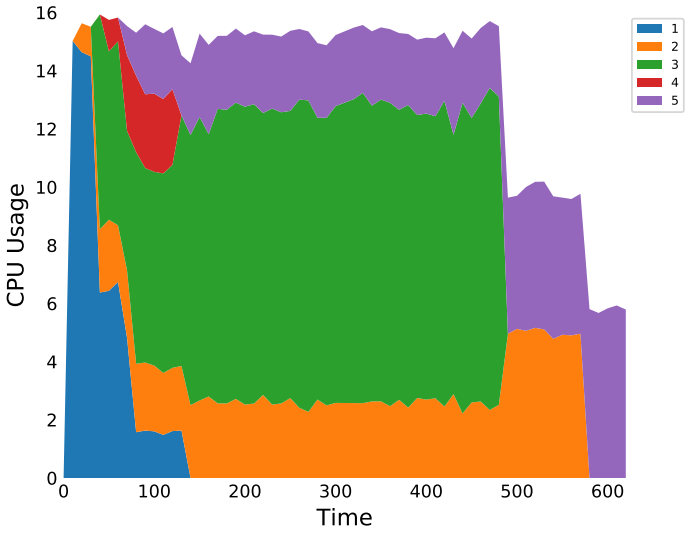


Fig. 6.4: CPU Usage of 5-Job FT

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

6.1.4 Scalability (Mixed)

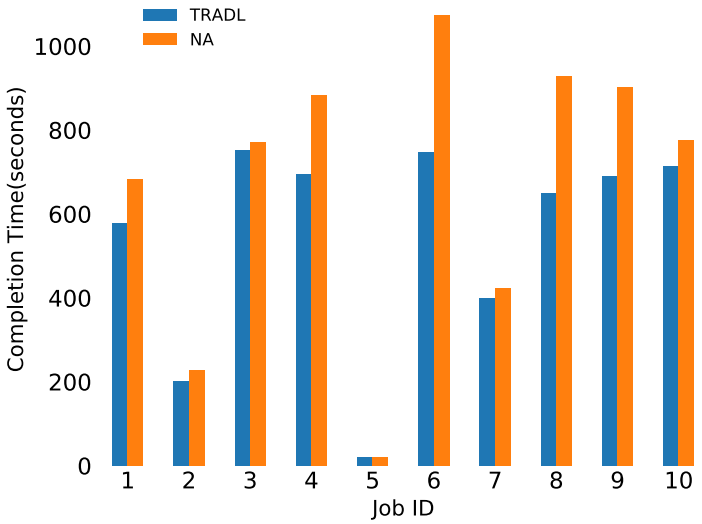


Fig. 6.5: 10 Jobs (TRADL v.s. NA)

In this subsection, we increase the number of concurrent jobs to evaluate TRADL at scale. In the experiments, 10 deep learning jobs of both AO and FT are randomly submitted to the system within the interval [0, 200]. As expected, a similar trend can be found on Fig. 6.5 such that the first job, who reaches acceptable value (Job-5) performs the same for both TRADL and NA (19.4s v.s. 19.2s). DQoES consistently improves the time cost for others. The largest reduction is recorded on Job-8, where it reduces from 762.9s to 649.2s that is a 30.7% improvement. In average, the time cost is reduced by 17.2%

Fig 6.6 and Fig. 6.7 present the CPU Usage of the above experiment. It is clear that TRADL achieves the improvement by dynamically allocate the resources with respect to the two-tier target. As shown on Fig. 6.6, all jobs share the resources equally and release only release when there is a job completes the whole training. TRADL, however, can update the resource configuration whenever necessary and terminate the job when it reaches the preset value.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

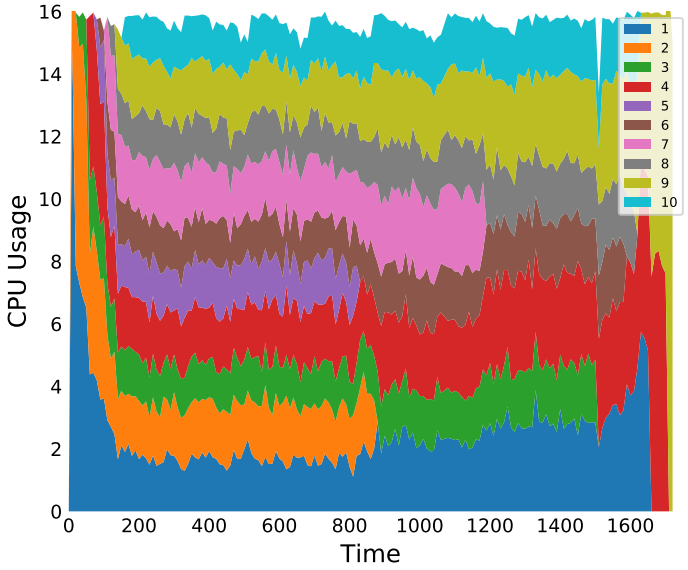


Fig. 6.6: CPU Usage of 10-Job NA

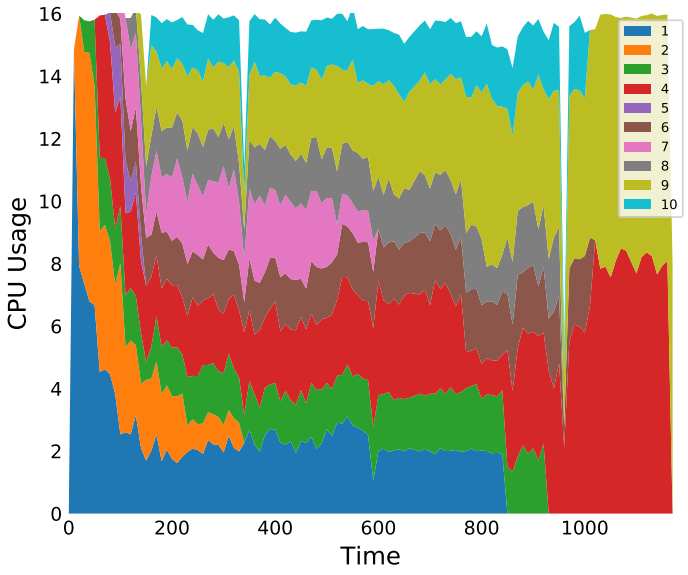


Fig. 6.7: CPU Usage of 10-Job `TRADL`

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

6.2 DQoES Performance Evaluation

In this section, we evaluate the effectiveness and efficiency of DQoES through intensive, cloud-executed experiments.

6.2.1 Experimental Framework and Evaluation Metrics

DQoES utilizes Docker Engine **engine** 19.03 and is implemented as a plugin module that runs on both local and cluster versions. We build a testbed on NSF Cloudlab [55], which is hosted by the Downtown Data Center - University of Utah. Specifically, the testbed uses the M510 physical node, which contains Intel Xeon D-1548 and 64GB ECC Memory.

DQoES is evaluated with various deep learning models using both the Pytorch and Tensorflow platforms. When conducting experiments, pretrained parameters from the platforms are loaded with the built-in workloads. The time for each image recognition task is far less than 1 second. However, the cost for real-time reconfiguration in DQoES fails to take action in seconds level since the more frequent it updates the system, the more overhead it introduces. Therefore, DQoES utilizes a batch processing of the images and define 100 images as a batch. Table 6.2 lists the models used in the experiments.

Table 6.2: DQoES -Tested Deep Learning Models

Model **pytorchexampletensorflowexample**	Platform
Visual Geometry Group (VGG-16)	P/T
Neural Architecture Search Network (NASNetMobile)	P/T
Inception Network V3	T
Residual Neural Network (Resnet-50)	P
Extreme version of Inception (Xception)	T

There are two system parameters in DQoES, (1) α, the threshold for classifying each job into different set G (outperform), B (underperform) and S (satisfied); (2) β, the value that controls the amplitude of resource adjustments at each round. Due to the page limit, we omit the discussion of these parameters. In the evaluation, we set $\alpha = \beta 10\%$. Note that these values can be easily update by the system administrator.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

The key metric that we consider in DQoES is the number of jobs in S, G and B. If all the containers are in S, it means that the system has achieved the user-specified objectives on each individual jobs and delivered best quality of experience according the clients. Thus, the difference between objectives and experiences is 0, $Q_S = 0$. It is possible that user inputs an unrealistic objective, e.g exceed theoretical boundaries. In this scenario, DQoES attempts to achieve the best possible experience to approach the objective. Therefore, metrics that we used to assess DQoES is the gap between predefined objective and real delivered experience that is presented by Q_S, Q_G, Q_B, the sum qualities for containers in set G, S and B.

We evaluate DQoES on individual severs and a cluster with 4 nodes. In addition, DQoES is tested with the following job submission schedules.

- Burst: it simulates a server with simultaneously workloads, which is challenge for DQoES to adjust resources according to each individual objective.
- Fixed schedule: the time to launch a job is controlled by the administrator. When a new job joins the system, DQoES will have to redistribute the resources to accommodate it.
- Random schedule: the launch times are randomized to simulate random submissions of jobs by users in a real cluster. DQoES has to frequently adjust resources to achieve best overall experience.

6.2.2 Single Model

We conduct a set of experiments with a single model. Each job is providing services inside a container with a specific objective. Our single model experiments use Resnet-50 as the source image.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

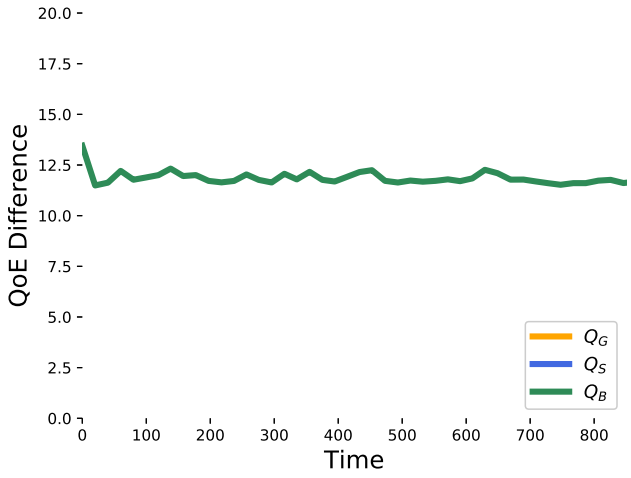


Fig. 6.8: Delivered QoE: 10 jobs with the same unachievable objectives (Burst schedule)

Identical Objectives

Firstly, we launch 10 models simultaneously to simulate the burst traffic. The objectives of all them are set to 20, which means making decisions for a batch of tasks, 100 images, take 20 seconds. Fig. 6.8 present the QoE from the user's side. As we can see from the figure, all of the containers are classified in set B, which means that they are all underperformed. This is due to the fact that, given the resources, the objective of 20 seconds per batch is unachievable. The system evenly distribute all the available resources, but average value of their QoEs is still underperformed. for example, at time 350s, the average delivered QoEs is 31.61s with a 0.35 standard deviation. Fig. 6.9 illustrates the CPU distribution among 10 models. Given an identical, unachievable objective, the best `DQoES` could do is to approach the targets by evenly distribute all the resources.

Next, we utilize the same setting, but update the identical objective values to 40, which makes it achievable by the system. Fig. 6.11 presents a more diverse result. With a dynamic resource configuration, running containers are classified into different sets. For example, at the very beginning, all of the 10 models are in set G, which means that they all perform better than the predefined objective and thus, occupy more resources than necessary. Consequently, `DQoES` starts reducing their resource limits to approach the target, 40s. At time 94.06s, running models in Container-1, 3, 6, 7, 8, 9, and 10, produce a deliverable experience, which falls within $1 \pm \alpha\times$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

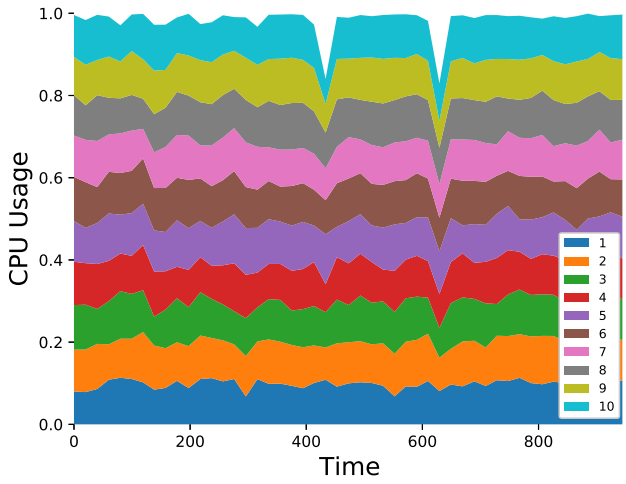


Fig. 6.9: CPU distribution: 10 jobs with the same unachievable objectives (Burst schedule)

objective ($\alpha = 10\%$ and $o_i = 40$). Therefore, they are classified to set S, which indicates they satisfied the objective. Whenever, there is, at least, one container in S, the Q_S with QoE $= 0$ shows on the figure. DQoES continues updates the resource allocation to improve QoEs of the other three models. However, at time 144.05s, container-7, 9 and 10 become underperformed with a batch cost at 47.57s, 50.91s, 47.93s. The reason is that DQoES adjust the resource adaptively and in the previous round of adjustment, it cuts too much resources from them. As the system goes, DQoES algorithms converge and all models achieve their targeted objectives. The small picture in Fig. 6.10 illustrates the number of containers in set S. We can clearly see the value goes up and down due to the adaptive adjustment, and stabled at 10 for the best performance. Fig. 6.11 presents the real-time resource distribution among all the models. In general, they obtain the same shares between each other due to the identical objective settings. In difference between Fig. 6.9 and Fig. 6.11 is that when the system stabilized, the experiments with achievable objectives has resources to accommodate for more workloads.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

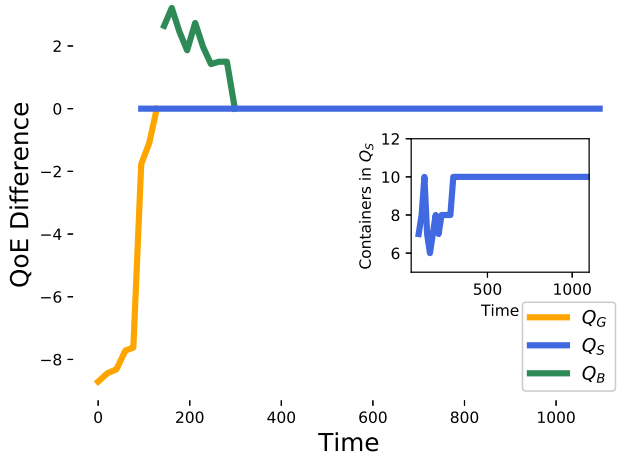


Fig. 6.10: CPU distribution: 10 jobs with the same achievable objectives (Burst schedule)

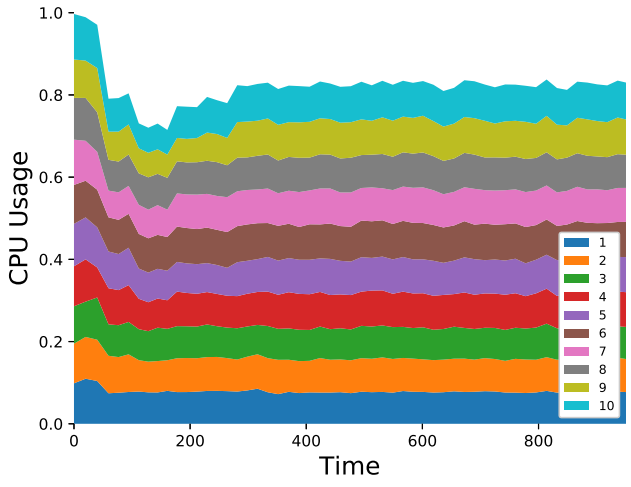


Fig. 6.11: Delivered QoE: 10 jobs with the same achievable objectives (Burst schedule)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

Varied Objectives

Next, we conduct a set of experiments with varied objectives. The values of objectives are randomly selected and consists of both achievable and unachievable targets, which generates more challenges for DQoES.

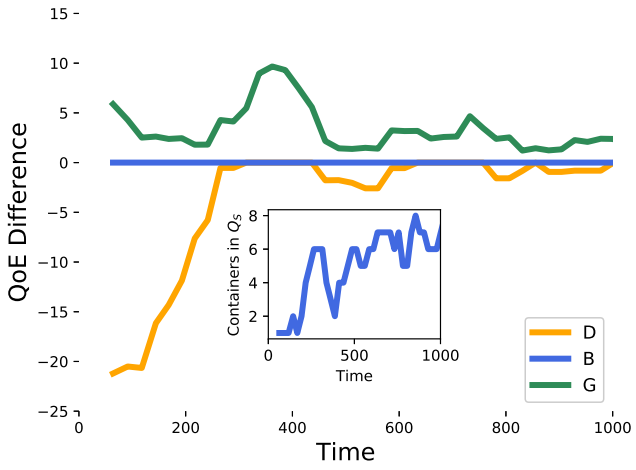


Fig. 6.12: CPU distribution: 10 jobs with varied objectives (Burst schedule)

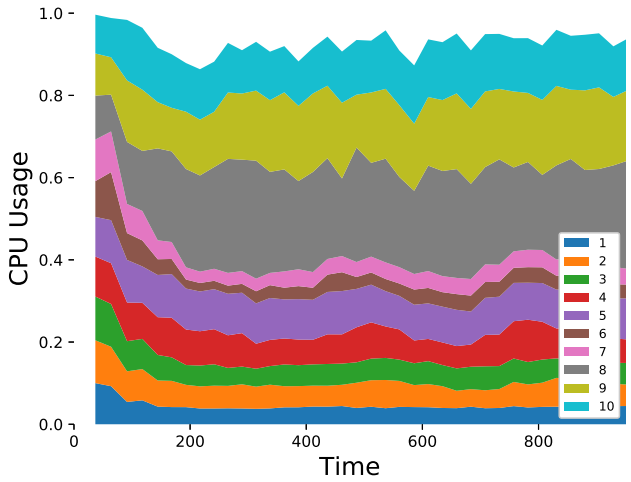


Fig. 6.13: Delivered QoE: 10 jobs with varied objectives (Burst schedule)

Fig. 6.12 and Fig. 6.13 plot the results from a burst schedule experiment. It clearly indicates that DQoES is able to redistribute resources to approach their objectives individually. At the meanwhile, DQoES achieves best overall QoE. For example, the objective values in this experiments are 75, 53,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

61, 44, 31, 95, 82, 5, 13, 25 for container 1-10, respectively. Obviously, target value 5 from container-8 is unachievable. DQoES attempts to approach this objective by allocating more resource to it than the others (Fig. 6.13). On Fig. 6.12, the number of containers in set S is stabilize 7 instead of 8 is because DQoES tries to minimize overall QoE system-wide.

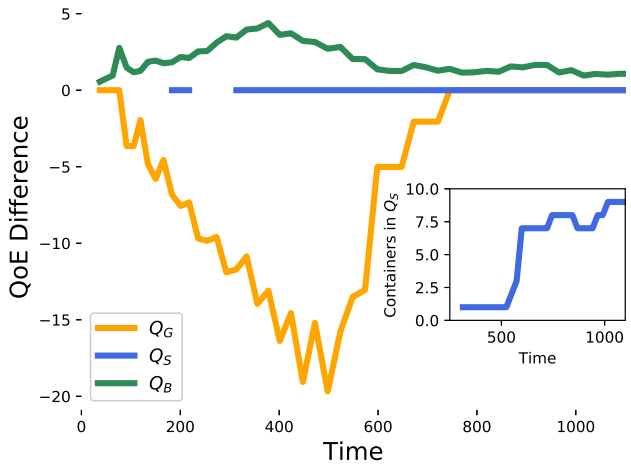


Fig. 6.14: Delivered QoE: 10 jobs with varied objectives (Fix schedule)

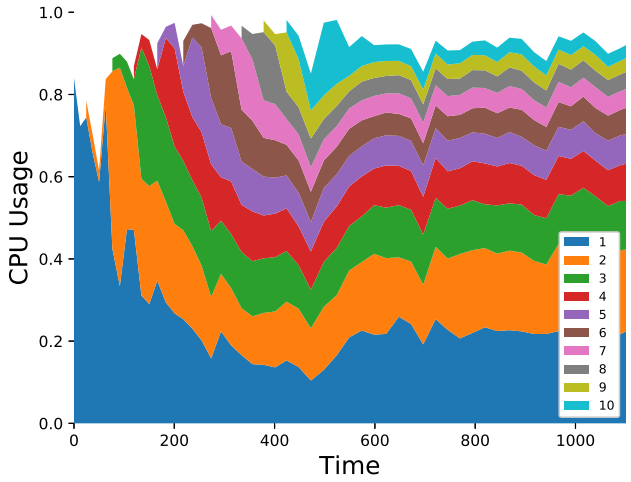


Fig. 6.15: CPU distribution: 10 jobs with varied objectives (Fix schedule)

Then, we randomly generate a set of objectives and start their host containers with a fix interval of 50 seconds. Fig. 6.14 shows the result of this experiment. Different from the previous experiments, the values of Q_B and Q_G changes rapidly. Due to the submission gap of 50 seconds, DQoES has to update the resource distribution continually. Whenever, a new container joins the system, DQoES adjusts the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

allocation to adapt new workload and its objective. Since the submission happens from 0 to 450 seconds, DQoES begin convergence after 450 seconds. At time 769.69s, the number of containers in set G becomes 0, which suggests that all the outperformed models have released resources to promote underperformed containers. As the system goes, the number of containers in S raises to 8 such that only the container-1 and 2 with unachievable targets are still in the underperformed set B. Fig. 6.15 verified the results from the resource allocation aspect that container-1 and 2 receive a larger amount of resources than others.

In this subsection, we conduct experiments with multiple models. It creates even more challenges for DQoES since different models have various resource usage patterns and diverse achievable targets. When running the experiments, we first randomly select a model image from Table 6.2 and assign an arbitrary number as its objective value. Then, each container randomly picks up a submission time from a given interval.

6.2.3 Multiple Models

Single Node Environment

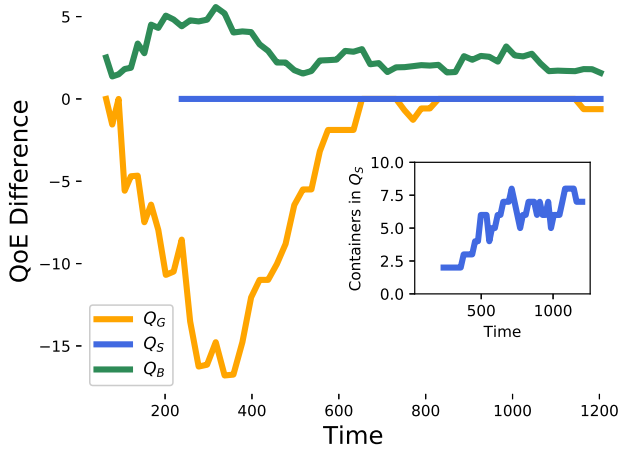


Fig. 6.16: Delivered QoE: 10 jobs with varied objectives (Random schedule)

We conduct the experiment in a single node setting, which hosts 10 running models and

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

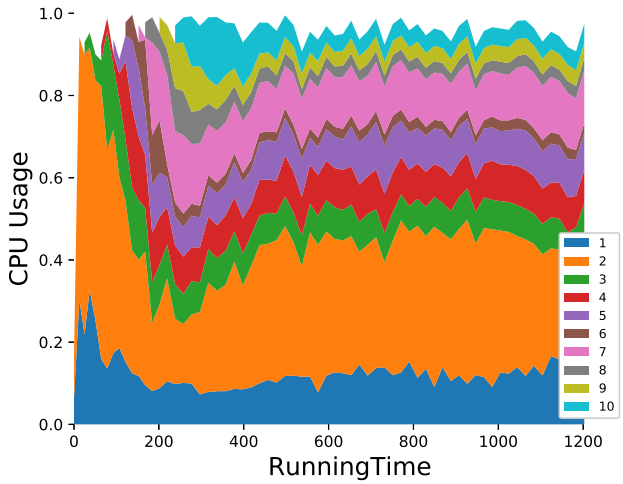


Fig. 6.17: CPU distribution: 10 jobs with varied objectives (Random schedule)

submission interval is set to [0, 300s]. A similar trend is discovered on Fig. 6.16 such that QoE of the system keeps getting worse from 0 to 300s. Due to a random submission schedule, it lacks of time for DQoES to adjust resource allocation at very beginning. However, given limited room, DQoES dynamically configures resources with respect to their individual objectives. For example, at time 257.07s, container-6's performance is 36.12s and its objective is 35. Therefore, it is classified to set S, which means it satisfied the predefined target. After 300s, the overall QoE keep approaching to 0 and the number of containers in S is increasing. With a stable workload, DQoES is able to quickly react and adjust the resources to improve the QoE. Fig. 6.17 clearly indicates that the resource is not evenly distributed and, during the submission interval, it is keep updating to adopt the dynamic workloads. At the end, it converges to a stable allocation.

Cluster Environment

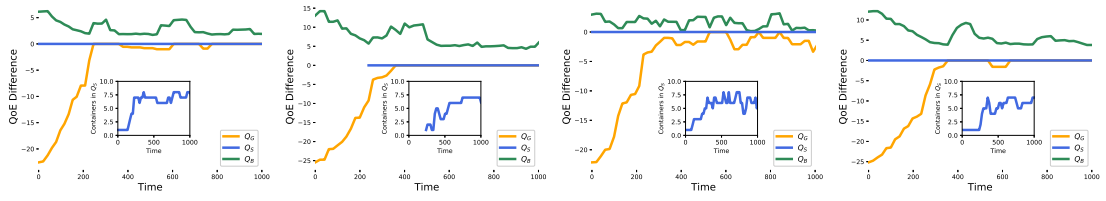
Finally, we evaluate DQoES in a cluster environment with 4 worker nodes. In these experiments, we launch 40 randomly selected models along with various objective values. Fig. 6.18 illustrates the delivered QoE of DQoES. A similar trend of results are found across all workers. The QoE keeps improving as the algorithms execute iteratively. However, depending on individual objectives that are randomly generated, the number of containers, which satisfied predefined values varies. For example,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1

there are 8, 5, 7, 6 for Worker-1, 2, 3, 4 respectively. In addition, when the first model appears in set S is also difference from each other. For instance, on Worker-4, container-32 satisfies the objective at 20.25s, where the batch cost 65.24s and the predefined target is 65. While on Worker-2, the first model that satisfies its objective is found at time 238.88s such that container-26 meet the requirement (68.01 v.s. 70). When more models with unachievable objectives, e.g. Worker-3, the number of containers in S is bouncing since DQoES tries to make the overall QoE approach to 0 and the distribution is keep updating within underperformed models.

The same experiment is conducted on the original Docker Swarm platform, where default resource management algorithms are in charge for scheduling decisions. Fig. 6.19 plots the results on each worker in the cluster. Clearly, due to missing mechanism to react on the associated objective values, whether a model can satisfy the target is purely depending on the value itself and how many concurrent running models on the same worker. As we can see that there are 1 container in set S on Worker-1, 3, and 4. On Worker-2, none of the models could meet theirs target. While not a fair comparison, DQoES is able to promote as much as 8x more models to satisfy their predefined objective values.

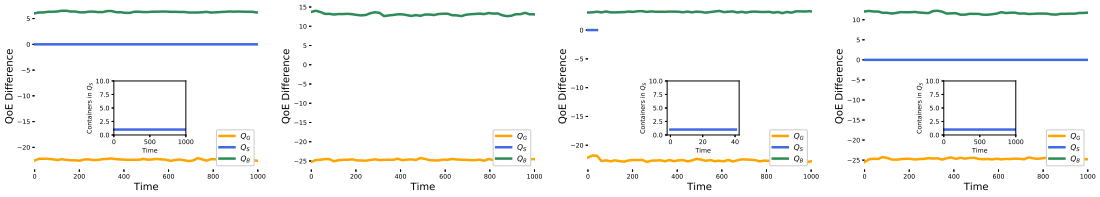
Fig. 6.20 and Fig. 6.21 present the CPU distribution of the two experiments. Comparing them, the improvement is achieved by dynamically adjust resource allocation at runtime with respects each objective values. When there is an unachievable one, DQoES attempts to fully utilize the released resources from the models with an achievable objective. With default system, however, the resource can only be evenly distributed, where each individual container gets its equal share of resource.



(a) QoE: Worker-1 **(b)** QoE: Worker-2 **(c)** QoE: Worker-3 **(d)** QoE: Worker-4

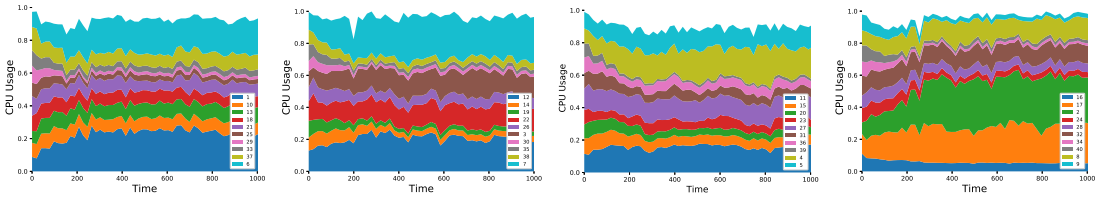
Fig. 6.18: Delivered QoE in the cluster with 40 models and varied objective with DQoES

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 November 2020 doi:10.20944/preprints202011.0017.v1



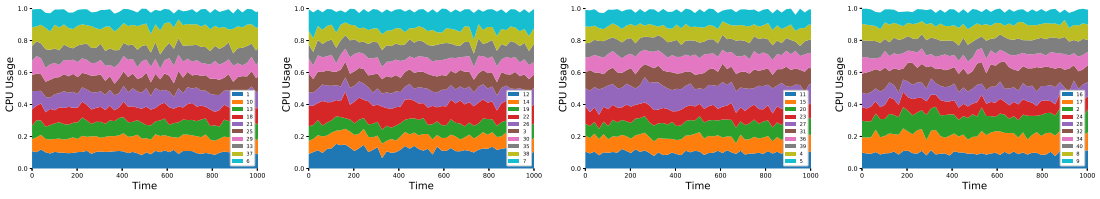
(a) QoE: Worker-1 **(b)** QoE: Worker-2 **(c)** QoE: Worker-3 **(d)** QoE: Worker-4

Fig. 6.19: Delivered QoE in the cluster with default resource management algorithms



(a) QoE: Worker-1 **(b)** QoE: Worker-2 **(c)** QoE: Worker-3 **(d)** QoE: Worker-4

Fig. 6.20: CPU distribution in the cluster with 40 models and varied objective with DQoES



(a) QoE: Worker-1 **(b)** QoE: Worker-2 **(c)** QoE: Worker-3 **(d)** QoE: Worker-4

Fig. 6.21: CPU distribution in the cluster with default resource management algorithms

 doi:10.20944/preprints202011.0017.v1

Conclusion

Considering the users' specification, including accelerating the training process and specifying the quality of experience, this thesis proposes two differentiate containers scheduling for deep learning applications: TRADL and DQoES .

To accelerate the training process, based on a two-tier target, we develop TRADL , which extracts Acceptable and Objective values of a predefined loss function from the target, monitors the training progress, and dynamically updates the resources configurations while the jobs are running. In TRADL , clients can specify a two-tier target, at this stage, TRADL reduces the resources that are occupied by this specific model so that others have more access to use. When it achieves the objective value, the training processing for the model completes and the model is ready to ship to current users. The TRADL system terminates the model, who reaches its objective, and release newly freed resources for other jobs. We implemented TRADL on the Docker platform and conducted extensive experiments with both Tensorflow and Pytorch. The results show a significant improvement, as large as 48.2% reduction, on the time cost of achieving the targets.

And to specify the quality of experience, we propose DQoES , a differentiate quality of experience based scheduler for containerized deep learning applications. When launching a back-end service, a value can be given to DQoES as a targeted QoE to an application. In a cluster,

 doi:10.20944/preprints202011.0017.v1

DQoES keeps monitoring running models as well as their associated QoE targets and tries to approach their targets through efficient resource management. When deploying a learning model, it accepts a targeted QoE value from client specifications. When multiple models are hosting in the system, DQoES is able to respect to their individual QoE targets and approach to the predefined QoE through to dynamically adjust the resources at runtime. DQoES is implemented as a plugin to Docker Swarm. Based on extensive cloud executed experiments, it demonstrates its capability to react to different workload patterns and achieves up to 8x times more satisfied models as compared to the existing system.

 doi:10.20944/preprints202011.0017.v1

Bibliography

[1] *Google adsense*, Website, https://www.google.com/adsense/start/.

[2] *Apple faceid*, Website, https://support.apple.com/en-us/HT208108.

[3] *Convolutional neural network*, Website, https://en.wikipedia.org/wiki/Convolutional_neural_network.

[4] *Recurrent neural network*, Website, https://en.wikipedia.org/wiki/Recurrent_neural_network.

[5] *Generative adversarial network*, Website, https://en.wikipedia.org/wiki/Generative_adversarial_network.

[6] *Vae*, Website, https://discuss.pytorch.org/t/multivariate-gaussian-variational-autoencoder-the-decoder-part/58235.

[7] *Yelp*, Website, https://engineeringblog.yelp.com/2015/10/how-we-use-deep-learning-to-classify-business-photos-at-yelp.html.

[8] *Relu*, https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[9] *Bidrnn*, Website, https://wizardforcel.gitbooks.io/tensorflow-examples-aymericdamien/3.07_bidirectional_rnn.html.

[10] *Amazon sagemaker*, Website, https://aws.amazon.com/sagemaker/.

[11] *Waymo*, Website, https://waymo.com/.

[12] *Usability engineering*, Website, https://www.nngroup.com/articles/response-times-3-important-limits/.

[13] W. Zheng, Y. Song, Z. Guo, Y. Cui, S. Gu, Y. Mao, and L. Cheng, "Target-based resource allocation for deep learning applications in a multi-tenancy system," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, 2019, pp. 1–7.

[14] *Docker swarm*, Website, `https://docs.docker.com/get-started/swarm-deploy/`.

[15] Y. Mao, J. Wang, and B. Sheng, "Dab: Dynamic and agile buffer-control for streaming videos on mobile devices.," in *FNC/MobiSPC*, 2014, pp. 384–391.

[16] H. H. Harvey, Y. Mao, Y. Hou, and B. Sheng, "Edos: Edge assisted offloading system for mobile devices," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, IEEE, 2017, pp. 1–9.

[17] Y. Mao, J. Wang, J. P. Cohen, and B. Sheng, "Pasa: Passive broadcast for smartphone ad-hoc networks," in *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, IEEE, 2014, pp. 1–8.

[18] Y. Mao, J. Wang, and B. Sheng, "Mobile message board: Location-based message dissemination in wireless ad-hoc networks," in *2016 international conference on computing, networking and communications (ICNC)*, IEEE, 2016, pp. 1–5.

[19] Y. Mao, J. Wang, B. Sheng, and F. Wu, "Building smartphone ad-hoc networks with long-range radios," in *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, IEEE, 2015, pp. 1–8.

[20] A. Acharya, Y. Hou, Y. Mao, and J. Yuan, "Edge-assisted image processing with joint optimization of responding and placement strategy," in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, IEEE, 2019, pp. 1241–1248.

[21] A. Acharya, Y. Hou, Y. Mao, M. Xian, and J. Yuan, "Workload-aware task placement in edge-assisted human re-identification," in *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, IEEE, 2019, pp. 1–9.

[22] Y. Mao, Y. Fu, S. Gu, S. Vhaduri, L. Cheng, and Q. Liu, *Resource management schemes for cloud-native platforms with computing containers of docker and kubernetes*, 2020. arXiv: 2010.10350 [cs.DC].

[23] Y. Mao, Y. Fu, W. Zheng, L. Cheng, Q. Liu, and D. Tao, *Speculative container scheduling for deep learning applications in a kubernetes cluster*, 2020. arXiv: 2010.11307 [cs.DC].

[24] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, and J. Murphy, "A woa-based optimization approach for task scheduling in cloud computing systems," *IEEE Systems Journal*, 2020.

[25] Q. Xie, E. Hovy, M.-T. Luong, and Q. V. Le, "Self-training with noisy student improves imagenet classification," *arXiv preprint arXiv:1911.04252*, 2019.

[26] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Large scale learning of general visual representations for transfer," *arXiv preprint arXiv:1912.11370*, 2019.

[27] H.-H. Li, Y.-W. Fu, Z.-H. Zhan, and J.-J. Li, "Renumber strategy enhanced particle swarm optimization for cloud computing resource scheduling," in *2015 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2015, pp. 870–876.

[28] S. Nathan, R. Ghosh, T. Mukherjee, and K. Narayanan, "Comicon: A co-operative management system for docker container images," in *2017 IEEE International Conference on Cloud Engineering (IC2E)*, IEEE, 2017, pp. 116–126.

[29] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in neural information processing systems*, 2017, pp. 1509–1519.

[30] D. Li, X. Wang, and D. Kong, "Deeprebirth: Accelerating deep neural network execution on mobile devices," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[31] X. Ling, S. Ji, J. Zou, J. Wang, C. Wu, B. Li, and T. Wang, "Deepsec: A uniform platform for security analysis of deep learning model," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 673–690.

[32] W. Zheng, M. Tynes, H. Gorelick, Y. Mao, L. Cheng, and Y. Hou, "Flowcon: Elastic flow configuration for containerized deep learning applications," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10.

[33] Y. Fu, S. Zhang, J. Terrero, Y. Mao, G. Liu, S. Li, and D. Tao, "Progress-based container scheduling for short-lived applications in a kubernetes cluster," in *2019 IEEE International Conference on Big Data (Big Data)*, IEEE, 2019, pp. 278–287.

[34] Y. Mao, J. Oak, A. Pompili, D. Beer, T. Han, and P. Hu, "Draps: Dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster," in *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, IEEE, 2017, pp. 1–8.

[35] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Freezeout: Accelerate training by progressively freezing layers," *arXiv preprint arXiv:1706.04983*, 2017.

[36] S. H. Hashemi, S. A. Jyothi, and R. H. Campbell, "Tictac: Accelerating distributed deep learning with communication scheduling," *arXiv preprint arXiv:1803.03288*, 2018.

[37] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ACM, 2016, pp. 50–56.

[38] M. Lopez-Martin, B. Carro, J. Lloret, S. Egea, and A. Sanchez-Esguevillas, "Deep learning model for multimedia quality of experience prediction based on network flow packets," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 110–117, 2018.

[39] *Artificial neural network*, `https://en.wikipedia.org/wiki/Artificial_neural_network`.

[40] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, IEEE, 2016, pp. 99–104.

[41] *Amazon web service, howpublished ="`https://aws.amazon.com/`",*

[42] *Microsoft azure*, `https://azure.microsoft.com/en-us/`.

[43] H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "Slaq: Quality-driven scheduling for distributed machine learning," in *Proceedings of the 2017 Symposium on Cloud Computing*, ACM, 2017, pp. 390–404.

[44] X. Tao, Y. Duan, M. Xu, Z. Meng, and J. Lu, "Learning qoe of mobile video transmission with deep neural network: A data-driven approach," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1337–1348, 2019.

[45] L. Vu, U. Kurkure, H. Sivaraman, and A. Bappanadu, "Large scale application response time measurement using image recognition and deep learning," in *International Symposium on Visual Computing*, Springer, 2018, pp. 511–526.

[46] Y. Mao, V. Green, J. Wang, H. Xiong, and Z. Guo, "Dress: Dynamic resource-reservation scheme for congested data-intensive computing platforms," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, IEEE, 2018, pp. 694–701.

[47] Z. Cao, J. Lin, C. Wan, Y. Song, Y. Zhang, and X. Wang, "Optimal cloud computing resource allocation for demand side management in smart grid," *IEEE Transactions on Smart Grid*, vol. 8, no. 4, pp. 1943–1955, 2016.

[48] Q. Zhang, Q. Zhu, and R. Boutaba, "Dynamic resource allocation for spot markets in cloud computing environments," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, IEEE, 2011, pp. 178–185.

[49] D. Ergu, G. Kou, Y. Peng, Y. Shi, and Y. Shi, "The analytic hierarchy process: Task scheduling and resource allocation in cloud computing environment," *The Journal of Supercomputing*, vol. 64, no. 3, pp. 835–848, 2013.

[50] L. Ye, "Optimization of rational scheduling method for cloud computing resources under abnormal network," in *International Conference on Advanced Hybrid Information Processing*, Springer, 2019, pp. 207–215.

[51] *Docker*, Website, `https://www.docker.com/`.

[52] *Kubernetes*, `https://kubernetes.io/`.

[53] *Tensorflow*, `https://www.tensorflow.org/`.

 doi:10.20944/preprints202011.0017.v1

[54] *Pytorch*, https://pytorch.org/.

[55] *Nsf cloudlab*, https://cloudlab.us/.