Liquid: Intelligent Resource Estimation and Network-Efficient Scheduling for Deep Learning Jobs on Distributed GPU Clusters

Rong Gu, Yuquan Chen, Shuai Liu, Haipeng Dai, Guihai Chen, Kai Zhang, Yang Che, and Yihua Huang

背景：1) 现有GPU 资源管理需要用户指定，低利用率；2) 未考虑集群的网络特性，导致低 job 性能。

Liquid 是一个高效的 GPU 资源管理平台，支持智能资源需求评估和调度。Liquid 使用一个回归模型来评估 job 的资源需求；提出集群网络高效的调度策略 (即时或批模式)；提出包括预调度数据传输，细粒度 GPU 共享和事件驱动通信在内的三个系统级优化

Abstract—Deep learning (DL) is becoming increasingly popular in many domains, including computer vision, speech recognition, self-driving automobiles, etc. GPU can train DL models efficiently but is expensive, which motivates users to share GPU resource to reduce money costs in practice. To ensure efficient sharing among multiple users, it is necessary to develop efficient GPU resource management and scheduling solutions. However, existing ones have several shortcomings. First, they require the users to specify the job resource requirement which is usually quite inaccurate and leads to cluster resource underutilization. Second, when scheduling DL jobs, they rarely take the cluster network characteristics into consideration, resulting in low job execution performance. To overcome the above issues, we propose Liquid, an efficient GPU resource management platform for DL jobs with intelligent resource requirement estimation and scheduling. First, we propose a regression model based method for job resource requirement estimation to avoid users over-allocating computing resources. Second, we propose intelligent cluster network-efficient scheduling methods in both immediate and batch modes based on the above resource requirement estimation techniques. Third, we further propose three system-level optimizations, including pre-scheduling data transmission, fine-grained GPU sharing, and event-driven communication. Experimental results show that our Liquid can accelerate the job execution speed by 18% on average and shorten the average job completion time (JCT) by 21% compared with cutting-edge solutions. Moreover, the proposed optimization methods are effective in various scenarios.

Index Terms—Job scheduling, resource management, deep learning, GPU clusters

1 Introduction

In recent years, Deep learning (DL) technology has been widely used in broad intelligent applications. GPU can usually run DL jobs more efficiently than CPU because of the massively parallel architecture. For example, experiments in literature [1] show that training a logistic regression model on Theano [1] framework with MNIST [2] datasets can process up to 38,000 records per second on GPU (Geforce GTX 285), but only 6,500 records per second on CPU (Intel Core 2 Duo E8500) in the same environment. Therefore, GPU becomes one of the most essential resources for running DL programs.

Because GPU resources are usually expensive, it is cost-inefficient to allocate the entire GPU clusters for each user (group) in practice. Moreover, a user usually does not train models all the time, which means exclusive allocation of all the GPU resources is not necessary. Thus, a natural way to optimize resource utilization is to share GPU resources among a group of users.

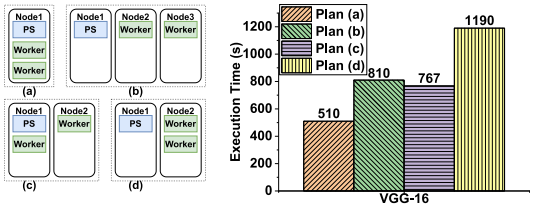
However, typical GPU resource sharing schemes face issues of resource usage conflicts, especially in multi-tenant DL GPU clusters [3]. To avoid conflicts and make full use of resources, we need efficient GPU management and intelligent scheduling methods to ensure fast DL job execution.

Accurate DL job resource requirement estimation and intelligent resource scheduling are two main challenges for building efficient multi-tenant DL job running platforms. Determining appropriate amount of computing resources for each given DL job is the prerequisite for job resource scheduling. Generally, because of the law of diminishing marginal utility, there exists an appropriate value of the computing resources amount for running a given DL job. Beyond that amount, the job execution time is similar, but the extra allocated computing resource is wasted. However, due to lack of distributed system knowledge, it is challenging for the data scientists or analysts to set accurate resource requirement for DL programs when submitting their DL jobs to clusters. Manual or simple rule-based resource requirement configuration methods would lead to job running failure or cluster resources abusing, which is not good for overall resource utilization and job execution performance.

Even though the job resource requirement is somehow determined, scheduling the GPU cluster resources to run

- *Rong Gu, Yuquan Chen, Shuai Liu, Haipeng Dai, Guihai Chen, and Yihua Huang are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu 210023, China. E-mail: {gurong, haipengdai, gchen, yhuang}@nju.edu.cn, {yqchen, liushuai}@smail.nju.edu.cn.*
- *Kai Zhang and Yang Che are with Alibaba Group, Hangzhou, Zhejiang 311121, China. E-mail: {zk131586, cheyang.cy}@alibaba-inc.com.*

Manuscript received 31 Aug. 2021; revised 12 Nov. 2021; accepted 12 Dec. 2021. Date of publication 28 Dec. 2021; date of current version 23 May 2022.
This work was supported in part by the China National Science Foundation under Grant 62072230, in part by Alibaba Group through Alibaba Innovative Research Program, the China National Science Foundation under Grants U1811461, 61832005, and 61702254 and in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization.
(Corresponding authors: Haipeng Dai and Yihua Huang.)
Recommended for acceptance by A.J. Peña, M. Si and J. Zhai.
Digital Object Identifier no. 10.1109/TPDS.2021.3138825



(a) Scheduling plans (b) Job execution time

Fig. 1. Example: different plans and the corresponding execution time for a VGG-16 DL job with the same amount of resources

DL jobs is also non-trivial. Popular DL frameworks, such as TensorFlow [4], PyTorch [5], and the like, support distributed training by exchanging large scale parameter gradients with other nodes per iteration via Parameter Server (PS) [6], [7] or Ring-all-reduce [8]. In aspects of universality and importance, the PS method has some advantages compared with the Ring-all-reduce method. First, PS supports asynchronous and synchronous parameter coordination which makes it more flexible to synchronize the parameters with relaxed consistency than the Ring-all-reduce method. This method can be adaptive to training environments with diversified computing resources very well. However, the Ring-all-reduce method can hardly support the asynchronous training because it needs to synchronize the parameters at each step [7]. Second, PS can manage parameters independently, and it is beneficial for parameter data compression, backup, fault tolerance, and elastic scalability, etc. Thus, we choose the PS as our underlying parameter synchronization platform.

本工作使用数据并行的PS通信架构

In PS, parameter servers store model parameters, and workers are responsible for calculating gradients. Network bandwidth would become bottleneck because workers need to communicate and synchronize with the PS in each training epoch [9]. For the same number of parameter server and worker instances, different instance scheduling/placement plans may lead to quite different execution performance.

PS架构中网络带宽成为瓶颈，不同放置策略会带来不同性能

For example, in our experiments, Fig. 1a shows all four possible scheduling plans for a VGG-16 DL job with one parameter server and two worker instances. In addition, Fig. 1b demonstrates the performance of different resource scheduling plans. It can be seen that the execution time of the same DL job even with the same amount of computing resources under different scheduling plans can reach a difference of up to twice, proving that resource scheduling plans have a significant impact on job execution performance. To improve the job execution performance, good scheduling plans should be aware of the cluster network to make good tradeoffs between the degree of computing parallelism and cross-node data transmissions.

考虑集群网络的调度方案，在计算并行度和跨节点数据传输间做权衡

Existing technologies suffer from the following shortcomings for DL job resource requirement estimation and cluster network-efficient DL job scheduling in various scenarios. First, many existing DL job scheduling platforms [10], [11], [12] require the users to specify job resource amount themselves when submitting DL jobs. However, as we mentioned above, the job resource requirement manually estimated by users is usually quite inaccurate, which would lead to cluster resource underutilization. Second, some DL job scheduling research works [13], [14] [15], [16] do not take the cluster network information into consideration at first. They try to reschedule the jobs when find performance issues during running. This would cause resource waste and increase the overall execution time. Some lousy allocation methods [17], [18] even do not consider network communication factors at all, which would decrease the job execution performance and the system scheduling throughput [19].

To address these issues, this paper proposes Liquid[1], an efficient GPU resource management platform for DL jobs with intelligent resource requirement estimation and scheduling. The main contributions of this paper can be summarized as follows:

分析了挑战，利用回归模型来评估job资源需求

1) First, we analyze the challenges in the DL job scheduling problem and propose a machine learning model based method for DL job resource requirement estimation to avoid over-allocating computing resources.

提出集群网络高效的调度策略，提高job性能，在即时模式使用best-fit算法，在批模式使用组遗传算法

2) Based on that, for a DL job with determined computing resources, we propose a cluster network-efficient scheduling solution to improve the execution performance of DL jobs. The best-fit algorithm and the grouping genetic algorithm are adopted for scheduling DL jobs in immediate and batch modes, respectively.

为了进一步提高GPU资源利用率，提出包括预调度数据传输，细粒度GPU共享和事件驱动通信在内的三个系统级优化

3) Furthermore, to further improve GPU resource utilization, we propose three system-level optimizations, including the pre-scheduling data transmission optimization, the fine-grained GPU sharing optimization, and the event-driven communication optimization.

Experimental results show that the proposed methods can significantly improve performance. Compared with cutting-edge solutions, Liquid can accelerate the job execution speed by 18% on average and shorten the average job completion time (JCT) by 21%. In addition, the pre-scheduling data transmission optimization reduces 44% of GPU resource idle time on average, the fine-grained GPU sharing optimization accelerates running speed of job sequences by 40% on average, and the event-driven communication optimization reduces GPU resource idle time by 13% on average.

2 Related Work

Popular cluster resource management platforms such as Kubernetes [20] can provide mature and generic resource management and scheduling for distributed applications, but do not consider DL job characteristics. Kubeflow [21] mainly focuses on combining Docker containers with DL workflow and depends on Kubernetes [20] to schedule the resources. However, both Kubernetes [20] and Kubeflow [21] can not estimate the resource requirement of ontop applications.

OpenPAI [10] is a cluster resource management platform led by Microsoft, which supports various machine learning applications. PaddlePaddle [11] is an end-to-end resource management and scheduling solution for DL applications. PipeSwitch [12] is a system that enables unused cycles of an inference application to be filled by training or other inference applications. However, the above work [10], [11], [12]

1. Liquid is open sourced at https://github.com/PasaLab/Liquid

are all limited to a specific scenario/framework with few build-in generic scheduling strategies, and also need users to specify the job resource requirements.

CloudScale [13] focuses on re-scheduling during execution. Optimus [16] improves cluster resource utilization by flexible allocation. Pollux [14] improves scheduling performance in DL clusters by adaptively co-optimizing interdependent factors. [15] presents an elastic scheduling algorithm that tames this property well into average job completion time (JCT) minimization. However, they need the job inside information for accurate model prediction, which is very intrusive to user programs and overlooks the characteristics of black box scheduling.

Gandiva [22] has a series of scheduling mechanisms such as suspend-resume, packing, time-slicing, migration, and grow-shrink. AntMan [23] is a DL infrastructure that co-designs cluster schedulers with DL frameworks. They [22], [23] can improve resource utilization a lot, but they have to modify DL frameworks, lacking compatibility.

FfDL [24] provides a scalable and fault-tolerant service for DL applications. Harmony [25] pursues a black-box approach for ML job placement. Tiresias [26] proposes an algorithm based on discretized two-dimensional gittins index and least attained service. Gavel [27] is a heterogeneity-aware scheduler that generalizes a wide range of scheduling policies. Nevertheless, these strategies allocate one GPU to one job, which would waste GPU resources for small jobs.

HiveD [17] is a buddy cell allocation algorithm to ensure sharing safety by efficiently managing dynamic cell binding from virtual private clusters to those in a physical cluster. It focuses on how to share GPUs safely, but ignores how to decide the job resource amount and schedule jobs with consideration of cluster network overhead for high GPU utilization. Salus [18] is proposed to support fine-grained sharing primitives which can be used to implement general and flexible sharing policies, while Liquid focuses on designing the lightweight sharing policies by estimating resource requirement of DL jobs. It means that Salus is somehow orthogonal to Liquid. Moreover, Salus is more intrusive to existing DL computing frameworks than Liquid. Salus needs to modify the source code of DL computing frameworks, such as Tensorflow, and requires extra sharing policy implementation from the users. However, Liquid runs on the standard DL computing frameworks and dispatch appropriate DL jobs to share GPU by estimating job resource requirements. Thus, Liquid brings better compatibility to existing running environment.

3 System Framework Overview

In this section, we introduce the system overview of Liquid, and take a glance at the main components and the basic process of the system.

Fig. 2 presents the overview of Liquid system framework. It includes three layers, namely the DL program layer, the DL job resource estimation and scheduling layer, and the cluster resource layer. The DL program layer lies on the top. At this layer, users submit their DL jobs which include DL program information and parameters such as $num_{batches}$, $batch_{size}$, etc. The cluster resource layer is at the bottom of the system framework. There exist various computing resources that are used to run the scheduled DL jobs.

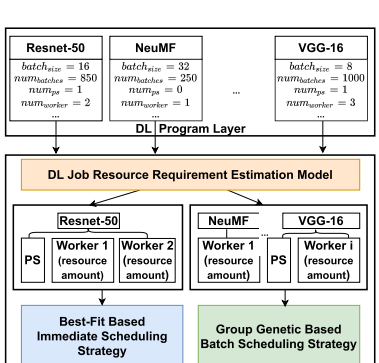


Fig. 2. System framework overview of Liquid.

In the middle of the framework lies the job resource estimation and scheduling layer which is the most work in this paper. The middle layer consists of three key components, namely the DL job resource requirement estimation model (Section 4), the best-fit based immediate scheduling strategy (Section 5.2), and the group genetic based batch scheduling strategy (Section 5.3). After a specific DL program is submitted, Liquid first adopts the job resource requirement estimation model to estimate the appropriate amount of the computing resources for each worker instance of the DL job.

Then, it uses the intelligent cluster network-efficient scheduling methods to allocate computing resource for each DL job. Specifically, there exist two scheduling scenarios. One is the immediate scheduling where jobs appear one by one with long time intervals. Thus, they need to be scheduled to run to reduce scheduling delay and make good use of GPU resources. The other one is the batch scheduling where jobs come quickly and can be grouped together for scheduling to achieve overall performance improvement.

For the above two scenarios, Liquid uses the best-fit based scheduling strategy for immediate scheduling, and the group genetic based strategy for batch scheduling, respectively. Finally, DL jobs are executed on allocated cluster resources, and the resources are released after the job execution. In implementation, besides the PS and worker containers that run the DL jobs, on each computing node we also have a manager daemon container that is responsible for initializing, monitoring, controlling the PS and worker containers through bash shell scripts and RPC methods. After, the DL jobs are scheduled to computing nodes according to the proposed scheduling strategies, the manager daemon on each computing node runs and controls its local PS and worker containers in Docker instances.

即时模式：jobs一个接一个来，间隔长，调度目标为减少调度延迟和提高利用率

批模式：jobs快速到达，进行群组调度，调度目标是提高整体性能，和高利用率(fitness)

4 DL Job Resource Requirement Estimation

In this section, we analyze the job resource requirement and elaborate on its estimation model.

Before scheduling a DL job, we need to know the resource requirement characteristics of the job. Allocating more resources for a DL job can increase its performance, but there is a limit there due to the law of diminishing marginal utility. In other words, there exist an appropriate value of the computing resources amount for a DL job. Allocating

为job分配更多资源能提高性能，但存在递减效应和极限，因此有最佳数量

resource beyond that amount would cause resource wasting. In many DL resource management platforms [20], [21], the job resource requirement is specified by DL programmers manually. This usually would result in resource over allocation due to the programmers' unfamiliarity with the complicated underlying distributed systems.

In fact, for many daily-run routine DL jobs, the appropriate resource amount is very related to the job types and key parameters, such as $batch_{size}$. Therefore, in this section, we design a machine learning based method to estimate the job resource requirement vector to avoid inaccurate user manual settings. The following subsections describe the definition of the job resource requirement vector and how to use historical job running data to estimate that vector.

4.1 Job Resource Requirement Vector

The main job resource requirement factors include computing resources (including memory) and communication resources. Because the DL jobs are usually computation-intensive and communication-intensive, the resource requirement vector can be defined as follow

$$\langle N * GPU, GMem, Bandwidth \rangle$$

It represents GPU computing power, GPU memory, and network bandwidth resources. However, it is challenging for the users, especially for data scientists, to specify the accurate DL job resource vector as they mainly focus on the mathematical theory and the application of DL models. Relying on users' specification will result in lowering job execution performance or a waste of resources.

Therefore, it is meaningful to analyze and estimate resource usage during the job execution. The accurate estimation can help the job to get appropriate resources for efficient execution and allow idle resources to be allocated to other jobs for overall benefits.

4.2 Resource Requirement Vector Estimation Model

In real world, many routine DL jobs are usually executed multiple times: (a) For development and testing purposes, users usually use small batch datasets to verify the effectiveness of proposed DL models; (b) Users usually use the AutoML technology to search for the best hyper-parameter combination with different hyper-parameters and execute them multiple times; (c) For production environment models, they will often get updated with newly generated data.

In multiple DL job executions, usually only the hyper-parameters and the size of the dataset change. These variables determine the resource usage of the job during execution. Besides, the same DL model often shows similar regularities. Therefore, it is possible to estimate the job resource requirement to be scheduled ahead by analyzing these variables with historical data of resource consumption.

For a DL program, two main factors determine the usage of these resources when running the program: one is the hyper-parameters of the training model, including $batch_{size}$, num_{gpus}, and the like; the other is the parameters of the jobs. For example, in the distributed parameter server architecture, the number of parameter server and worker instances significantly affects the DL program's resource usage characteristics at run time.

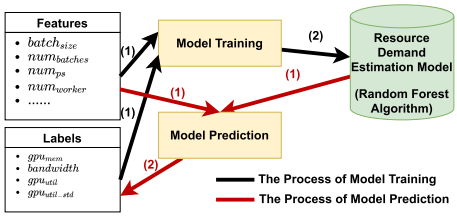


Fig. 3. Resource requirement vector estimation workflow.

The training and prediction process of the job resource requirement vector estimation model is shown in Fig. 3. The input features include the number of parameter server instances num_{ps}, the number of worker instances num_{worker}, dataset size $num_{batches}$, $batch_{size}$, and the like. The output labels of the model are the resource requirement vector, including network communication bandwidth $bandwidth$, GPU computing resource usage gpu_{util}, and GPU memory occupation gpu_{mem}. To better characterize the job characteristics, the model's labels also include GPU computing power utilization standard deviation gpu_{util_std}.

Estimating a continuous result by several continuous or discrete parameters is a typical regression task. We can use machine learning techniques to obtain accurate results efficiently with a small amount of training data collected from previous job running logs. Among the machine learning algorithms, we select the random forest algorithm to estimate the job's resource requirement vector after comparing and analyzing existing regression algorithms. The random forest algorithm uses decision trees as the base learner. It is an integrated algorithm with higher accuracy than most single algorithms. Compared with other algorithms, the random forest algorithm has many advantages in addressing our problem, including not being vulnerable to overfitting, being able to handle the default value, capable of handling higher data dimensions without feature selection, and so on.

5 Network-Efficient DL Job Scheduling

After determining the amount of resource for DL jobs using the estimation solution proposed in Section 4, we discuss how to schedule the resources to the DL jobs according to the cluster network information in this section. As explained in Fig. 1, the execution time of the same DL job using the same amount of resources can also be quite different under different scheduling plans.

Therefore, we focus on designing intelligent DL job scheduling methods under various scenarios to improve GPU resource utilization and job execution performance.

5.1 Problem Analysis

In cluster resource management platforms, we can regard physical machines as a group of abstract resource in the pool. Users apply for resources on demand. In Liquid, users do not have to modify existing DL programs or computing frameworks. We allocate resources by containers which are mapped to underlying physical machines. Container mechanisms such as resource isolation and environment isolation ensure that multiple users can share cluster resources without conflicts. Therefore, the resource scheduling problem in Liquid is to find a mapping from the containers to the idle computing nodes and satisfy the specific resource requirement. The DL

最佳数目和 job 类型以及超参相关

资源需求向量：1) GPU 数目；2) GPU 显存；3) 网络带宽资源

观察：1) 使用小数据集验证模型有效性；2) AutoML 自动重复搜索最佳超参；3) 利用新数据更新生产环境下的模型

许多 jobs 仅超参和数据集大小改变，且相同模型的性能有规律性 -> 可以利用历史资源数据和参数相对大小来评估 job 资源需求

随机森林算法，输入特征为超参，输出表签为资源需求向量 + GPU 算力利用率标准差

在 Liquid，用户不需要修改已有代码或框架，其资源调度问题为：找到从容器到空闲计算节点的映射，并满足特定资源需求

调度关注减少平均 JCT 和 makespan（完成时间，即 max JCT）。总时间考虑了发射时间、训练时间、参数同步时间（主要）和模型保存时间

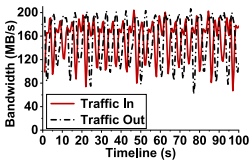


Fig. 4. Network communication traffic of one worker in training VGG-16.

job performance metrics include average job completion time (JCT) and makespan. Our work mainly focuses on reducing the average JCT and makespan since they are the crucial metrics in many DL computing services.

The execution process of the DL model training job consists of N iterations. Each iteration can be further divided into training time T_t and parameter synchronization time T_{sy}. Besides, job execution time also includes launching time T_l and model saving time T_{sa}.

$$T_e = T_l + N * (T_t + T_{sy}) + T_{sa} \tag{1}$$

The process of distributed DL training has frequent synchronization of numerous parameters among instances, which is computation-intensive and communication-intensive.

Some work [9] finds that most of the job execution time for complicated distributed DL model training is spent on synchronizing model parameters, namely T_{sy}. As shown in Fig. 4, in our experiments, we also find that in a distributed VGG-16 model training job, the communication bandwidth of one worker can reach 200 MB/s. Reducing the synchronization time T_{sy} can significantly reduce total execution time T_e. Therefore, in addition to computing resources such as GPU, we also need to consider the network communication bandwidth among containers where job instances are located.

根据 node_1 和 node_2 是否位于相同 rack（机架，与 switch 绑定）/ 相同 domain（域），利用 rack / domain 内的 nodes 数目直接表示 cost（不考虑跨域，通信质量差）

$$Cost(Node_1, Node_2) = \begin{cases} 0, same\ node \\ number\ of\ nodes\ in\ the\ rack, same\ rack \\ number\ of\ nodes\ in\ the\ domain, cross\ rack \end{cases} \tag{2}$$

In a cluster, computing nodes are usually connected in a network with switches in various racks. $Cost(Node_1, Node_2)$ represents the essential communication cost between the computing nodes. It is mainly determined by the delay and average available bandwidth. We use the number of nodes sharing bandwidth resources to describe this. Since the communication quality across network domains is low, we need to avoid schedule different instances/containers of a job to different network domains.

调度策略应减少节点间和 switches 间的通信量

Therefore, we need to design a cluster network-efficient DL job scheduling solution, to reduce the communication traffic across computing nodes and network switches, thereby improving job execution performance.

流程

Fig. 5 shows the high-level workflow of DL job scheduling in Liquid. First, it selects the jobs to be scheduled. Second, it uses the resource demand estimation method proposed in Section 4 to filter out invalid nodes which do not have enough resource. Third, it uses the network-efficient scheduling algorithm proposed in Sections 5.2 and 5.3 for immediate and batch scheduling to generate efficient resource placement plans. Finally, it runs the DL jobs on the resources allocated by the plan.

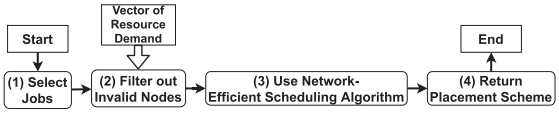


Fig. 5. Workflow of the network-efficient DL job scheduling.

5.2 Best-Fit Algorithm Based Immediate Scheduling Strategy

In this subsection, we discuss how to schedule DL jobs in the immediate mode. In this scenario, DL jobs are submitted to cluster one by one with long time intervals. According to Term 1, to reduce the launching time T_l, Liquid schedules and executes each DL job immediately upon its submission in this scenario. This immediate scheduling strategy guarantees no risk for long-time non-effective waiting time cost. To achieve good execution performance for each DL job, we propose a best-fit algorithm based immediate scheduling strategy that takes the cluster network into account.

Jobs 逐一到达且长间隔，为了减小发射时间，一收到 job_i 就调度并执行

Since the network bandwidth heavily determines the parameter synchronization time, it is mainly considered when generating the resource scheduling plan. First, we evaluate different scheduling plans based on factors such as job characteristics and cluster network. We define the *score* (the samller, the better) that comprises two components: network communication cost $Cost_\kappa$ and computing node load $Fitness_\kappa$, where κ denotes scheduling plan ID. The notations is defined in Table 1. The purpose of scheduling of job i is

定义 score，由通信开销和每节点 load 和的加权和组成，目标是最小化 job_i 的 score

$$\min_{\kappa \in \alpha_i} Score_\kappa, \tag{3}$$

where α_i denotes the set of all logical scheduling plans of job i,

$$Score_\kappa = \lambda * Cost_\kappa + (1 - \lambda) * \sum_{j \in \beta_\kappa} Fitness_j, \tag{4}$$

where β_κ denotes the set of nodes to be used in scheduling plan κ, λ ranges from 0 to 1 which decides the prioritization of these two factors.

Fitness = –（当前请求的 GPUs + 已使用的 GPUs）/ 总 GPUs < 0，这样 min score 等价于 max fitness。

job_i 的通信开销定义为调度策略 k 下，所有 ps 和 worker 间通信 cost 的总和

$$Cost_\kappa = \sum_{m=1}^{numps_\kappa} \sum_{n=1}^{numw_\kappa} Cost(PS_m, W_n) \tag{5}$$

$$Fitness_j = -\frac{ReqGPU_j + UsedGPU_j}{TotalGPU_j} \tag{6}$$

The computing node load mainly considers GPU utilization and tends to distribute instances in a centralized manner

TABLE 1
Notations

Notations	Definitions
κ	One scheduling plan
$Cost_\kappa$	The network cost under scheduling plan κ
$Fitness_\kappa$	The node load under scheduling plan κ
α_i	The set of all logical scheduling plans of job i
$Score_\kappa$	The score under scheduling plan κ
β_κ	The set of nodes to be used in κ
$numps_\kappa$	The number of parameter servers under κ
$numw_\kappa$	The number of workers under κ
γ	A batch of job
κ_i	One scheduling plan of job i

即时模式需要快速给出调度决策，以避免阻塞调度队列造成高调度延迟 -> best-fit algorithm

to reduce fragmentation. The network communication cost characterizes the communication cost between job instances/containers. It aims to reduce cross-node and cross-switch data transmission traffic.

In immediate scheduling, it is necessary to generate the scheduling plan quickly to avoid blocking the scheduling queue and causing high scheduling delays. We use the best-fit algorithm, which can quickly give a good scheduling plan and is suitable for immediate scheduling.

The main idea of the best-fit algorithm is to select the most suitable computing node in each round until all instances/containers are placed. Algorithm 1 shows the steps of the best-fit algorithm during scheduling:

实际上就是贪心，找 score 最小的 node 放

这里不需要真的做 profile，因为 score 中 cost 是根据 nodes 数目算，而 fitness 则根据 -(当前请求 GPUs + 已使用 GPUs) / node 内总 GPUs 进行计算

(a) For each instance of the job, select candidate computing nodes that meet the resource requirement estimated by the solution in Section 4.
(b) Use the evaluation function to calculate the score after placing the instance on the candidate computing node.
(c) Select the candidate computing node with the smallest score, and place the instance on the computing node.
(d) Iterate step (a) - step (c) until all instances are placed, or no remaining instances cannot be placed.

Algorithm 1. Best-Fit Algorithm Based Scheduling Strategy

Input: candidate node *nodes*, ps/worker instance *containers*
Output: placement strategy *placement*

```
placement ← null
for container in containers do
  bestNode ← null
  bestScore ← MAX
  /* select the best node */
  for node in nodes do
    placement.place(node, container)
    /* evaluate tmp placement */
    score ← evaluator.calculate(placement)
    if score < bestScore then
      bestNode ← node
      bestScore ← score
    end if
    /* revert placement */
    placement.remove(node, container)
  end for
  placement.place(bestnode, container)
  nodes.remove(bestnode)
end for
Return placement
```

使用组遗传算法进行求解，将 ps/worker instances 在多节点上分组放置，并以 round-based 的方式在给定时限内迭代优化

Jobs 同时到达，进行群组调度以达到更好的整体性能 -> 组遗传算法

5.3 Grouping Genetic Algorithm Based Batch Scheduling Strategy

Besides the immediate job scheduling studied in Section 5.2, we further research the batch job scheduling in this subsection. This solution is usually used in the scenario where DL jobs come quickly and can be grouped in one batch for scheduling to possibly achieve better overall performance.

We first analyze the difficulty in this problem, then adopt the grouping genetic algorithm to address it. Different DL jobs have different sensitivity to network bandwidth, and the change of job executing time differs a lot with different scheduling plans. For example, Fig. 6 compares the execution time of three DL jobs (VGG-16, ResNet 50, and Inception V3) with different scheduling plans in our experiments. From the experiment, we can find that VGG-16 model training time varies the most under different scheduling plans even with the same amount of resources. This is because that the VGG-16 model training job has more data communication during parameter synchronization and requires higher network bandwidth.

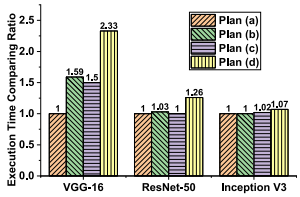


Fig. 6. Example: execution time comparing ratio of different jobs under different plans.

不同 DL jobs 对网络带宽敏感性不同，取决于参数同步时通信量的大小，因此需要在调度策略中考虑 job 的敏感性特性

Therefore, it is necessary to take the characteristics, such as network bandwidth sensitivity, of all DL jobs into consideration when scheduling multiple jobs together for better overall execution performance.

To address the multiple job scheduling problem, in Liquid we propose a batch scheduling strategy based on the grouping genetic algorithm. The batch scheduling goal of a group of jobs γ can be defined as

群组调度目标是最小化组内所有 jobs 的 Score 之和

$$\min \sum_{\kappa_i \in \alpha_i} \sum_{i \in \gamma} Score_{\kappa_i} \tag{7}$$

多维装箱问题，NP-complete

This is a complex multi-dimensional boxing problem which is proved to be NP-complete [28].

To address this problem, we adopt the grouping genetic algorithm (GGA) [28] which is a genetic algorithm heavily modified to suit the structure of grouping problems like the batch job scheduling scenario in this paper. The goal of GGA is to find a good partition for a set. In our problem, each element in the set is a ps/worker instance, while each partition is a computing node that host a bunch of ps/worker instances using containers. By adopting GGA, we can group together the ps/worker instances into different computing nodes, and achieve good overall execution performance. The genetic algorithm gradually optimizes the target function through multiple rounds of iteration and finds better results within the given time constraints.

For example, Fig. 7 shows how to use grouping encoding method to express the resource scheduling problem where J1-W1, J1-W2, J1-W3, J1-W4, and J1-PS1 belong to Job 1 as they have the same prefix J1. Among them, J1-PS1 is the parameter server, others are workers. Similarly, J2-W1, J2-W2, and J2-PS1 belong to Job 2, and J2-PS1 is the parameter server, others are workers. Based on this, we introduce the workflow example as follows.

From the genetic algorithm concept perspective, each chromosome represents a resource scheduling plan, each gene represents a container, and each genome represents a computing node. In crossover and mutation operations, each computing node is operated as a unit. The genetic algorithm generates better individuals by crossover, mutation, and selection of existing ordinary individuals.

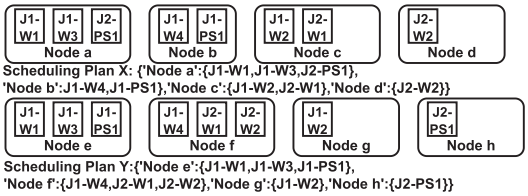


Fig. 7. Example: representation of resource scheduling plan based on grouping encoding (Node means computing node, J1-W1 means Worker Instance-1 which is part of the Job-1).

用 first–fit 和 rand–fit 作为初始调度方案

Therefore, it is necessary to generate an initial population (different solutions), which requires as much diversity as possible, namely covers all genes (genome in this paper). We use the first-fit algorithm and random-fit algorithm to generate the initial scheduling plans.

In the GGA, it is necessary to measure each scheduling plan to reflect the adaptability of chromosomes. This function is called the fitness function. Calculating the fitness function value to determine the degree of one plan reflects the principle of survival of the fittest in natural evolution.

The purpose of scheduling is to reduce communication overhead, thus the fitness function is the negative value for the scheduling goal of this section. The smaller the network communication overhead, the better the fitness. A new variable $NumNU$ is introduced to speed up the convergence of the algorithm on this basis. $NumNU$ is the number of computing nodes required by the scheduling plan so that the algorithm can preferentially choose a resource scheduling plan with fewer nodes when the network communication overhead is close.

max fitness <– min sum(score_i) + min(调度策略需要的 nodes 数目)，以保证 locality 并加快算法收敛

$$GeneticFitness = -\sum_{i \in \gamma} Score_{\kappa_i} - NumNU \tag{8}$$

Besides, if the total resource requirement of a container placed on a computing node exceeds the resource limitation of the node, or some containers have not been placed, the scheduling plan is invalid in that case. Then, the fitness is set to a small number. As a result, invalid scheduling plans are eliminated in the next round of evolution.

选择 fitness 高的，并保留一些普通的，以防收敛到局部最优

After determining the fitness function, it is necessary to set a selection strategy for selecting offspring individuals from the population. This selection strategy preferentially selects individuals with high fitness and retains some ordinary individuals because selecting only individuals with high fitness may cause to converge at a local optimal value and fail to obtain the overall optimal value. The tournament selection method runs multiple rounds of elimination each time and selects the best. In the genetic algorithm, the crossover can concentrate better genes of the parent to the identical offspring, thereby improving the fitness of the offspring.

Fig. 8 demonstrates the crossing operation workflow:

Crossing 操作是把 plan X 某一个 node_c 上的放置照抄到 plan Y (可能之前没有 node_c)，然后把 Y 中相应的容器移到 node_c

(a) Select two plans X and Y according to the selection policy from current plans, and select the intersection point (computing node) and intersection location. Then, add the selected computing node with its containers to the intersection of another scheduling plan to generate a new plan. In this example, it will insert plan X's Node c into plan Y.

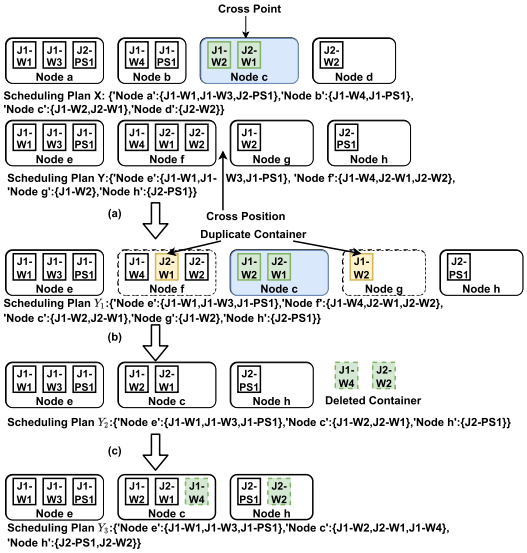


Fig. 8. Example: workflow of the crossover operation in grouping genetic algorithm based scheduling strategy.

(b) After crossover, there may be duplicate computing nodes and containers in the new scheduling plan. Because the basic unit of crossover is a computing node, we delete computing nodes which have duplicate containers. In this example, because J1-W2 and J2-W1 are duplicated, Node f and Node g with duplicate containers should be deleted.

(c) Since these corresponding containers on the deleted nodes are also deleted, they need to be added to the remaining computing nodes again. The deleted containers are added to the remaining computing nodes using the first-fit algorithm. In this example, J1-W4 and J2-W2 are on the deleted nodes, they should be added to the remaining computing nodes. According to the first-fit algorithm, Node c has an empty container, then J1-W4 will be placed on Node c. Similarly, J2-W2 will be placed on node h.

把有容器被移走的 nodes 从 plan Y 中删掉，多出来的容器按 first–fit 进行放置

Crossover can inherit good genes to the next generation, but if the initial population lacks the optimal genes, the optimal individual cannot be obtained. Therefore, mutation operations on existing chromosomes are required. Fig. 9 illustrates the workflow of the mutation operation: It selects a resource scheduling plan Y, then selects a computing node randomly, and deletes the computing node and the containers on that node. After that, according to the first-fit algorithm, it replaces the deleted container on the remaining computing nodes to generate a new scheduling plan. In this example, Node g is deleted, and J1-W2 will be placed on node h according to the first-fit algorithm.

Mutation 操作：把 plan Y 的随机一个节点删掉，相应的容器按 first–fit 放置到其他存在的节点，得到新的 plan Y'
只是 scale down，当放不下时 plan Y' 被标记为非法并被删除

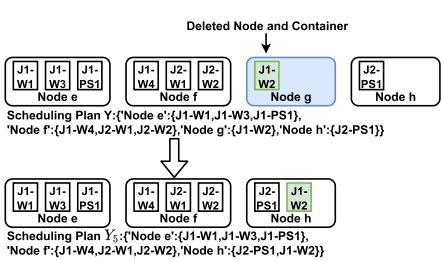


Fig. 9. Example: workflow of the mutation operation in grouping genetic algorithm based scheduling strategy.

After using the above method to generate a final scheduling plan, then the scheduling plan will be actually executed on real nodes through Docker.

6 System Optimizations

To further improve the performance of Liquid, we propose a series of system-level optimizations in system design for various application scenarios.

既然 model 预测可能不准，不如将预测结果作为初始值，根据前几轮的运行结果进行修改（比如蒸馏操作）

6.1 Pre-Scheduling Data Transmission

The entire DL job running process can be divided into three phases: the launching phase, the training phase, and the saving phase. The utilization of GPU during the launching phase and the saving phase is low.

By default, GPU resources will not be allocated to the next job until a job is finished and the resource is released. However, with the proposed pre-scheduling data transmission optimization, the following job will be scheduled to the GPU ahead for preparation when a job comes to the end of execution. When the previous job finishes running and enters the saving phase, the next job will immediately enter the training phase. It is similar to the pipeline technique and can reduce GPU idle period.

Here, we adopt a machine learning model to estimate the job execution time. Similar to the resource requirement vector estimation model, the input characteristics of the model include CPU, memory, GPU, model hyper-parameters, input data size, and the like. The output labels of the regression model include launching phase time cost T_l, training phase time cost T_t, and saving phase time cost T_{sa}.

6.2 Fine-Grained GPU Sharing

There often exist many model training or inference tasks for development and testing purposes in a lot of real-world scenarios. They usually have a low requirement for GPU resources, but their exclusive use of GPU will cause a non-negligible waste of resources.

If we can guarantee job execution performance with the premise of correctness, multiple jobs scheduled to the same GPU card can utilize GPU resource more efficiently. It is necessary to analyze the GPU resource usage of the job to ensure that multiple jobs do not exceed the upper limit of the total amount of resources when sharing one GPU card. We use the job resource requirement estimation model to fulfill this. The DL program's utilization rate of resources also has a certain regularity and short cyclical. In a time frame, the utilization rate is smooth. Therefore, we choose the average utilization of GPU computing resources and the maximum occupancy of GPU memory resources to represent the estimated value of the job resource requirement.

Due to the accuracy issue of the model, the job resource occupation may still exceed the upper limit during actual scheduling, which will affect the regular operation of other jobs. To prevent the job resource occupation from exceeding resource limitation, we can limit the maximum job GPU memory resource usage by only specifying the GPU memory usage parameter in the DL programs in this scenario.

6.3 Event-Driven Communication

The reuse of the GPU resources during DL job scheduling includes the following steps: (a) The previous job is completed; (b) The local resource collector reports the latest node resource information in the next *heartbeat*; (c) The scheduler allocates resources to the next job; (d) The following job is scheduled to this node.

It can be seen that since the heartbeat messaging mechanism is periodic, the computing resources can hardly be fully used in all time. Instead of waiting for the following heartbeat, if the scheduler can be immediately notified when the previous job is completed, the idleness of related resources can be reduced to a certain extent.

In this optimization, the heartbeat mechanism is improved to allow nodes to send packets immediately to deliver that mission-critical event information about resource state changes. After optimization, the local resource collector checks and reports resource usage regularly and tracks the job execution status in real-time.

7 Evaluation

In this section, we conduct extensive experiments to evaluate the performance of Liquid.

7.1 Experimental Setup

We conduct experiments in a Alibaba Cloud cluster with NVIDIA Tesla T4 GPU and a local cluster with NVIDIA Tesla K80 GPU to evaluate Liquid under both high-end and low-end computing environments. The local cluster is composed of 1 manager node and 5 computing nodes. Each node has 24 Intel(R) Xeon(R) E5-2630 v2 CPUs, 4 NVIDIA Tesla K80 GPUs, 188GB DDR3 memory, and two 2TB SAS (Ext4) disks. They are connected via a 10Gbps Ethernet. And, the OS version is CentOS Linux release 7.6.1810 (Core). NVIDIA Driver 410.129, CUDA 10.0, Docker 19.03, Kubernetes v1.20.2, Kubeflow v1.1.0, and related software is deployed in the K80 cluster. The Alibaba Cloud cluster consists of 20 ecs.gn6i-c40g1.12xlarge VMs. One machine acts as the manager node and computing node, the other are all computing nodes. Each node has 48 virtual Intel Xeon(Skylake) Platinum 8163 CPUs, 2 NVIDIA Tesla T4 GPUs, 186GB DDR4 memory, one 500GB ESSD. They are connected via a 15Gbps Ethernet. And the OS version is Alibaba Cloud Linux 2.1903 LTS. NVIDIA Driver 418.181, CUDA 10.1, Docker 19.03, Kubernetes v1.18.8, Kubeflow v1.1.0, and related software is deployed on the NVIDIA Tesla T4 cluster. MySQL 5.6, Redis 5.0.7, Gitea 1.12, HDFS 2.7.4, Nginx 1.17, Python 3.6, TensorFlow 1.14/2.1 are run in Docker containers on these two clusters.

Workloads and Data Sets. We use the following five representative DL jobs with typical data sets in our experiments.

1) *CNN-MNIST:* It uses Convolutional Neural Networks (CNN) [29] with MNIST dataset [2].
2) *NeuMF:* It uses Neural Matrix Factorization model (NeuMF) [30] with MovieLens 20M dataset [31].
3) *ResNet-50:* It uses ResNet-50 [32] with ImageNet dataset [33], [34].
4) *Inception V3:* It uses Inception V3 [35] with ImageNet dataset [33], [34].
5) *VGG-16:* It uses VGG-16 [36] with ImageNet dataset [33], [34].

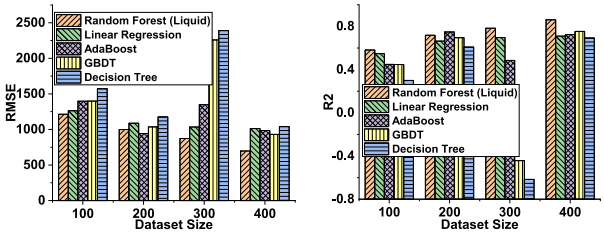
前一个 job 刚完成训练并进入保存步骤，后一个 job 就立即开始训练，减少 GPU 空闲

该优化的实现依赖于利用一个模型预测 job 各阶段时间 (和 Optimus 类似，不靠谱)

将低资源利用率的多个 jobs 放在相同 GPU 上混布

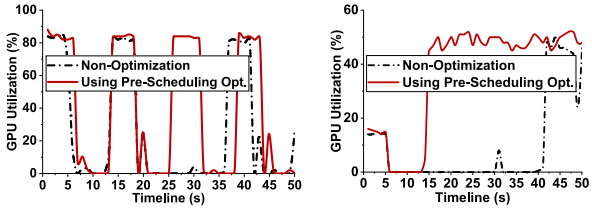
依赖于利用模型预测 job 的 GPU 使用情况，并将平均 GPU 利用率和最大占用 GPU 显存作为预测值

调度器周期性检查资源状态 (heartbeat) 会导致空闲，因此由 local resource collector 实时收集状态，并立即发送信号告知调度器 这个靠谱，在我们的工作中可以用



(a) RMSE (smaller is better) (b) R2 (larger is better)

Fig. 10. Accuracy (RMSE and R2) of various regression algorithms for job resource amount estimation (Liquid uses random forest).



(a) CNN-MNIST (b) NeuMF

Fig. 11. GPU utilization comparison between using and not using pre-scheduling data transmission optimization on NVIDIA Tesla K80 GPU.

Metrics: Our experiments involve the following four metrics from different aspects.

1) *Job Completion Time (JCT):* Job completion time includes three parts, namely the waiting time of the job in the scheduling queue, the system scheduling time of resource allocation with container startup during the scheduling process, and the actual execution time of the job.
2) *Makespan:* The makespan is total time cost of the entire batch of jobs. It is defined as $Makespan = t_{end} - t_{begin}$, where t_{end} represents the time when all jobs finish running, t_{begin} represents the time the first job begins running.
3) *Root Mean Square Error (RMSE) and Coefficient of Determination (R2):* The root mean square error is defined as $RMSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$ and the coefficient of determination is defined as $R2 = \frac{\sum_{i=1}^{N}(f_i-\bar{y})^2}{\sum_{i=1}^{N}(y_i-\bar{y})^2}$, where N represents the number of samples, f_i represents the predicted value of sample i, y_i represents the actual value of sample i, $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$.

Comparing Systems: Our experiments involve the following three cutting-edge systems, Kubernetes [20], Kubeflow [21], and HiveD [21], for performance comparison.

7.2 Accuracy of Job Resource Requirement Estimation Model

In this subsection, we evaluate the accuracy of proposed job resource requirement estimation model in this paper. We run DL jobs to match the appropriate resource amount offline. The pre-run offline DL jobs are generated by combining the above DL jobs with random computing resource requirement (number of instances, number of GPUs required, and the like) and random model parameters (batch size and the like). The collected pre-run job execution log information is used as the training data for the estimation model.

After accumulating a certain amount of job scheduling data, we randomly divide the historical scheduling data, including job meta-information and actual resource requirement vector, into a training set and a test set. The size of the test set is 50. We increase the number of jobs in the training set gradually to estimate the resource requirement of the jobs in the test set, and then check the accuracy of the model.

Figs. 10a and 10b show the RMSE and the R2 scores of various regression algorithms using the same data set collected by the above way. The value of RMSE is the smaller the better. The value of R2 is the larger the better.

As can be seen from Figs. 10a and 10b, with the size of the training set increases, the accuracy of all the models also gets continuously improved. The R2 index of the random forest algorithm is close to 0.9, which means that the random forest algorithm based estimation model for job resource requirement has the highest accuracy.

Overall, the accuracy of each algorithm is on the rise with the increasing amount of data. Among them, the random forest algorithm [37] adopted in Liquid has the highest accuracy. Moreover, it enhances stability, has no large fluctuations, and its R2 index is up to 0.86. Compared with the random forest algorithm used in our system, the decision tree algorithm and the gradient boosting decision tree algorithm (GBDT) both have significant fluctuations, and the adaBoost algorithm has apparent fluctuations and is overfitting in the job resource estimation scenarios. The overall performance of linear regression has improved steadily, but it is challenging to achieve high accuracy due to the simplicity of the model.

7.3 Effectiveness of System Optimizations

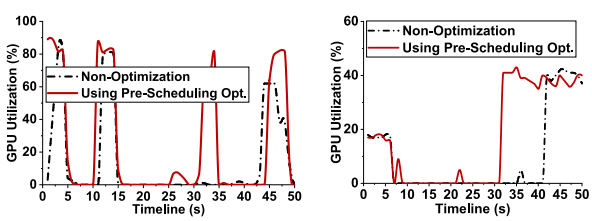
In this subsection, we evaluate the effectiveness of the system-level optimizations proposed in Liquid.

7.3.1 Pre-Scheduling Data Transmission Optimization

In this group of experiments, we evaluate the effectiveness of the pre-scheduling data transmission optimization proposed in Section 6.1. We use two job sequences as our workloads, namely 2 CNN-MNIST jobs, and 2 NeuMF jobs.

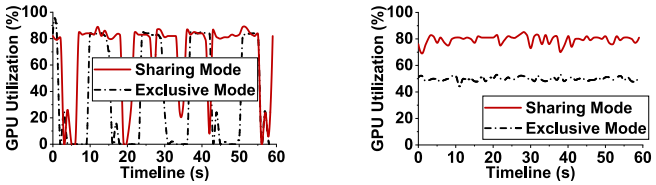
Figs. 11 and 12 show the change of GPU utilization during job execution when using and disabling the pre-scheduling data transmission optimization. The experiments of Figs. 11a and 11b are conducted on NVIDIA Tesla K80 GPU, while the experiments of Figs. 12a and 12b are conducted on NVIDIA Tesla T4 GPU.

We take the CNN-MNIST job sequence running on K80 GPU as an example to discuss the evaluation results. As shown in Fig. 12a, when not using the pre-scheduling data transmission optimization, GPU utilization is nearly idle between the 15th and 43th seconds, which corresponds to



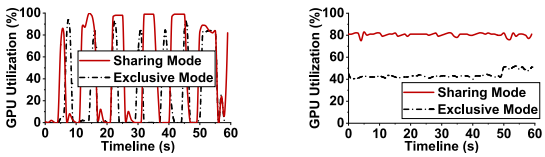
(a) CNN-MNIST (b) NeuMF

Fig. 12. GPU utilization comparison between using and not using pre-scheduling data transmission optimization on NVIDIA Tesla T4 GPU.



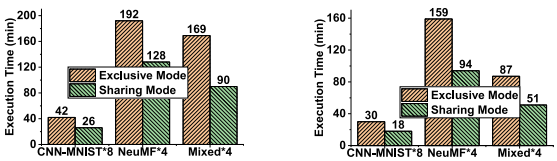
(a) CNN-MNIST (b) NeuMF

Fig. 13. GPU utilization comparison between using and not using fine-grained sharing optimization on NVIDIA Tesla K80 GPU (the former is Sharing Mode, the latter is Exclusive Mode).



(a) CNN-MNIST (b) NeuMF

Fig. 14. GPU utilization comparison between using and not using fine-grained sharing optimization on NVIDIA Tesla T4 GPU (the former is Sharing Mode, the latter is Exclusive Mode).



(a) NVIDIA Tesla K80 GPU (b) NVIDIA Teala T4 GPU

Fig. 15. Job execution time performance comparison between using and not using fine-grained sharing optimization on different GPU cards (the former is Sharing Mode, the latter is Exclusive Mode).

duration of 28 seconds. However, after using the proposed optimization, the GPU is effectively utilized between the 30th and 35th seconds, which means that the GPU idleness period is reduced by up to 18%. Similarly, the idle time of GPU resources is reduced by about 50%, 77%, and 30% in Figs. 11a, 11b, and 12b respectively.

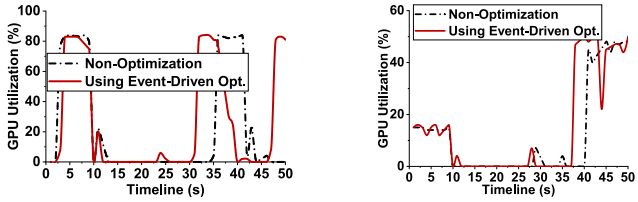
To conclude, these experiments have proved that the pre-scheduling data transmission optimization is effective. This is because it can reduce the idle time of GPU resources by launching the coming job ahead to the computing node without affecting the performance of the previous job.

7.3.2 Fine-Grained GPU Sharing Optimization

In this part, we evaluate the effectiveness of the fine-grained GPU sharing optimization proposed in Section 6.2. The CNN-MNIST and NeuMF DL jobs are used as workloads in this group of experiments.

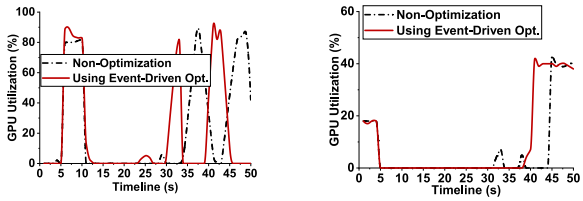
Figs. 13 and 14 show the GPU utilization of the CNN-MNIST and NeuMF job execution in exclusive and sharing modes. The experiments in Figs. 13a and 13b are conducted on NVIDIA Tesla K80 GPU, while the experiments in Figs. 14a and 14b are run on NVIDIA Tesla T4 GPU.

The GPU utilization of CNN-MNIST changes periodically. There is a clear wave in the utilization, thus where the total utilization is less than 50% on average. In contrast, the GPU utilization of NeuMF is stable but keeps less than 50% on average. Therefore, both of them can run two same jobs on one GPU to improve the resource utilization. As can be



(a) CNN-MNIST (b) NeuMF

Fig. 16. GPU utilization comparison between using and not using event-driven communication optimization on NVIDIA Tesla K80 GPU.



(a) CNN-MNIST (b) NeuMF

Fig. 17. GPU utilization performance comparison between using and not using event-driven communication optimization on NVIDIA Tesla T4 GPU.

seen from the red lines in Figs. 13 and 14, when 2 CNN-MNIST jobs or 2 NeuMF jobs share the same GPU, the GPU utilization is obviously improved compared with running them in the exclusive mode.

Furthermore, we analyze the total execution time when using and not using the fine-grained GPU sharing optimization, namely the exclusive mode and the sharing mode. We execute three sequences of jobs, namely 8 CNN-MNIST jobs (CNN-MNIST*8), 4 NeuMF jobs (NeuMF*4), 4 mixed jobs (Mixed*4, namely 2 CNN-MNIST jobs with 2 NeuMF jobs in random order), under the same environment.

Figs. 15a and 15b present the total executing time of multiple jobs in the sharing mode and the exclusive mode. It can be seen that the total execution time of multiple jobs in the sharing mode is reduced by 38%, 33%, and 47% compared with that in the exclusive mode on K80 GPU in Fig. 15a. And the execution time of multiple jobs in the sharing mode is reduced by 40%, 41%, and 41% compared with that in the exclusive mode on T4 GPU in Fig. 15b. The overall DL job execution performance is improved by increasing the GPU utilization. It proves that the fine-grained GPU sharing optimization proposed in Section 6.2 is effective.

7.3.3 Event-Driven Communication Optimization

This subsection evaluates the effectiveness of event-driven communication optimization proposed in Section 6.3. We record the GPU resource utilization in a timeline when executing CNN-MNIST jobs and NeuMF jobs in sequence.

Figs. 16 and 17 demonstrate the change of GPU utilization over time using the fixed heartbeat mechanism (interval is 5s in our setting) and event-driven communication optimization, respectively. The experiments in Figs. 16a and 16b are conducted on K80 GPU, and the experiments in Figs. 17a and 17b are run on T4 GPU.

As shown in Fig. 17a, when not using event-driven communication optimization, GPU utilization is nearly idle between the 10th and 35th seconds, which corresponds to duration of 25 seconds. However, after using the proposed optimization, the GPU is effectively utilized between the

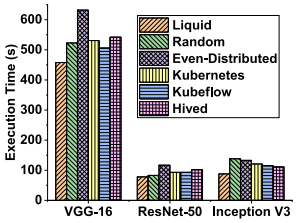


Fig. 18. Execution performance comparison among Liquid, baseline solutions and cutting-edge systems in immediate scheduling scenario.

31th and 35th seconds, which means that GPU idleness period is reduced by up to 16%. Similarly, the idle time of GPU resources is reduced by about 15%, 10%, and 10% in Figs. 16a, 16b, and 17b respectively.

To sum up, the experimental results show that the event-driven communication optimization can effectively reduce the idle time of GPU during executing jobs. Moreover, the optimization is suitable for many scenarios.

7.4 Overall Performance Comparison

Finally, this set of experiments evaluate the overall performance of Liquid with comparison to related cutting-edge systems and baseline solutions.

We use ResNet-50, VGG-16, and Inception V3 DL jobs as the workloads in the following experiments. We also use Kubernetes [20], Kubflow [21], HiveD [17], and baseline strategies such as the random strategy and the even-distributed strategy for performance comparison. Kubernetes [20], Kubflow [21], and HiveD [17] are widely-used cutting-edge job resource management and scheduling platforms. And, the random strategy allocates the instance to computing node randomly, the even-distributed strategy tries to place the instances to each computing node averagely.

There are two different scenarios in DL job scheduling. One is the immediate scheduling scenario where jobs appear one by one with long time intervals. The other is the batch scheduling scenario where jobs come quickly and can be grouped together for scheduling.

We first analyze the job execution performance in immediate scheduling scenarios. Fig. 18 demonstrates the job execution time using different strategies, including Liquid (immediate mode), random strategy, even-distributed strategy, and the default scheduling policies in Kubernetes [20], Kubeflow [21], and HiveD [17].

The experimental results show that the best-fit algorithm based immediate job scheduling strategy in Liquid performs best among all solutions. It accelerates the average job execution speed by 14%, 32%, 19%, 16% and 20% compared with the random strategy, the even-distributed strategy, default scheduling policies in Kubernetes, Kubeflow, and Hived respectively. The reason is that the immediate job scheduling strategy in Liquid can reduce the communication cost a lot by taking both computing and communication into account when deciding the resource scheduling plans.

Then, we discuss the job execution performance in batch scheduling scenarios. The next set of experiments evaluates the performance of the grouping genetic algorithm based batch job scheduling strategy in Liquid. In addition, to evaluate the effectiveness and scalability of our proposed scheduling solution Liquid under various scale of workload and computing resource, we scale up the workload with the increasing computing nodes and GPU cards proportionally. The overall performance comparison experiments are conducted on various number of computing nodes, including 5 nodes, 10 nodes, 15 nodes, and 20 nodes as shown in Fig. 19. Each computing node is equipped with 2 NVIDIA TESLA T4 GPU cards. The workload in experiments with 5 computing nodes is a set of deep learning training jobs, which include 4 ResNet-50 jobs, 2 Inception V3 jobs, and 2 VGG-16 jobs. Correspondingly, the workload running on 10 computing nodes is a set of DL jobs consisting of 8 ResNet-50 jobs, 4 Inception V3 jobs, and 4 VGG-16 jobs. And, so do the workload size on 15 nodes and 20 nodes increase.

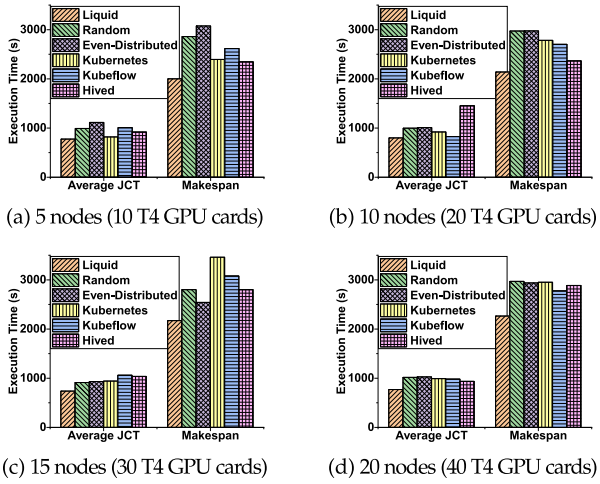


Fig. 19. Execution performance comparison among Liquid, baseline solutions and cutting-edge systems in batch scheduling scenario with different numbers of nodes.

As shown in Figs. 19a, 19b, 19c, and 19d, the average JCT and makespan for the grouping genetic algorithm based batch job scheduling strategy in Liquid are always the shortest. Take Fig. 19d as an example, Liquid reduces the makespan by 24%, 23%, 23%, 18% and 22% compared with the random strategy, even-distributed strategy, and default scheduling policies in Kubernetes, Kubeflow, and Hived, respectively. In addition, Liquid reduces the average JCT by 24%, 25%, 22%, 22%, and 18% compared with the random strategy, even-distributed strategy, and default scheduling policies in Kubernetes, Kubeflow, and Hived, respectively.

Liquid works well with increasing workloads and computing resources. The workload running time of Liquid stays stable when scaling up workloads and computing resources together. Thus, it achieves good scalability. The reason why Liquid achieves the best performance compared with the other methods is that Liquid schedules multiple jobs in batches to generate a scheduling plan from the global view for improving the overall resource utilization, while other solutions usually only satisfy few jobs or other aspects, resulting in local optimal scheduling.

8 Conclusion and Future Work

In this paper, we propose Liquid, an efficient GPU resource management platform with intelligent resource estimation and scheduling strategies for DL applications on distributed GPU clusters. First, we propose a job resource requirement estimation model based on the machine learning method

to analyze job resource requirement. Based on this, we propose cluster network-efficient scheduling strategies to improve DL job execution performance for both immediate and batch modes respectively. Finally, we build the prototype system with several system-level optimizations.

Experimental results demonstrate that Liquid accelerates the average job execution speed by 18% and shortens the average job completion time (JCT) by 21% compared with the cutting-edge systems. Moreover, the proposed optimizations are effective under various scenarios.

In the future, we plan to further study the online job resource requirement estimation method to make the scheduling strategy works in more dynamic environments. Furthermore, we are about to researching how to reschedule jobs to different nodes with the preemption mechanism.

Acknowledgments

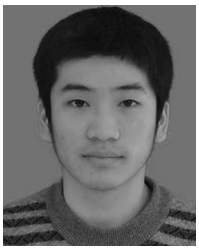
We thank the anonymous reviewers and editors and for their insightful comments.

References

[1] J. Bergstra *et al.*, "Theano: A CPU and GPU math expression compiler," in *Proc. 9th Python Sci. Comput. Conf.*, 2010, pp. 1–7.
[2] "MNIST," 1998. Accessed: May 10, 2021. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[3] M. Jeon *et al.*, "Analysis of large-scale multi-tenant GPU clusters for DNN training workloads," in *Proc. 28th USENIX Annu. Tech. Conf.*, 2019, pp. 947–960.
[4] "TensorFlow," 2015. Accessed: May 10, 2021. [Online]. Available: https://github.com/tensorflow/tensorflow/
[5] "PyTorch," 2016. Accessed: May 10, 2021. [Online]. Available: https://github.com/pytorch/pytorch/
[6] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Oper. Syst. Des. Implementation*, 2014, pp. 583–598.
[7] Y. Jiang *et al.*, "A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters," in *Proc. 14th USENIX Symp. Oper. Syst. Des. Implementation*, 2020, pp. 463–479.
[8] "Bringing HPC Techniques to Deep Learning," 2021. Accessed: May 15, 2021. [Online]. Available: https://github.com/baidu-research/baidu-allreduce
[9] J. H. Park *et al.*, "Accelerated training for CNN distributed deep learning through automatic resource-aware layer placement," *Comput. Res. Repository*, vol. 58, no. 3, pp. 1–13, 2019.
[10] "OpenPAI," 2017. Accessed: May 20, 2021. [Online]. Available: https://github.com/microsoft/pai/
[11] S. Liu *et al.*, "A unified cloud platform for autonomous driving," *Computer*, vol. 50, no. 12, pp. 42–49, 2017.
[12] Z. Bai *et al.*, "PipeSwitch: Fast pipelined context switching for deep learning applications," in *Proc. 14th USENIX Symp. Oper. Syst. Des. Implementation*, 2020, pp. 499–514.
[13] Z. Shen *et al.*, "CloudScale: Elastic resource scaling for multi-tenant cloud systems," in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011, pp. 1–14.
[14] A. Qiao *et al.*, "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," in *Proc. 15th USENIX Symp. Oper. Syst. Des. Implementation*, 2021, pp. 1–18.
[15] C. Hwang *et al.*, "Elastic resource sharing for distributed deep learning," in *Proc. 18th USENIX Symp. Netw. Syst. Des. Implementation*, 2021, pp. 721–739.
[16] Y. Peng *et al.*, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. 13th Eur. Conf. Comput. Syst.*, 2018, pp. 1–14.
[17] H. Zhao *et al.*, "HiveD: Sharing a GPU cluster for deep learning with guarantees," in *Proc. 14th USENIX Symp. Oper. Syst. Des. Implementation*, 2020, pp. 515–532.
[18] P. Yu and M. Chowdhury, "Salus: Fine-grained GPU sharing primitives for deep learning applications," *Proc. 3rd Conf. Mach. Learn. Syst.*, 2020, pp. 1–15.
[19] L. Luo *et al.*, "Plink: Discovering and exploiting datacenter network locality for efficient cloud-based distributed training," in *Proc. 3rd Conf. Mach. Learn. Syst.*, 2020, pp. 1–16.
[20] "Kubernetes," 2014. Accessed: May 26, 2021. [Online]. Available: https://kubernetes.io/
[21] "Kubeflow," 2018. Accessed: May 26, 2021. [Online]. Available: https://www.kubeflow.org/
[22] W. Xiao *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *Proc. 13th USENIX Symp. Oper. Syst. Des. Implementation*, 2018, pp. 595–610.
[23] W. Xiao *et al.*, "Antman: Dynamic scaling on GPU clusters for deep learning," in *Proc. 14th USENIX Symp. Oper. Syst. Des. Implementation*, 2020, pp. 533–548.
[24] K. Jayaram *et al.*, "FfDL: A flexible multi-tenant deep learning platform," in *Proc. 20th Int. Middleware Conf.*, 2019, pp. 82–95.
[25] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. 38th IEEE Int. Conf. Comput. Commun.*, 2019, pp. 505–513.
[26] J. Gu *et al.*, "Tiresias: A GPU cluster manager for distributed deep learning," in *Proc. 16th USENIX Symp. Netw. Syst. Des. Implementation*, 2019, pp. 485–500.
[27] D. Narayanan *et al.*, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *Proc. 14th USENIX Symp. Oper. Syst. Des. Implementation*, 2020, pp. 481–498.
[28] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *J. Heuristics*, vol. 2, no. 1, pp. 5–30, 1996.
[29] J. Bouvrie, "Notes on convolutional neural networks," Tech. Rep. Cogprints-5869, 2006.
[30] H.-J. Xue *et al.*, "Deep matrix factorization models for recommender systems," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 3203–3209.
[31] "ml-20m," 2015. Accessed: May 26, 2021. [Online]. Available: http://files.grouplens.org/datasets/movielens/
[32] K. He, *et al.*, "Deep residual learning for image recognition," in *Proc. 29th IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[33] "CIFAR-10," 2009. Accessed: May 26, 2021. [Online]. Available: http://www.cs.toronto.edu/ kriz/cifar-10-binary.tar.gz/
[34] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," Tech. Rep. SemanticScholar-18268744, 2009.
[35] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. 29th IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.
[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015, pp. 1–14.
[37] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

Rong Gu received the PhD degree from Nanjing University, Nanjing, China, in 2016. He is an associate research professor in Nanjing University. His research interests include parallel and distributed computing, big data systems. His research papers have been published in many conference and journals, including *IEEE Transactions on Parallel and Distributed Systems*, IEEE ICDE, IEEE IPDPS, IEEE ICPP, *Journal of Systems Architecture*, *Parallel Computing*, *Journal of Parallel and Distributed Computing* and SPE.

Yuquan Chen received the BS degree in Southeast University, Nanjing, China. He is currently working toward the MS degree in Nanjing University, Nanjing, China. His research interests include distributed storage system and parallel algorithms.



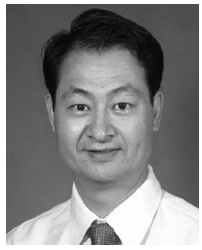
Shuai Liu received the BS degree in Jilin University, Changchun, China and the MS degree in Nanjing University, Nanjing, China. His research interests include distributed storage system and parallel algorithms.

Haipeng Dai received the PhD degree from Nanjing University, Nanjing, China, in 2014. He is currently an associate professor with Nanjing University. His research interests include data mining and mobile computing. He has published research papers in VLDB, IEEE ICDE, *SIGMETRICS*, IEEE ICDCS, IEEE ICNP, *IEEE Transactions on Parallel and Distributed Systems*, and *IEEE Transactions on Mobile Computing*. He received Best Paper Award (ICNP'15), Best Paper Award Runner-up (SECON'18), and Best Paper Award Candidate (INFOCOM'17).

Guihai Chen received the PhD degree from the University of Hong Kong, Hong Kong, in 1997. He is currently a professor and deputy chair of the Department of Computer Science, Nanjing University, China. He had been invited as a visiting professor by many foreign universities including Kyushu Institute of Technology, University of Queensland, and Wayne State University, USA. His research interests include parallel computing, high-performance computing and data engineering.

Zhang Kai received the MS degree in computer science from the Beijing Institute of Technology, Beijing, China, in 2006. He is currently a staff engineer of Alibaba Cloud. His main technical expertise includes distributed computing and cloud systems.

Yang Che received the MS degree in operational research and cybernetics from the University of Science and Technology Beijing, Beijing, China, in 2008. He is currently a staff engineer of Alibaba Cloud. His main technical expertise includes distributed computing and cloud systems.



Yihua Huang received the PhD degree in computer science from Nanjing University, Nanjing, China. He is a professor with computer science department and State Key Laboratory for Novel Software Technology, Nanjing University, China. His main research interests include parallel and distributed computing and big data parallel processing.