Proteus: Distributed machine learning task scheduling based on Lyapunov optimization

1st Yongan Xiang
School of computer science and technology
Xi'an Jiaotong University
Xi'an, China
xiangya@stu.xjtu.edu.cn

2nd Bin Shi
School of computer science and technology
Xi'an Jiaotong University
Xi'an, China
Shibin@xjtu.edu.cn

***Abstract*—With the prevalence of machine learning applications, an increasing number of machine learning tasks is transplanted into cloud computing platform. The cloud for machine learning consists of heterogeneous resources such as GPUs, CPUs, memory, etc, which brings challenges for resource management. So, how to schedule machine learning tasks and allocate appropriate GPU resources for computing, so that the cluster can maximize the use of resources and reduce task computing time has become a concern in the industry and academia. This paper proposes a scheduling strategy named Proteus, which is based on Lyapunov optimization. Through the Proteus, we can make our tasks have a minimum turnaround time in a long time sequence, which is the time from the submission of the task to the completion of the task. By performing a comprehensive analysis, we implement the scheduling algorithm and conducts several simulation experiments. The experimental result shows that our scheduling strategy can achieve significant results in most task scheduling environments, reducing the turnaround time of tasks to 40%-50% of the original. This shows that the Proteus can provide higher resource utilization and performance of cloud clusters and reduce task turnaround time.**

***Index Terms*—distributed machine learning, Lyapunov optimization, task scheduling**

I. Introduction

In recent years, big data technology has developed vigorously, and application data in various industries has shown explosive growth. And machine learning, as a key technology for mining and using data value, has attracted great attention in industry and academia. Because massive data can promote the algorithm model of machine learning [1], even some algorithms require large-scale data to exert their value, which makes the algorithm model continue to be complicated, and the amount of training data increases explosively, making machine learning tasks appear to be large. Big model and big data are becoming two features of machine learning tasks. For example, the machine learning algorithm of Google's famous Seti space search project uses up to 100 billion sample levels of large-scale data, and its model parameters reach a billion levels [2]. Faced with the huge demand for computing power in design, cloud vendors have released cloud computing platforms for machine learning tasks. By providing virtual machines equipped with GPU, TPU, and other heterogeneous acceleration devices, RDMA, and other high-speed network devices to meet the increasing demand for machine learning computing power demand. Due to the huge cost advantage, cloud computing platforms for machine learning tasks have gradually become an important choice for supporting massive data processing and intelligent learning. Cloud computing-related technologies for machine learning will occupy an important position in the entire intelligent industry chain and future development.

However, the cloud platform is open to anyone, so it has the characteristics of large-scale, diversified tasks and requirements. Furthermore a large number of expensive computing resources also make cloud computing platforms for machine learning tasks higher requirements for system efficiency [11]. A cloud platform that manages thousands of GPUs must be able to use reasonable resource consumption to meet user service quality requirements, ensure effective use of resources, and reduce system overhead as much as possible while improving the reliability of systems and services [10]. It has always been a key concern in industry and academia [8]. For machine learning task, it has the characteristics of large amounts of data and large model parameters. By studying the computing process of the machine learning task, we can find that the task will perform multiple repetitive calculations in the GPU and merge the trained parameters after completing computing [9]. Finally, after completing the specified number of operations, the training results will be output. In this training process, the spent time will vary according to the size of the task model and the size of the training set. It often ranging from a few minutes to a few hours, or even a few days [5]. This situation may be acceptable for some environments with small tasks. But for GPU cloud clusters which are designed for profit, machine learning tasks' long-term character often becomes a bottleneck for corporate profitability.

Considering the unique characteristic of machine learning tasks, changing the task to a distributed mode is a simple and effective way to speed up the operation. In theory, parallelization can obviously greatly accelerate the execution speed of tasks. However, blindly accelerating tasks, on the one hand, may cause a waste of resources and affect the waiting time of subsequent tasks. On the other hand, in a distributed system, parameter transfer also takes a certain amount of time. Therefore, whether parallelization is required or not, and the degree of parallelization, need to be determined according to the actual application.

In order to solve the challenge of distributed machine learning task scheduling, this paper proposes a scheduling strategy based on Lyapunov optimization. In more detail, the major contributions of this paper to the field are as follows:

- According to the relevant characteristics of machine learning tasks, we analyzed the information that needs to be known in advance when scheduling tasks. And on this basis, we have verified the impact of these characteristics on the calculation time through experiments.
- Considering the additional communication overhead required for distributed computing, we estimate the parameters of the model based on the structure of the machine learning model and combine the bandwidth of the cluster to calculate the communication time.
- Considering the multi-objective optimized cloud cluster scheduling environment, we propose a task scheduling strategy based on Lyapunov optimization. Derived through rigorous formulas, and finally transformed into a single-objective optimized decision function.

II. Task motivation

For the existing scheduling models, it is roughly divided into a scheduling strategy that quickly allocates tasks and only uses a single GPU, and a fully distributed strategy that uses more GPUs as much as possible. But according to our research, although the principle of single allocation reduces the waiting time of tasks, due to resource limits, tasks need to occupy GPU resources for a long time to complete tasks. In this case, our task's turnaround time is still a large value. The other strategy to allocate as many GPUs as possible will face the waste of resources and the effect of longer waiting time for subsequent tasks. In this way, it seems that neither of the two methods can achieve the multi-objective optimization of machine learning task scheduling under the cloud cluster [14].

Therefore, we thought that if in a long-term sequence, we can reasonably convert optimization objectives such as queue length, task waiting time, and number of idle resources into constraints, so that the problem only becomes how to solve the average minimum turnaround time of tasks [12] . Then we can make reasonable decisions for each moment of the long-term sequence according to our optimization goals. In the end, we can achieve the minimum average turnaround time in the entire scheduling process.

According to the above analysis, after considering various factors of distributed machine learning scheduling, we propose our scheduling strategy and named Proteus: a distributed machine learning task scheduling strategy based on Lyapunov optimization. The reasons are as follows:

- First of all, our task is a multi-objective optimization task. In the process of using Lyapunov optimization, some optimization objectives can be converted into constraints to reduce the optimization objectives, which is convenient for us to calculate.
- The characteristic of Lyapunov optimization is that we do not need to know any information about the future during the scheduling process. We only need to make scheduling decisions based on the current queue backlog and the cluster operating environment.
- The Lyapunov optimization is to greedily minimize the drift multiplication in each time slot (that is the cost we propose later), to obtain the optimal solution for long-term scheduling, not the optimal solution at every moment.

III. Scheduling design

In this section, we first propose some assumptions about the scheduling environment based on the characteristics of machine learning task scheduling. Subsequently, we analyzed the relevant information that needs to be understood in advance during out Proteus process. The purpose of doing this is to distinguish the task as a distributed advantageous type or a single GPU computing advantageous type through task information.

A. Scheduling hypothesis

For tasks submitted by users to a GPU cluster, we generally store the tasks in the buffer queue in the order submitted by the users, and then deploy the tasks to the appropriate location of the cluster for calculations at an appropriate time. Here, we want to take into account the factors of task fairness and prevent task starvation, so we choose the traditional FIFO algorithm to perform scheduling analysis on the tasks in the queue. In the tasks submitted by users, there will be designated files used to describe the related information of the tasks, and our cluster controller can perform scheduling analysis in advance through these designated files to facilitate our subsequent scheduling.

In addition, when multiple tasks are placed on the same GPU for calculation, there will be a problem of bandwidth occupation. When two compatible tasks and two incompatible tasks are calculated in the same GPU, there will be a very big difference in time. In order to avoid this additional factor from producing unexpected results for our scheduling decision, we believed that a GPU can only be occupied by one task or a part of the task at any time. The following are some of the parameters we need to consider.

B. Task calculation time

For a machine learning task, due to its large-scale data and recalculation, a task may take a long time to calculate in the cluster. During this period of time, tasks that cannot be placed in the cluster can only be in a ready state in the waiting queue, and only when a task leaves the cluster, will subsequent tasks be placed in the cluster for calculation. Therefore, in the scheduling process, the first thing we need to consider is the actual calculation time of a task.

In our model, we assume that the queue is scheduled according to the FIFO algorithm, so under normal circumstances, we will give priority to the task that enters the buffer queue first.

As we understand that the computing power of the GPU is not only related to the attributes of the GPU devices in the cluster but also related to the model structure of the task itself and the batch size set by the task. To more reasonably grasp the accuracy of scheduling estimation in different environments, we test the task model in advance before task scheduling. Select a smaller number of pictures to pre-estimate the time of the task to obtain the total computing time of the task when the batch size is 1. After that, we can derive the estimated computing time of tasks under different batch size conditions based on the relationship between batch size and GPU computing power. This part of the analysis will be detailed described in the third party.

C. Communication time

Since our task is based on a distributed machine learning task, we will distribute the data to each GPU participating in the calculation while performing the calculation during the task. And when each GPU completes independent calculations, we also need to merge the parameters, so that we can make the model continue to calculate in the correct direction. In the communication process, the time spent in communication will be related to the size of our model and the actual bandwidth between GPUs in the cluster. Related experiments have proved that in the case of a small task calculation, our communication cost may take up a larger part of the time, which in turn affects our overall calculation time. Therefore, when we schedule tasks, we must consider the communication costs brought about by the scheduling.

IV. Scheduling parameter analysis

In the process of designing the scheduling policy, we briefly described the scheduling decisions that we need to consider. In this section, we will formulate descriptions of these variables respectively, so that our variables can be realized in the final calculation and scheduling process, and then we will make a simple deformation derivation of the optimization theory formula to meet our actual situation.

A. Analysis of task calculation time

We need to explain here that for different clusters, the GPU equipment may be different. The image loading speed of the cluster, cache, system optimization, and other factors will affect the calculation time of the task. Therefore, to make the scheduling algorithm universally adaptable to different clusters, our scheduling design needs to preprocess the tasks.

Through our experimental analysis, we learned that GPUs have different computing capabilities for different machine learning models. In addition to the structure of the model itself, it is also related to the task-related training parameters set by the user. Fig. 1 shows the relationship between the calculation time of different models and the model batch size obtained during our research. In our scheduling, through preprocessing analysis, the model calculation time can be estimated when the batch size is 1. For the estimated calculation time corresponding to the remaining batch sizes, we conducted correlation analysis on the data and found that the model calculation time has the following relationship under different batch sizes:

$$Cost_{bat(b)} = \alpha(b)Cost_{bat(1)} \tag{1}$$

Among them, $\alpha(b)$ represents the ratio parameter between the estimated calculation time and the pre-test time (in the case of a batch size of 1) when the batch size is b.

To verify the rationality of our proportional parameters, we solved the theoretical estimation time of some models and then verified them in the actual cluster environment. The relevant data are as follows:

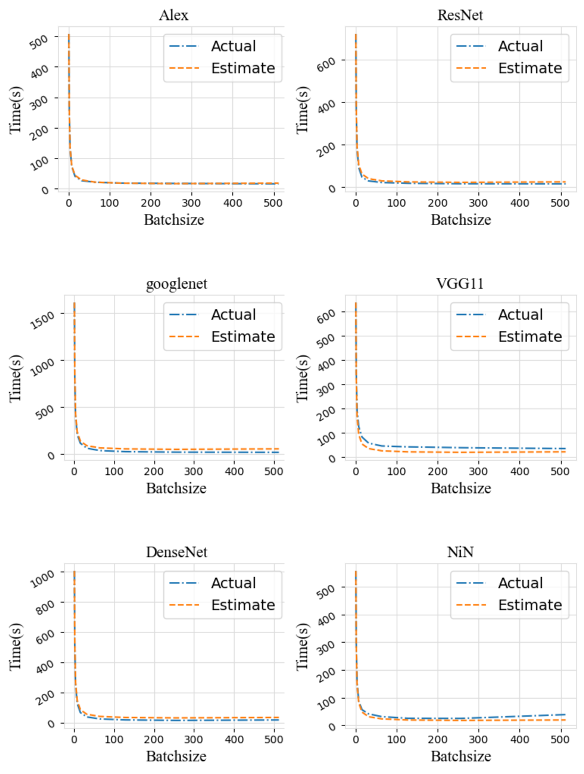

Fig. 1. The relationship between normal model with batchsize

We can see that in most cases, the error between our estimated value and the actual value is kept within a relatively small range, but as the batch size increases, the error of some models also begins to increase. But in fact, under the absolute error, the difference between the actual time and the theoretical time is insignificant compared to the entire long-term task scheduling, so the error under large batch size will not affect our scheduling.

B. Analysis of model communication time

In data-parallel machine learning tasks, each GPU will merge all data results once after completing the calculation. At this time, GPUs need to communicate, and the communication content is the weight matrix of the model. It can also be understood as the parameters of the model.

Different machine learning models have different parameters. We need users to provide the structural information of the model in advance during scheduling analysis, and then our cluster can estimate the model parameters of the task. For common models, we can obtain the calculation formulas based on the principles of machine learning.

In the convolutional layer, the amount of parameters is related to the size of our input and output channels and the convolution kernel, so the size of the parameters calculated at one time is as follows:

$$Params = C_{in} * C_{out} * K^2 \tag{2}$$

C_{in} represents input channels' number, C_{out} represents output channels' number, and K represents the kernel size.

In the fully connected layer, our calculation formula is:

$$Params = H * W * C_{in} * C_{out} \tag{3}$$

H, W represents the shape of the input graph.

Through estimation calculation, we obtain the parameters of some common models as follows:

TABLE I
PARAMS OF COMMON MACHINE LEARNING MODELS

Model	***Alex***	***ResNet18***	***ResNet50***	***GoogleNet***
Params(M)	61.1	25.56	11.69	6.6
Model	***DenseNest121***	***VGG11***	***VGG16***	***Inception_V3***
Params(M)	7.98	132.86	138.36	27.16

V. Optimization target

According to the previous analysis, we learned that calculation time and communication time will be related to the number of GPUs we choose for a task. For our long-term scheduling tasks, we want to see that our scheduling method can complete the task in a short time, and the task can also make full use of the resources in the cluster. By the Lyapunov optimization scheduling method, for our multi-objective optimization scheduling problem, we will convert the optimization goal of the tasks' accumulation into our virtual queue, so that the problem can be transformed into a single-objective optimization. In this case, we can facilitate scheduling decisions by solving the minimum average turnaround time of tasks. The following is our specific analysis.

A. Calculation time

According to the calculation, we have obtained the formula for the estimated calculation time of tasks' single batch as (1).

For the total calculation time of the task and considering the batch size, we also need to consider the size of the data set provided by the task. Here, we assume the data set size as m. Besides, considering that if the task's data set is small, one training can't make the parameters converge, we may also retrain the data set. Therefore, we also need to consider the epoch of the task, here we assume the epoch as e. So we can determine the task's total running time according to the number of GPUs used, the formula is (4).

$$Cost(n) = cost_{bat(b)} \frac{em}{bn} \tag{4}$$

B. Communication time

For communication time, we can make the following analysis. First, we assume the bandwidth between GPUs in the cluster is BW. For a task with a data volume of m, if the task processing size is preset to b, then we can find the iter value corresponding to the task is $iter = \frac{m}{nb}$, then for distributed computing, the total number of parameter merging is $\frac{m}{nb} \times e$. Here we will assume that during the communication process, each GPU is transmitting at the same time, so we can get the total communication time during the entire task execution process as (5).

$$Comm(n) = (\frac{m}{nb} \times e) \times \frac{params}{BW} \tag{5}$$

C. Optimization target

For each task, if it can be completed in a faster manner in the cluster according to our scheduling method, then the sum of the task's calculation time and communication time will be a smaller value. This is equivalent to that we can always maintain the average task turnaround time at a small value. (V-C) is the formula of our optimization decision objective.

$$\begin{aligned} min.C(n) = Cost(n) + Comm(n) \\ sub: 0 \leq n \leq N \\ n \in Z^+ \vee 0 \end{aligned} \tag{6}$$

N represents the maximum number of GPUs in the cluster that can be used at this moment.

VI. Lyapunov optimized scheduling policy

Based on the previous introduction, we know that Lyapunov optimization is a method of greedily selecting the current minimum drift penalty [3] [13]. In this section, we will reasonably use the Lyapunov optimization method to schedule our task queue.

A. Task buffer queue

We set Q(t) to represents the queue length of a working GPU cluster at time t, and we We set Q(t) to represent the queue length of a working GPU cluster at time t, and we use the estimated task time when the task is first put into the cluster using a batch size of 1 as the task's accumulated length. Because assuming one task takes a short time, it will not have a significant impact on GPU scheduling, but when the task takes a long time, our scheduling may be greatly affected.So we can know the current accumulation of tasks through the length of the queue. The specific formula is (7).

$$Q(t+1) = \max[Q(t) - kCost_i(1), 0] + \sum_{r \in R} Cost_j(1) \tag{7}$$

$Cost_i(1)$ represents the length of the task at the head of the queue and i represents the number of the task. Since our decision-making is based on the FIFO algorithm, in general, we will prioritize the scheduling analysis from the task at the head of the queue. k represents a switch [ON.OFF], when $n > 0$, it means tasks are placed in n GPUs for work. In this case, $k_i = 1$. In other situation, $k_i = 0$. $Cost_j(1)$ represents the new task with number j in the time slot and it will be placed at the end of the queue. R represents all sources which can submit tasks.

B. Delay tolerant queue (virtual queue)

Here, we want to transform the optimization goal of task waiting time. In the optimization model, we construct a new virtual queue and call it a delay-tolerant queue [7].

The reason for the additional setting of a virtual queue, we can understand that our decision about every task can positively affect the waiting time of subsequent tasks in the queue. But our task queue Q cannot provide dynamic feedback about our decisions. That is, if our previous task has a good backlog in the queue, it means the number of tasks is not large, and the head task can wait for a longer time to obtain the optimal scheduling method.

In this case, when the subsequent task arrives at the head of the queue for scheduling analysis, due to the previous task's optimal decision, our virtual queue $Z(t)$ may already have a large length. It can be also understood that the previous decision has made our subsequent tasks have a long waiting time. In this case, we need to maintain the stability of the queue, so we can only choose a non-optimal decision to schedule the task until our virtual queue $Z(t)$ gradually becomes smaller and the queue backlog returns to a better state. Throughout the long-term task scheduling process, our decision-making will repeat this process continuously.

Based on this idea, we designed our virtual queue calculation formula as (8).

$$Z(t+1) = \max[Z(t) + l_{Q>0}(\epsilon_d - kCost_i(1)) - l_{Q=0}\frac{Z(t)}{2}, 0] \tag{8}$$

Because Lyapunov optimization will greedily choose the optimal situation at this moment, but in the long-term time series, this choice may make the queue in a worse state (the queue is too long), therefore, our virtual queue uses ϵ-persistent service strategy design [4], where the value of ϵ reflects the tolerance of the waiting time of our design task. Besides, the indicator function $l_{Q>0}$ means that when $Q > 0$ is 1, It is 0 at other times. The same applies to $l_{Q=0}$. When the queue is empty, we think that the current queue is in good condition, so our delay tolerance queue will also quickly approach a smaller value.

C. Lyapunov optimization function

According to the two queues we built above, since the scheduling is analyzed based on the current backlog of the queue, we merge the two queues to construct a joint matrix:

$$\Theta(t) = [Q_d(t), Z_d(t)] \tag{9}$$

After that, according to the Lyapunov framework and based on the joint matrix, we define the Lyapunov function as (10).

$$L(\Theta(t)) = \frac{1}{2}\{Z_d(t)^2 + Q_d(t)^2\} \tag{10}$$

$L(\Theta(t))$is a measure of the backlog of tasks in the queue. Now, we can also define the single-slot Lyapunov drift function.

$$\Delta\Theta(t) = E\{L(\Theta(t+1)) - L(\Theta(t))|\Theta(t)\} \tag{11}$$

Since our optimization goal is to make the long-term machine learning task scheduling have a minimum turnaround time, in addition to the construction of the Lyapunov drift function, we need to add a penalty function, which is the time the task spends in the cluster. Subsequently, we can transform the formula (11) into Lyapunov drift-penalty term (12).

$$\Delta\Theta(t) + VE\{C(n)|\Theta(t)\} \tag{12}$$

In the process of calculation, it does not directly minimize the drift-penalty function but strives to minimize its upper bound. Moreover, theoretically proves that this method does not destroy the optimality and performance of the scheduling [3]. For the formula on the right, we need to solve for the minimum value. So the problem is transformed into how to allocate the number of GPUs so that the right side of the equation obtains the minimum value under the current backlog queue. The formula is (13).

$$\begin{aligned}\Delta\Theta(t) &= \frac{1}{2}\{Q(t+1)^2 - Q(t)^2\} + \frac{1}{2}\{Z(t+1)^2 - Z(t)^2\} \\ &\le B - kCost_i(1)\{Z(t) + Q(t)\} + (kCost_i(1))^2\end{aligned} \tag{13}$$

here

$$\begin{aligned}B = &\frac{1}{2}\{\epsilon_d^2 + (\frac{Z(t)}{2})^2 + 2Z(t)\epsilon_d \\ &+ (\sum_{r\in R} Cost_j(1))^2 + 2(\sum_{r\in R} Cost_j(1))Q(t)\}\end{aligned}$$

We need to explain that we have simplified the right side of the formula into two parts. The basis of classification is whether the variables in the formula are related to our scheduling. It can be seen that at a certain moment, the length of each queue and the amount of newly arrived tasks in the queue are constant. Only the variable k can be changed when we allocate GPU, which can affect our formula, so we extract the term that contains k.

Finally, by adding$VE\{C(n)|\Theta(t)\}$to the right side of the formula again, the decision function in the scheduling process is formed as (14). Among them, V is a non-negative parameter, which can be a compromise between queue stability and cluster computing efficiency. The larger the V, the higher the computing efficiency of the cluster; otherwise, the lower the efficiency [3].

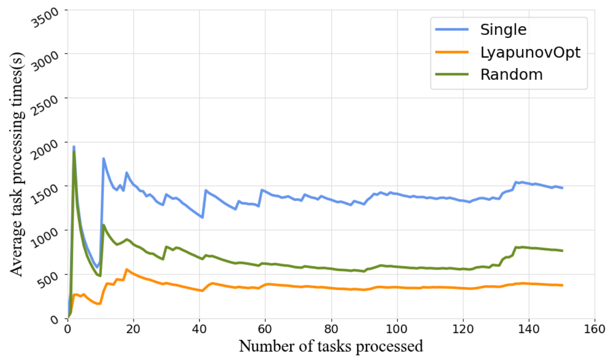

Fig. 2. Small data set, dispersion

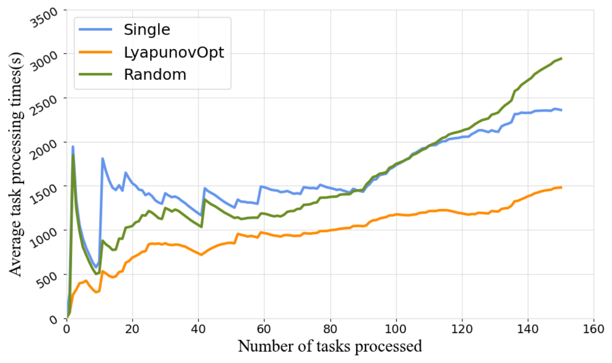

Fig. 3. Small data set,concentration

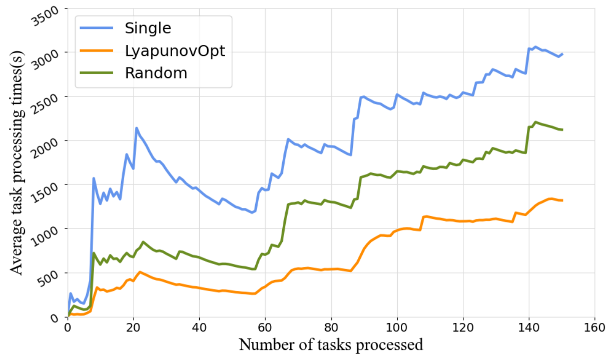

Fig. 4. Big data set, dispersion

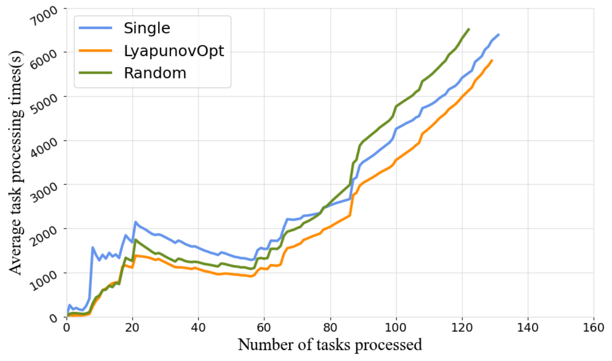

Fig. 5. Big data set,concentration

$$\begin{aligned} min.S(n) = VE\{C(n)|\Theta(t)\} + (kCost_i(1))^2 \\ - kCost_i(1)\{Z(t) + Q(t)\} \end{aligned} \tag{14}$$

According to formula (14), we analyze: when the task is not assigned to the cluster at this moment, the running time of task i is 0, and because $n = 0$, so we have $k = 0$, then for the formula, there will be $S(0) = 0$. We can understand that when we do not schedule at this moment, our scheduling cost is 0. So the condition that we can allocate GPU is that the S(n) is a negative value, which means we can obtain a smaller value from scheduling than doing nothing so that we can schedule, otherwise, we will continue to wait. At the next moment, the decision will be made again. Finally, we get the conclusion of the scheduling decision:

$$n = \begin{cases} n & \text{if } (S(n) < 0)\text{and}(\forall m \in (0, N], S(n) \leq S(m)) \\ 0 & else \end{cases} \tag{15}$$

The calculation results show that in each time slot, the tasks' assignment will be more inclined to allocate n GPUs with the lowest task cost to work, and when the optimal situation cannot be met, our scheduling will be based on the current queue backlog length, and consider whether to implement a non-optimal allocation strategy.

VII. Experimental verification

In this section, we will verify our Lyapunov optimized scheduling strategy in the actual environment.

A. Experimental design

1) Laboratory equipment: We used RTX2080Ti for calculation and analysis in the early small-scale test experiment. For long-term serial experiments, due to the limitations of laboratory equipment, we use programming simulation for implementation and verification.

2) Model selection: In our experiment, we selected four types of commonly used machine learning models for test calculations, they are VGG16, Alex, GoogleNet, RestNet18.

3) Scheduling strategy: To verify that in the Proteus process, it is our scheduling strategy that plays a major role in task optimization scheduling, rather than distributed reason. We designed two comparative scheduling strategies. The first scheduling strategy is to allocate only a single GPU to each task, and the second scheduling strategy is to perform randomly distributed scheduling on all tasks, randomly specifying the number of GPUs required by the task.

4) Task design: We designed four different task environments for our Proteus, which are: small data models are mostly and the tasks reach the cluster time dispersion; small data models are mostly and the tasks reach the cluster time concentration; large data models are mostly and the task reaches the cluster time dispersion; large data models are mostly, and the tasks reach the cluster time concentration.

B. Experimental results

Our experiment recorded the average tasks' turnaround time based on different scheduling methods. And we also use the same task sequence in our experiment for different scheduling strategies. The detailed results are shown in 'Fig. 2-Fig. 5'.

C. Result analysis

We can see from the results that the Proteus model has good results for the centralized arrival and decentralized arrival of small data models, and the decentralized arrival of big data models. Compared with the scheduling of a task using one GPU and the distributed scheduling of randomly allocating the number of GPUs, our optimization model can greatly reduce the average turnaround time of the task. Compared with the results of only using a single GPU scheduling, our optimized scheduling can reduce the original time by 50%-60%; compared to the randomly distributed scheduling, our average task turnaround time is also reduced by about 30%. It can be seen from the experimental data that our optimization model has good adaptability and efficiency.

Compared with the other two scheduling strategies, in a scheduling environment where tasks with a large amount of data arrive intensively, although the average task turnaround time is also reduced, the magnitude is not as obvious as in the previous cases. This is because, in a large data set task environment, the speed of cluster processing tasks may be slower than the speed of task arrival, resulting in the arrival of task concentration, which will change over time and accumulate in the cache queue. Our optimization model is based on the current optimal greedy scheduling method. The accumulation of task queues is poor, which will make our scheduling move in the direction of quickly assigning tasks to reduce queue accumulation. Therefore, in the long-term scheduling, tasks may tend to be distributed to a single GPU, which also leads to the experimental results that our scheduling method does not play a very good role. But in fact, in our daily task scheduling process, we seldom encounter the situation where a large number of large data volume models are concentrated into the cluster for scheduling under the long-term sequence set in the experiment. Therefore, the Proteus still has the ability to significantly reduce the turnaround time for general task scheduling.

VIII. Conclusion

This article introduces a scheduling method named Proteus based on Lyapunov optimization to perform distributed scheduling of machine learning tasks in GPU clusters. Considering the large data sets of machine learning tasks and the characteristics of large model parameters, the tasks are reasonably scheduled according to the available resources of the cluster at each moment. Converting multiple optimization goals such as task waiting time and running time into a single optimization goal. The final experimental results show that the Proteus model can obtain good experimental results in most task environments compared to the current common scheduling strategies. The average task turnaround time has been reduced to 30%-50% of the traditional situation, which improves the resource utilization of the cluster and can also provide users with better services.

Acknowledgement

This research was partially supported by National Science Foundation of China 62002282, and "The Fundamental Theory and Applications of Big Data with Knowledge Engineering" under the National Key Research and Development Program of China with Grant No. 2018YFB1004500, and the MOE Innovation Research Team No. IRT17R86.

References

[1] Dean J. Large-scale deep learning for building intelligent computer systems. Keynote on the ninth international conference on web search and data mining, WSDM2016.

[2] Hirabayashi H. Seti (search for extraterrestrial intelligence)[M]. Astrobiology. Springer,Singapore, 2019: 451-459

[3] Neely MJ. Stochastic Network Optimization with Application to Communication and Queueing Systems. Morgan and Claypool, 2010.

[4] Neely MJ. Opportunistic scheduling with worst case delay guarantees in single and multi-hop networks. In Proceeding of the INFOCOM IEEE,2011. 1728-1736. [doi:10.1109/INFCOM.2011.5934971]

[5] Chris Edwards. 2015. Growing Pains for Deep Learning. Commun. ACM 58, 7(jun 2015), 14–16. DOI:http://dx.doi.org/10.1145/2771283

[6] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," in CoRR abs/1404.5997,2014.

[7] Xiao Wenhua, Bao Weidong, Zhu Xiaomin, Shao Yiyang, Chen Chao, Jianhong Wu. Cost minimization method for multi-source big data cloud processing[J]. Journal of Software, 2017, 28(03): 544-562 .

[8] Z. Chen et al., "Deep Learning Research and Development Platform: Characterizing and Scheduling with QoS Guarantees on GPU Clusters," in IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 1, pp. 34-50, 1 Jan. 2020, doi: 10.1109/TPDS.2019.2931558.

[9] Xiao W, Bhardwaj R, Ramjee R, et al. Gandiva: Introspective cluster scheduling for deep learning[C]//13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018: 595-610.

[10] Mayer R, Jacobsen H A. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools[J]. ACM Computing Surveys (CSUR), 2020, 53(1): 1-37.

[11] Amaral M, Polo J, Carrera D, et al. Topology-aware gpu scheduling for learning workloads in cloud environments[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2017: 1-12.

[12] Lin Chuang, Chen Ying, Huang Jiyan, et al. Research on multi-objective optimization model and solution of service quality in service computing[J]. Chinese Journal of Computers, 2015, 38(10): 1907-1923.

[13] Li Lei, Xue Yang, Lu Nianling, et al. Design of online task and resource scheduling for container cloud queue based on Lyapunov optimization[J]. Journal of Computer Applications, 2019, 39(02):190-196.

[14] Peng Y, Bao Y, Chen Y, et al. Optimus: an efficient dynamic resource scheduler for deep learning clusters[C]//Proceedings of the Thirteenth EuroSys Conference. 2018: 1-14.