Learning-Driven Interference-Aware Workload Parallelization for Streaming Applications in Heterogeneous Cluster

Haitao Zhang, Xin Geng, and Huadong Ma, *Senior Member, IEEE*

Abstract—In the past few years, with the rapid development of CPU-GPU heterogeneous computing, the issue of task scheduling in the heterogeneous cluster has attracted a great deal of attention. This problem becomes more challenging with the need for efficient co-execution of tasks on the GPUs. However, the uncertainty of heterogeneous cluster and the interference caused by resource contention among co-executing tasks can lead to the unbalanced use of computing resource and further cause the degradation in performance of computing platform. In this article, we propose a two-stage task scheduling approach for streaming applications based on deep reinforcement learning and neural collaborative filtering, which considers fine-grained task division and task interference on the GPU. Specifically, the Learning-Driven Workload Parallelization (LDWP) method selects an appropriate execution node for the mutually independent tasks. By using the deep Q-network, the cluster-level scheduling model is online learned to perform the current optimal scheduling actions according to the runtime status of cluster environments and characteristics of tasks. The Interference-Aware Workload Parallelization (IAWP) method assigns subtasks with dependencies to the appropriate computing units, taking into account the interference of subtasks on the GPU by using neural collaborative filtering. For making the learning of neural network more efficient, we use pre-training in the two-stage scheduler. Besides, we use transfer learning technology to efficiently rebuild task scheduling model referring to the existing model. We evaluate our learning-driven and interference-aware task scheduling approach on a prototype platform with other widely used methods. The experimental results show that the proposed strategy can averagely improve the ~~throughout~~ for distributed computing system by 26.9 percent and improve the GPU resource utilization by around 14.7 percent.

Index Terms—Parallel computing, heterogeneous computing, task scheduling, deep reinforcement learning, neural collaborative filtering, interference aware

1 Introduction

During the last few years, CPU-GPU heterogeneous computing has been widely used to handle compute-intensive problems [1], [2], [3], such as data mining, video processing, and machine learning. Moreover, heterogeneous cluster provides high computing power for the above compute-intensive tasks. A critical issue in the heterogeneous cluster is how to schedule multiple tasks to achieve optimal system throughput and improve resource utilization. This problem becomes more challenging with multitasking concurrent execution supported by the modern GPUs [4], [5].

To improve system throughput and resource utilization in heterogeneous cluster, various scheduling techniques have been proposed. Several recent work [6], [7], [8], [9] has proposed the task scheduling methods for hybrid CPU-GPU devices to efficiently use heterogeneous resources for optimal performance. However, the above work only considers the task scheduling on hybrid CPU-GPU devices, and lacks the consideration of task scheduling in heterogeneous clusters. Although several researches [10], [11], [12], [13] have proposed some optimized task scheduling approaches for distributed computing platform, most of them do not consider the heterogeneity of the computing node. In [13], the authors proposed a two-stage resource provisioning and task scheduling method based on deep reinforcement learning, which considers task scheduling on both data center level and cluster level. However, the above work does not take into account the full utilization of GPU resources.

With the development of GPU architecture, modern GPUs provide a basic spatial multitasking framework to support concurrent execution of different tasks, which helps to improve overall system throughput. However, it can be difficult to effectively schedule concurrent tasks on a GPU due to the task interference, which may leads to resource contention and load imbalance. Several work [14], [15], [16], [17] has proposed interference-aware scheduling methods on GPU. In [14], the authors proposed an interference-driven cluster management framework to predict and handle interference for GPU-based heterogeneous clusters. Novel scheduling policies were proposed in [15] to achieve high GPU utilization and improve system throughput. In [16], the authors proposed a novel dynamical application slowdown estimation model for concurrent GPGPU applications to dynamically detect system unfairness. Mystic [17] is an interference-aware scheduler using collaborative filtering for concurrent execution applications on GPU-based

• *The authors are with the Beijing Key Lab of Intelligent Telecomm. Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing 100876, China. E-mail: {zht, gengxin, mhd}@bupt.edu.cn.*

Manuscript received 1 Feb. 2019; revised 20 May 2020; accepted 29 June 2020. Date of publication 13 July 2020; date of current version 29 July 2020. (Corresponding author: Haitao Zhang.) Recommended for acceptance by O. Rana. Digital Object Identifier no. 10.1109/TPDS.2020.3008725

cluster. However, the above work only considers task scheduling on GPU, and does not take into account the heterogeneity of hybrid CPU-GPU architecture.

The learning-driven task parallelization strategy presented in our previous work [18] considers fine-grained task division and dependencies of each subtask, which can achieve high system throughput. However, the previous strategy lacks considering the interference caused by resource contention between co-executing tasks on the GPU, which will lead to underutilization of system resources. To handle the issues in the previous work, we propose a novel task scheduling approach which simultaneously utilizes deep reinforcement learning based cluster scheduler and deep collaborative filtering based node scheduler to optimize resource utilization and achieve high task throughput. In addition, the task scheduling model can be easily transferred to adapt to the flexible changes of the underlying physical platform. The major contributions of our work are summarized as follows.

- We define the streaming application parallelization problem with fine-grained task division and task interference detection in the CPU-GPU heterogeneous cluster.
- We propose a two-stage task scheduling approach for streaming applications in CPU-GPU heterogeneous cluster, which includes a Learning-Driven Workload Parallelization (LDWP) method based on Deep Reinforcement Learning (DRL) and a Interference-Aware Workload Parallelization (IAWP) method based on Neural Collaborative Filtering (NCF). The objective is to maximize the cluster task throughput in long term through automatically generating the best scheduling actions and improve the resource utilization by considering the task interference. The LDWP method can select the current optimal execution node for tasks, and the IAWP method can assign fine-grained subtasks to appropriate heterogeneous computing unit and distribute appropriate subtasks to GPU to reduce interference among subtasks, thereby avoiding resource contention between co-executing subtasks on the GPU. For making the learning of neural network more efficient, we use pre-training in the two-stage task scheduling approach.
- We propose a transfer learning based scheduling generalization method for addressing the scheduling model relearning issue. For cluster-level task scheduling, the parameter-transfer method is used to quickly obtain an effective scheduling network model for the current environment based on the previously constructed model when the configurations of the heterogeneous cluster change. For node-level task scheduling, the existing learnt NCF model parameters can be directly used for the newly added worker node based on the resource configuration of the existing worker nodes.
- We analyze the performance of our scheduling approach with other three widely used methods on different factors. The extensive experimental results show that our learning-driven and interference-aware workload parallelization approach can achieve higher throughput and optimize resource utilization compared with other three methods.

The rest of this paper is organized as follows. Section 2 introduces the techniques used to realize our approach. Section 3 defines the problem to be solved. Section 4 presents our learning-driven and interference-aware scheduling approach. Section 5 presents the experimental results. Section 6 reviews the related work and Section 7 concludes this paper.

2 Prerequisites

Before describing our problem, we briefly reviews Deep Reinforcement Learning (DRL) and Neural Collaborative Filtering (NCF) techniques used in this paper.

2.1 Deep Reinforcement Learning

A general reinforcement learning problem contains of three main concepts, namely, environment state, action and reward. The agent interacts with an environment sequentially. At each time step t, the agent observes some environment state s_t, and is asked to choose an action a_t. Following the action, the state of the environment is converted to s_{t+1}, and the agent receives a reward r_t. The learning goal is to maximize the expected cumulative reward, where the rewards are discounted by a factor $\gamma \in [0, 1]$ per time step. The future discounted reward at time t is defined as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ where T is the time step when the episode terminates. The optimal action-value function $Q^*(s, a)$ is defined as the maximum expected return that can be achieved by following a policy, after observing some state sequence s and then taking some action a. This function obeys the Bellman equation $Q^*(s, a) = \mathbf{E}[r + \gamma \max_{a'} Q^*(s', a')]$ and can be estimated by using a function approximator, $Q(s, a; \theta) \approx Q^*(s, a)$. Then, the parameters θ are optimized to solve the Bellman equation. Typically, the model-free Q-learning algorithm can iteratively update the action-value function $Q(s, a; \theta)$ for choosing actions that maximize a quality function $Q_{t+1}(s_t, a_t)$ at a specific time step t. However, Q-learning algorithm is highly unstable or divergent when combined with non-linear function approximators.

Deep Q-Network (DQN) using deep neural network as function approximator is recently proposed to solve the above problem, and this solution is much more stable in practice. A neural network function approximator with weights θ is defined as a Q-network. Like Q-learning, DQN can be trained iteratively by updating the parameters θ of the Q-network to reduce the mean-squared error of the Bellman equation, directly from experience samples obtained from the algorithms interactions with the environment. The optimal target value y is estimated by using the parameters θ_i^- from some previous iteration, $y = r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$. And meanwhile the loss functions $L_i(\theta_i)$ is optimized at each iteration as follows.

$$L_i(\theta_i) = \mathbf{E}[(r + \gamma \ \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2]. \quad (1)$$

The parameters θ are updated by Stochastic Gradient Descent (SGD) algorithm [19]. At each time-step t, action is selected by an ε-greedy policy with respect to the current Q-network $Q(s, a; \theta)$.

2.2 Neural Collaborative Filtering

Collaborative filtering is a technique from machine learning that has been widely used in recommender systems. There are typically two methods used in collaborative filtering. Using a *neighborhood based method*, system recommends items to a user based on the similarity between users or the similarity between items. Another widely used method is to use a *latent factor model* which characterizes both users and items in a latent factor domain, instead of describing the relationships between users or items alone. Matrix Factorization (MF) is a powerful technique to derive latent factor models which allow us to capture the interactions between users and items by mapping the rating matrix into a joint latent space of user and item features [20]. For example, a rating matrix $R_{M\times N}$ consisting of ratings by M users and N items and each cell y_{ui} represents the rating that user u gives to item i. This matrix can be decomposed into two low-rank matrices $U_{M\times K}$ and $V_{N\times K}$, where K is the dimension of the latent feature space. The goal of the MF is to estimate the missing values $\hat{y}_{ui}$ in the matrix through

$$\hat{y}_{ui} = f(U_u, V_i) = \sum_{k=1}^{K} U_{uk} V_{ik} = U_u^T V_i, \tag{2}$$

where U_u denotes the user latent feature vector and V_i denotes the item latent feature vector.

However, the traditional collaborative filtering methods based on MF techniques suffer from the cold start problem as well as the sparsity problem. In addition, the MF techniques model feature interactions in a linear way by using the fixed inner product, and this is insufficient for capturing the non-linear and complex feature interactions in real-world data. To tackle these problems, several general NCF frameworks [21], [22], [23] are proposed by integrating MF with deep neural network. Instead of using inner product, these approaches can leverage a Multi-Layer Perception (MLP) network to learn the complex and non-linear user-item interaction function while reserving the strengths of linearity of MF. The inputs of neural network are user latent vector $\mathbf{u}_u$ and item latent vector $\mathbf{v}_i$ obtained from the embedding layer. The output of neural network is the predicted score $\hat{y}_{ui}$. The general NCF predictive model can be formulated as

$$\hat{y}_{ui} = \phi_{out}(\phi_X(...\phi_2(\phi_1(\mathbf{u}_u, \mathbf{v}_i))...)), \tag{3}$$

where X is the number of hidden layers, ϕ_{out} and ϕ_x denote the mapping function for the output layer and xth neural collaborative filtering layer. NCF trains by minimizing the loss between $\hat{y}_{ui}$ and its target value y_{ui}. And then the optimization can be done by performing SGD algorithm.

3 Problem Analysis

3.1 Workload Model

We focus on the task parallelization mechanism on hybrid CPU-GPU cluster, where tasks can be presented by a pipeline which consists of multiple fine-grained subtasks. Each subtask is executed according to a predefined workflow with the individual input and output, and there is a fixed dependency between the subtasks. For example, the processing pipeline of video synopsis (a typical video application) is shown in Fig. 1, which usually includes the following main operation steps: Video data reading, GMM background modeling, frame differencing, tube extraction, tube optimization, stitching, and video synopsis generation. Face recognition task usually consists of the following main steps: Image file loading, face locations, batch face locations, face landmarks, face encodings, face distance, and face comparison. In a typical task we focus on, some subtasks must be performed by the CPU, and some subtasks with less computation requirements should be performed on the CPU to achieve a high cost-performance ratio. In addition, some subtasks can be executed on the CPU or the GPU, and are expensive in terms of computation cost. If there exist available GPU resources, it is better to accelerate these compute-intensive subtasks through GPUs.

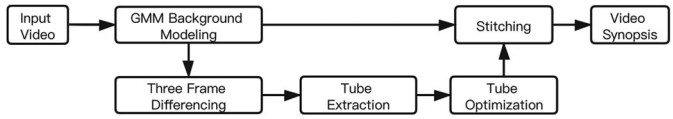

Fig. 1. The processing pipeline of video synopsis.

The efficiency of the task execution is influenced by the characteristics of the task and data size. Therefore, for optimizing the task allocation, we characterize the ith task by $T_i = (TID_i, TSize_i)$ where TID_i is the ID number of task type and $TSize_i$ is the amount of input data. There are no dependencies between the tasks. Further, each task can be represented by a workflow graph $T_i = G(N_i, E_i)$ where N_i is the vertex set of G and E_i is the edge set of G. The subtask set N_i is defined as $N_i = \{ST_{i1}, \ldots, ST_{i,n_i}\}$ where n_i is the number of subtasks in T_i, and each directional edge $E_i^{jk} \in E_i$ represents the dependency from the subtask ST_{ij} to the next subtask ST_{ik} in the pipeline. Each subtask $ST_{ij} (\in N_i)$ is characterized by $ST_{ij} = (STID_{ij}, TSize_i)$ where $STID_{ij}$ is the ID number of subtask type and $TSize_i$ is the amount of input data of the task T_i. Note that the execution efficiency of the subtask ST_{ij} is indirectly influenced by the characteristics $TSize_i$ of the task input data even though the input of the subtask ST_{ij} is not the task input data. We will let the deep network model learn the association pattern between the characteristics $TSize_i$ of the source data and the resource usage characteristics of the subtasks, and thus avoid the complex prediction for the input data of the subtask ST_{ij}.

3.2 System Model

In this work, our distributed computing platform consists of a cluster manager node and a cluster of hybrid CPU-GPU worker nodes, where each worker node comprises multi-core CPUs and multiple GPUs. The cluster manager node is responsible for responding to the task processing requests of the Application System (AS) and controlling the task execution in the cluster. At the cluster level, we consider a task

TABLE 1
Variables Used in Node Feature Vector

Variable	Meaning
$Sr_i^c(t)$	CPU utilization
$Sr_i^g(t)$	GPU utilization
$Sr_i^{cm}(t)$	Host memory utilization
$Sr_i^{gm}(t)$	GPU memory utilization
$Sr_i^{tx}(t)$	Node uplink traffic rate
$Sr_i^{rx}(t)$	node downlink traffic rate

processing request can be divided into a series of tasks each of which is a minimum scheduling unit without dependency between each other. The cluster manager can analyze each task to get the task-related configuration requirements, and then adds the task to a task waiting queue of the cluster. In the cluster manager node, the Cluster Scheduler (CS) can collect the information of worker node resource and task execution states, and use the Learning-Driven Workload Parallelization (LDWP) method to schedules the tasks in the waiting queue to the appropriate worker nodes.

The task allocated to a worker node is first divided into the fine-grained subtasks which are respectively put into the subtask pending queue and the subtask ready queue according to the execution state of its upstream subtask. Then, the Node Scheduler (NS) in each worker node use the Interference-Aware Workload Parallelization (IAWP) method to selects the appropriate computing units (CPU or GPU) for the subtasks in the subtask ready queue. The subtasks assigned to the GPU are first added to a priority queue of GPU subtasks, and then the NS dispatches appropriate subtasks to the GPU taking into account the interference of subtasks. Note that the subtask must be assigned to the CPU if it is required to execute on the CPU device.

We assume the platform has M worker nodes $\{S_1, \ldots, S_M\}$. The resource running statuses influence the execution efficiency of the tasks. In this work, we represent each node i by the feature vector $S_i(t) = (Sr_i^c(t), Sr_i^g(t), Sr_i^{cm}(t), Sr_i^{gm}(t), Sr_i^{tx}(t), Sr_i^{rx}(t))$ at the time step t where the meaning of the variables are listed in Table 1.

3.3 Interference Analysis

With the support of multitasking in modern GPUs, previous work makes an effort on executing multiple application kernels simultaneously on a single GPU device to fully utilize resources of the GPU. However, it can be difficult to effectively schedule concurrent tasks on a GPU due to the tasks interference caused by resource contention between co-executing tasks. In our platform, the IAWP method can predict the characteristics of the subtasks to be performed on the GPU, leveraging the information obtained from the previously executing subtasks. The subtasks with similar characteristics contend for the same resources, and the interference will occur if they are scheduled on the same GPU. Therefore, for improving the execution efficiency of the concurrent subtasks, the IAWP method schedules subtasks according to the predicted similarities between the subtasks in the priority queue of GPUs and the currently running subtasks on GPUs.

The subtask interference is caused by the competition for GPU resources such as Streaming Multiprocessors (SM), memory resources (L1 cache, L2 cache, texture cache and DRAM memory) and interconnect network. We use SM usage, L1 cache usage, L2 cache usage, texture cache usage and DRAM memory usage to characterize the SM and memory resources mentioned above. For profiling the interconnect network, we use global load throughput and global store throughput metrics. We consider the difference in the above resource performance metrics of the subtasks when comparing the similarity between subtasks. If two subtasks are similar, they are considered to be interfering on the same GPU.

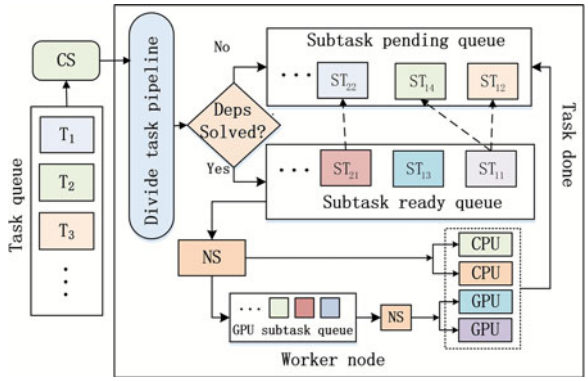

Fig. 2. Two-stage scheduling model.

Objective. In our defined distributed computing platform, AS continues to submit the task processing requests. The objective is to maximize the task throughput in the hybrid CPU-GPU cluster by designing the efficient workload parallelization mechanism while taking into account multi-tasking of the GPUs.

4 Workload Parallelization in Heterogeneous Cluster

4.1 Design Overview

In this section, we propose a two-stage task scheduling approach based on deep reinforcement learning model and neural collaborative filtering model to maximize the cluster task throughput. The task execution workflow is shown in Fig. 2.

Stage 1. The tasks to be executed are added to the cluster level task queue continuously. In the first scheduling stage, the CS allocates the top task in the cluster task queue to one of M worker nodes using the LDWP method. We utilize the DQN model for the cluster level scheduling, because it can continue to learn the complex resource utility pattern of the underlying platform for optimizing a specific workload under varying conditions. The input of the DQN model is the current environment observation vector that includes not only the features of the computing platform resources but also the features of the tasks. The outputs of the DQN model are the expected values of all scheduling actions. Then we perform the task scheduling action which has the maximum expected value compared with other scheduling actions in the action space.

Stage 2. In the CPU-GPU worker node, the allocated task is first divided into multiple subtasks according to the task pipeline. If the dependent upstream subtask of the divided subtask ST_{ij} is completed, ST_{ij} is added into the subtask ready queue. Otherwise, the subtask ST_{ij} is added into the

subtask pending queue. In the second scheduling stage, the NS allocates the top subtask in the subtask ready queue to one of N computing units. The subtasks assigned to the GPU are first added to a GPU subtask priority queue, and then the NS schedules the appropriate subtasks to the GPUs using the IAWP method. Since the the LDWP method does not determine the real parallel execution process of tasks, node scheduling strategy needs to determine how to com-bine the subtasks for efficient parallel execution and make full use of GPU resources. So we need to consider the inter-ference between the incoming subtask and the subtasks exe-cuting on the GPU.

For optimizing the utilization of heterogenous resources and task parallelism, we utilize the NCF model to online estimate the speedup of subtasks and predict the perfor-mance metric values of subtasks simultaneously, where speedup is the relative performance gain of each subtask on the GPU compared to its performance on a CPU core. The inputs of the network model are subtask vector and metric vector that describe subtasks and metrics, where metrics include the speedup, the CPU resource usage and the GPU resource usage discussed in Section 3.3. The output of the network model is the predictive value which is the perfor-mance metric value of a subtask. The the IAWP method assigns subtasks to the appropriate computing unit utilizing the relative order of speedup. Then the IAWP method calculates the similarity of subtasks using the performance metric vectors of subtasks, and decides which subtasks will be exe-cuted concurrently with the existing subtasks on the GPU.

输入和输出好像差不多？？？答: 同一行的不同列

4.2 Task Scheduling in Heterogeneous Cluster

4.2.1 DQN Based Scheduling

In the first scheduling stage, the CS assigns the top task in the cluster task queue to an appropriate worker node in the hybrid cluster using the LDWP method. The setup of the proposed DQN based cluster task scheduling mechanism is described as follows.

Action Space. For the cluster-level task scheduling, we need to choose a worker node S_j from the node set $\{S_1, \ldots, S_M\}$. So the action space for the CS can be represented as $A_{CS} = \{S_1, \ldots, S_M\}$.

State Space. The optimal scheduling action a_t at the time step t is determined based on current observation x_t, which is the combination of the current task features $T_i = (TID_i, {}^{c}_{}TSize_i)$ and the resource state features $S_j(t) = (Sr_j^c(t), Sr_j^g(t), Sr_j^{cm}(t), Sr_j^{gm}(t), Sr_j^{tx}(t), Sr_j^{rx}(t))$ $(j = 1, \ldots, M)$ of all worker nodes. Therefore, the current observation $x_t = (TID, TSize, Sr_1^c(t), Sr_1^g(t), Sr_1^{cm}(t), Sr_1^{gm}(t), Sr_1^{tx}(t), Sr_1^{rx}(t), \ldots, Sr_M^c(t), Sr_M^g(t), Sr_M^{cm}(t), Sr_M^{gm}(t), Sr_M^{tx}(t), Sr_M^{rx}(t))$. And the state s_t is a sequence of actions and observations, $s_t = x_1, a_1, x_2, a_2, \ldots, a_{t-1}, x_t$, which are the input to the proposed DQN for the LDWP method.

Reward. The objective of the LDWP method is to maxi-mize the long-term task throughput of the hybrid cluster by taking a sequence of actions. After taking action a_t at the current state s_t, the platform will evolve into a new state s_{t+1} and receive a reward r_t from the environment. The reward at the time step t is the task throughput increase due to the action a_t, which is the current throughput minus the previous throughput.

The LDWP method uses experience replay and target network to train large neural network with high speed of converging. At each time step t during an agent interaction with the environment, the experience tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ is obtained and stored into a replay memory $D_t = \{e_1, \ldots, e_t\}$. The network model is updated based on the mini-batch of experience tuples randomly selected from the pool of stored experience samples. Therefore, during the process of model update, the data efficiency of the experience and the stability of the learning procedure are high, and the variance of the model updates is reduced by using the randomly sampled learning data instead of the sequential experience. In addition, for further improving the stability of the DQN method, a separate neural network (target network) is used for generating the target Q value y in the Q-learning update. The target network has same structure with evaluation network, but the network parameters of target network are cloned from the evaluation network in every μ steps. This modification outperforms the standard Q-learning method by adding delay between the time an update to Q value is made and the time the update affects the target values, which eliminates divergence and oscillations even more.

The Learning-Driven Workload Parallelization algorithm is shown in Algorithm 1 with the above algorithm setup.

Algorithm 1. DQN Based LDWP Method

1: Initialize replay memory;
2: Initialize action-value function Q with random weights θ;
3: Initialize target action-value function Q' with random weights $\theta^- = \theta$;
4: **for** episode = 1, E **do**
5: Initialize state sequence $s_1 = \{x_1\}$;
6: **for** t = 1, T **do** T是任务的个数？？？
7: With probability ε select a random action a_t;
8: Otherwise select an action $a_t = \arg\max_a Q(s_t, a; \theta)$;
9: Execute action a_t and observe next observation x_{t+1}, reward r_t; t+1和 t 写反了？
10: Store transition (s_{t+1}, a_t, r_t, s_t) in memory;
11: Sample random mini-batch of transitions (s_{j+1}, a_j, r_j, s_j) from memory;
12: $target_j = \begin{cases} r_j, \text{ if episode terminates at step j+1;} \\ r_j + \gamma \max_{a'} Q'(s_{j+1}, a'; \theta^-), \text{ otherwise;} \end{cases}$
13: Perform a gradient descent step on $(target_j - Q(s_j, a_j; \theta))^2$ with the network parameters θ;
14: Every μ steps, clone Q to Q';
15: **end for**
16: **end for**

4.2.2 Discussion

Several previous work [24], [25] has pointed out the impor-tance of pre-training in deep learning. The purpose of pre-training is to make the learning of neural network more effi-cient, and it appears to work better than traditional neural network training methods. We first train MLP with random initializations until convergence. And then we use the parameters obtained from pre-training as the initialization parameters of DQN model, and we optimize it with SGD to make the training more efficient.

In the hybrid cluster, the worker nodes can be added and deleted flexibly. For example, we increase the cluster scale to improve the computing power of the cluster, or some worker nodes failed. In our scheduling approach, the change of the cluster will result in the change of the feature space of the cluster-level DQN, and consequently the network model of scheduler needs to be reconstructed for normal use. However, the relearning of DQN model is expensive, and it may take more time to achieve good online scheduling results. In order to solve this problem, the knowledge transfer from the previously constructed DQN model can significantly improve the performance of DQN relearning [26].

For generalizing the LDWP method, we use the parameter-transfer method [26] in this paper to adjust the previously learned DQN model. When the feature space of cluster changes, the hidden layer parameters of the original DQN model are retained to learn the new DQN model, and the input and output of the new model are modified according to the current state of the cluster. Specifically, the input of the new DQN model consists of the current task characteristics and the resource state features of the current worker nodes, and the output of the new DQN model is the expected values of the current available scheduling actions.

Our LDWP method adopts the online learning process, so the dominating part of each scheduling decision is from step 7 to 14 in Algorithm 1. Selecting a random action in Step 7 can be done in $O(1)$ time. Selecting an action in Step 8 is affected by the size of the state space P and the size of the action space M, so it can be performed in $O(PM)$ time. Executing action and storing transition in Step 9-10 can be done in $O(1)$ time. Calculating target value in Step 12 can be done in $O(PM)$ time. Updating target network in Step 14 can be performed in $O(PM)$ time. Thus, the overall time complexity of LDWP algorithm is $O(PM)$.

4.3 Task Scheduling in CPU-GPU Node

In the second scheduling stage, the NS of the worker node S_j selects an appropriate computing unit for the subtasks in the node subtask ready queue according to the relative GPU-vs-CPU speedup values. The subtasks assigned to the GPU are first added to a GPU subtask priority queue, and then NS selects appropriate subtasks from the queue to be executed simultaneously with the existing subtasks on the GPU considering the subtask interference. The details of the proposed NCF based IAWP method is described as follows.

4.3.1 Initialization

First, we characterize subtask ST_{ij} defined in Section 3.1 by a vector with fine-grained performance metrics $ST_{ij} = (STr_{ij}^{C}, STr_{ij}^{Mem}, STr_{ij}^{SM}, STr_{ij}^{L1}, STr_{ij}^{L2}, STr_{ij}^{Tex}, STr_{ij}^{Dram}, STr_{ij}^{Tpl}, STr_{ij}^{Tps}, TSize_i)$ where the meaning of the variables are listed in Table 2.

For obtaining the estimated speedup values of each subtask and detecting the interference between subtasks according to their similarity, we construct a SubTask Description Matrix (STDM) to characterize subtasks, where each row represents a subtask, each column represents a metric of performance features, and each cell represents a performance metric value of a subtask. For selecting appropriate computing units for subtasks before considering subtask interference on the GPU, we add a column named speedup to the matrix, where speedup is the relative performance gain of each subtask on the GPU compared to its performance on a CPU core. Each subtask in the matrix can be represented by the vector $ST_{ij}^{'} = (ST_{ij}, spd)$, where spd denotes the estimated speedup.

TABLE 2
Variables Used in Subtask Feature Vector

Variable	Meaning
STr_{ij}^{C}	CPU usage
STr_{ij}^{Mem}	Host memory usage
STr_{ij}^{SM}	GPU SM usage
STr_{ij}^{L1}	GPU L1 cache usage
STr_{ij}^{L2}	GPU L2 cache usage
STr_{ij}^{Tex}	GPU texture cache usage
STr_{ij}^{Dram}	GPU memory usage
STr_{ij}^{Tpl}	Global load throughput of GPU
STr_{ij}^{Tps}	Global store throughput of GPU

The NS needs to obtain performance metric values for each subtask to aid the selection of computing units and the detection of subtask similarity. However, it cannot be tolerated to obtain full characteristics for each incoming subtasks in clusters, which leads to redundant computation and introduces delay. Therefore, the NS runs two short profiling for each incoming subtasks to obtain two metric values randomly selected from the feature vector $ST_{ij}^{'}$. We use the proc file system to profile the performance metrics in the CPU and the NVIDIA profiler tools to profile the performance metrics in the GPU.

The profiler needs to run long enough to profile each function in the subtask at least once. To obtain an appropriate time for profiling, we analyze 16 subtasks from Table 5 with different profiling time. We first run these subtasks in full to get the correct performance metrics. Then we run these subtasks with different profiling time to obtain the performance metrics, and calculate the accuracy of these performance metrics compared to the correct performance metrics. As shown in Table 3, we analyze the accuracy of the performance metrics obtained when the profiling time is 2-second, 3-second, 4-second, 5-second and 6-second, respectively. As can be seen from the table, the accuracy continues to increase until the profiling time is 4 second, and the accuracy no longer changes significantly. We found that a 4-second profiling is the most appropriate, because using 4-second profiling can not only obtain effective performance metrics of feature vectors, but also do not cause a waste of system resources. We maintain a CPU core and a GPU for initiating the profilers. Subtasks can also be performed on this GPU and CPU core when there are available resources. The profiling of metrics for incoming subtask is collected and filled in a vector and then we stitch the vector onto the STDM.

4.3.2 NCF Based Performance Metrics Prediction

To fill the missing values in the STDM, we use NCF model. The inputs of the NCF model are two feature vectors $\mathbf{x}_s$ and $\mathbf{x}_m$ that describe subtask s and performance metric m. We use the types of a subtask and a performance metric as the

TABLE 3
Accuracy of Performance Metrics With Different Profiling Time

Profiling time	Accuracy
2-second	78.9%
3-second	87.1%
4-second	92.3%
5-second	92.5%
6-second	92.6%

input feature and respectively convert them to a binarized sparse vector with one-hot encoding. Above the input layer is the embedding layer which projects the sparse vectors of the input layer to the dense vectors. Due to the one-hot encoding of subtask (metric) ID of input layer, the obtained embedding vector can be seen as the subtask (metric) latent vector of latent factor model. We represent $\mathbf{U}^T\mathbf{x}_s$ as the subtask latent vector $\mathbf{u}_s$ and $\mathbf{V}^T\mathbf{x}_m$ as the metric latent vector $\mathbf{v}_m$, where $\mathbf{U} \in \mathbb{R}^{M\times K}$ and $\mathbf{V} \in \mathbb{R}^{N\times K}$ denote the latent factor matrix for subtasks and metrics, and M and N denote the numbers of subtasks and metrics respectively.

The subtask latent vector and performance metric latent vector are then sent to the MPL to discover the interactions between subtasks and metrics while mapping the latent vectors to predicted value. The output of the NCF model is the estimated missing value $\hat{y}_{sm}$. The NCF model is defined as

$$\begin{aligned} \mathbf{a}_1 &= \phi_1(\mathbf{u}_s, \mathbf{v}_m) = \begin{bmatrix} \mathbf{u_s} \\ \mathbf{v}_m \end{bmatrix}, \\ \phi_2(\mathbf{a}_1) &= g_2(\mathbf{W}_2^T\mathbf{a}_1 + \mathbf{b}_2), \\ &\cdots\cdots \\ \phi_L(\mathbf{a}_{L-1}) &= g_L(\mathbf{W}_L^T\mathbf{a}_{L-1} + \mathbf{b}_L), \\ \hat{y}_{sm} &= \sigma(\mathbf{h}^T\phi_L(\mathbf{a}_{L-1})), \end{aligned} \tag{4}$$

where $\mathbf{W}_x$, $\mathbf{b}_x$, g_x and $\mathbf{h}$ are respectively the weight matrix, bias vector, activation function for the xth layer of the perception network, and edge weights of the output layer. The activation function for each layer is the Relu function which is well-suited for sparse data and is less likely to be overfitting. Then training is performed by minimizing the mean square loss between $\hat{y}_{sm}$ and its target value y_{sm}. We also use regularization to prevent overfitting in NCF model. The loss function can be formulated as

$$J = \frac{1}{2MN}\left[\sum_{s=1}^{M}\sum_{m=1}^{N}(\hat{y}_{sm} - y_{sm})^2 + \lambda\sum_{w_{sm}} w_{sm}^2\right], \tag{5}$$

where λ denotes the weight of regularization, and w_{sm} denotes the weight of training instance obtained from the weight matrix $\mathbf{W}_x$. Then we use SGD algorithm to optimize the NCF model.

4.3.3 Interference Aware Scheduling

The IAWP method first schedules subtasks to appropriate computing units utilizing the relative speedup values obtained from the STDM. If a CPU core is idle, the subtask with the minimum estimated speedup value is assigned to the CPU core. We first create a fixed-length GPU subtask priority queue to manage the subtasks dispatched to the GPU. If the priority queue is not full, the subtask with the maximum estimated speedup value will be added to queue in turn.

Next, the IAWP method uses the performance metric vectors of subtasks to detect the interference between the subtask in the priority queue and the subtasks executed on the GPU, and then decides which subtask will be executed concurrently with the existing subtasks. There is interference between two subtasks if the two subtasks are similar. Since we only consider subtask interference on the GPU, we simplify the performance metric vector of the subtask to $G_i = (STr_{ij}^{SM}, STr_{ij}^{L1}, STr_{ij}^{L2}, STr_{ij}^{Tex}, \sout{Tr_{ij}^{Dram}}, STr_{ij}^{Tpl}, \sout{S}STr_{ij}^{Tps}, TSize_i)$. We use the cosine of subtask performance metric vectors to measure the similarity between subtasks, which is defined as

$$cos(\theta) = \frac{\mathbf{G}_i \cdot \mathbf{G}_j}{\|\mathbf{G}_i\|\|\mathbf{G}_j\|} = \frac{\sum_{k=1}^{n} G_{ik} \times G_{jk}}{\sqrt{\sum_{k=1}^{n} G_{ik}^2} \times \sqrt{\sum_{k=1}^{n} G_{jk}^2}}, \tag{6}$$

where θ is the angle between two vectors, and $cos(\theta) \in [-1, 1]$ denotes the similarity of two vectors. The lager the $cos(\theta)$ indicates the smaller \sout{the} angle between the two vectors, which means the two vectors are more similar; and the smaller the $cos(\theta)$ indicates the larger \sout{the} angle between the two vectors, which means the two vectors are less similar. We calculate the similarity between one of the subtasks in the priority queue and each subtask executing on the GPU separately using equation (6). Then we use the reciprocal of the average of all calculated $cos(\theta)$ values as the priority of the subtasks in the priority queue. If the utilization of GPU and device memory is below the given threshold, the NS assigns the subtask with highest priority to the GPU.

The node-level NCF based Interference-Aware Workload Parallelization algorithm is shown in Algorithm 2.

4.3.4 Discussion

To make the learning of neural network more efficient, we use pre-training in the IAWP method. We first train MLP with random initializations until convergence. And then we use the parameters obtained from pre-training as the initialization parameters of NCF model.

For training MLP from scratch, we use the Adaptive Moment Estimation (Adam) [27] algorithm, which combines the advantages of Adagrad [28] for processing sparse gradient and the advantages of RMSprop [29] for handling non-stationary targets. The Adam method can compute adaptive learning rates for different parameters and obtain faster convergence rate than normal SGD. After feeding pre-trained parameters into NCF model, we optimize it with SGD instead of Adam, because Adam needs to save momentum information to update parameters correctly. Since we only initialize NCF with pre-trained parameters, it is not appropriate to further optimize NCF using momentum-based approaches.

For the node-level task scheduling, we also use the parameter-transfer method [26] to adjust the previous NCF model. When a new worker node is added to the cluster, the current learnt NCF model can be directly used for the newly added node if the configurations or the number of

GPUs of the original worker node and the newly added node are the same. If the number of GPUs of the worker node is different, we use the performance metric parameters for the GPU in the original NCF model to fill the performance metric parameters in the new model. For example, if the original worker node has 1 GPU and the newly added node has 2 GPUs, we make two copies of the performance metric parameters for the GPU in the original NCF model as the performance metric parameters in the new NCF model. We assume that the number of GPUs in the worker node is 1 or an integer multiple of 2. In this case, the transferred NCF model used in the newly added node needs to takes a while to be stable.

Algorithm 2. NCF Based IAWP Algorithm

1: Initialize subtask description matrix STDM;
2: Initialize neural network with random weights $\mathbf{W}$;
3: Define two feature vectors $\mathbf{x}_s$ and $\mathbf{x}_m$ as inputs of the NCF model;
4: Obtain subtask latent vector $\mathbf{u}_s$ and metric latent vector $\mathbf{v}_m$ from the embedding layer;
5: Predict missing values in the STDM using MLP, $\hat{y}_{sm} = \phi_{out}(\phi_X(...\phi_2(\phi_1(\mathbf{u}_s, \mathbf{v}_m))...))$;
6: Perform a gradient descent step on the loss function (5) with network weights $\mathbf{W}$ and fill STDM with the optimal values;
7: **if** CPU is idle **then**
8: Assign subtask with minimum estimated speedup value to the CPU;
9: **end if** 这里没体现可能有多个GPU和CPU
10: **for** The fixed-length GPU subtask priority queue is not full **do**
11: Add subtask with maximum estimated speedup value to the GPU subtask priority queue;
12: **end for**
13: **for** Each subtask in the priority queue **do**
14: Obtain the subtask performance metric vectors $\mathbf{G}_i$ from STDM;
15: Calculate the similarity between $\mathbf{G}_i$ and the executing subtasks using the cosine similarity;
16: Use similarity as the priority of the subtask;
17: **end for**
18: **if** The utilization of GPU and memory is below the given threshold **then**
19: Assign subtask with highest priority to the GPU;
20: **end if**

The time complexity of the IAWP method is determined by the number of subtasks n and the complexity of making scheduling decisions. The complexity of the IAWP method is analyzed as follows. The initialization in Step 1-2 can be done in $O(n)$ time. Predicting missing values in subtask feature vector in Step 3-6 can be performed in $O(n)$ time. Assigning appropriate subtasks to the CPU in Step 7-9 can be done in $O(n)$ time. Adding appropriate subtasks to the GPU subtask priority queue in Step 10-12 can be done in $O(n)$ time. The complexity of the similarity calculation is influenced by the number of subtasks executing on the node. Due to the limitation of the maximum number of the co-executing subtasks on the GPU, the complexity of the similarity calculation is determined by the number of GPUs in the node which is defined as G. Therefore, calculating similarity for n subtasks in Step 13-17 can be performed in $O(nG)$ time. Assigning subtask with highest priority to the GPU in Step 18-20 can be done in $O(1)$ time. Thus, the overall time complexity of IAWP algorithm is $O(n + nG)$.

这里也应该是O(nG)吧

TABLE 4
Physical Servers

Type	CPU Num	Mem	GPU Mem	GPU Num	Num
1	2	32GB	24GB	2	2
2	2	32GB	8GB	4	4
3	2	16GB	24GB	4	4
4	2	8GB	8GB	2	2

5 Experimental Evaluation

5.1 Experiment Setup

We evaluate the performance and efficiency of our two-stage task scheduling approach using a distributed computing platform which is built according to the system model given in Section 3. The distributed computing platform is built on a cluster with 12 physical server nodes, and the details of the configurations of the nodes are shown in Table 4, where the CPU is Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40 GHz with 16 cores and the GPU is Tesla P4 or Tesla K80. One of the nodes is used as cluster manager and cluster scheduler, and the others are the worker nodes. The physical nodes in our cluster are interconnected through a Gigabit switch. The streaming applications in our platform are implemented based on Docker (version 1.11.2-cs3) containers which can be dynamically created and destroyed. The container platform is managed by using Kubernetes (version 1.6.0) technology.

We have implemented two typical video processing applications for the experimental evaluation, which are widely used in the intelligent video surveillance systems. One is the intrusion detection, and the other is the video synopsis, and these video processing pipelines can be divided into a series of subtasks. Some of these video subtasks have already been implemented in the existing OpenCV library, and we have implemented the other video subtasks. For validating our approach, we also use the face recognition application, which can be divided into fine-grained subtasks. In addition, for the adaptive subtask scheduling in the CPU-GPU node, some subtasks have two implementation versions which are respectively the CPU version and the GPU version. The details of these streaming applications and their fine-gained subtasks used in our experiments are shown in Table 5. Note that there are many subtask division strategies for one application, and we select one representative division strategy for our experimental analysis.

The DQN used in our experiments is a multi-layer perception network with two hidden layers. The input of the network model is the environment state vector. The output of the network model is the expected values of all scheduling actions. One of the two hidden layers has 256 neurons, and other layer has 128 neurons. The activation function for each neuron is the Relu function which has fast convergence and simpler implementation. We use SGD method to optimize the network model, and set the size of the batch experiences is $batch\_size = 20$. In the DQN model, the learning

TABLE 5
CPU and GPU Implementations of Related Subtasks

Application	Subtask Num	Pipeline operation	
		CPU version	GPU version
Intrusion Detection	6	GMM background modeling, Frame differencing, Gaussian pyramid, DOG pyramid, Scale space extrema detection, Tracking	GMM background modeling, Frame differencing, Gaussian pyramid, DOG pyramid, Scale space extrema detection
Video Synopsis	5	GMM background modeling, Frame differencing, Tube extraction, Tube optimization, Stitching	GMM background modeling, Frame differencing, Tube extraction,
Face Recognition	7	Load image file, Face locations, Face landmarks, Face encodings, Face distance, Compare faces	Batch face locations, Face landmarks, Face encodings, Face distance, Compare faces

rate lr and discount rate γ are two parameters which can significantly affect the overall computational performance of the cluster. Based on the previous work [18], we find that the combination of learning rate lr = 0.1 and discount rate γ = 0.8 can achieve optimal result. Therefore, we use a small learning rate 0.1 to avoid the divergence of the training algorithm, and a large discount rate 0.8 means that the more future estimated rewards will be used to influence the task scheduling decision.

In our experiments, the DCF model is based on a multi-layer perception network with three hidden layers. The inputs of the network model are subtask vector and performance metric vector that consist of subtask type and metric type with one-hot encoding. The output of the network model is the missing value of the subtask description matrix. The first hidden layer of the three hidden layers has 128 neurons, the second hidden layer has 64 neurons, and the last hidden layer has 32 neurons.

5.2 Convergence

The convergence is an important factor of the iterative learning algorithm. For evaluating the convergence of our DQN based cluster scheduling method and DCF based node scheduling method, we first randomly select 3000 processing tasks, including 1000 video synopsis tasks, 1000 intrusion detection tasks and 1000 face recognition tasks. And then we continuously submit the 3000 tasks to the distributed computing platform with the task arrival rate 50/min. The results of the convergence evaluation experiments are shown in Figs. 3 and 4.

For demonstrating the utility of pre-training for DQN model, we compare the convergence speed of DQN with and without pre-training. As shown in Fig. 3, in the first 300 iterations, the average reward of the DQN model without pre-training is lower because the cluster-level scheduler tends to randomly select worker nodes for tasks. The average reward starts to fast rise from the 800 iterations. When the number of iterations reaches 1300, the reward curve is gradually stabilized. For DQN model with pre-training, the average reward starts to fast rise from the 400 iterations and when the number of iterations reaches 700, the reward curve is gradually stabilized. We can see from this figure that the DQN with pre-training achieves better performance than DQN without pre-training, because initialization with pre-trained parameters is more conducive to convergence than random initialization.

For demonstrating the utility of pre-training for DCF model, we compare the convergence speed of DCF with and without pre-training. For DCF without pre-training, we use the Adam to learn it with random initializations. As shown in Fig. 4, the training loss of DCF model is continuously decreasing with more iterations, and it drops rapidly in the first 100 iterations. We can also see from this figure that the DCF with pre-training achieves better performance than DCF without pre-training.

5.3 Performance Analysis

For evaluating our two-stage task scheduling approach, we compare the efficiency of our approach with the following three methods. (1) The random selection method assigns each task to the worker node or computing unit randomly. (2) In the ECT method [30], a scheduler calculates the Minimum Completion Times (MCT) for tasks in the computing unit, and then assigns the task to the computing unit with

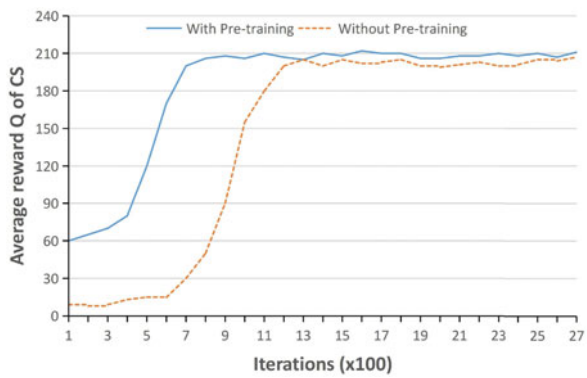

Fig. 3. Convergence of cluster scheduler across 3000 tasks with and without pre-training.

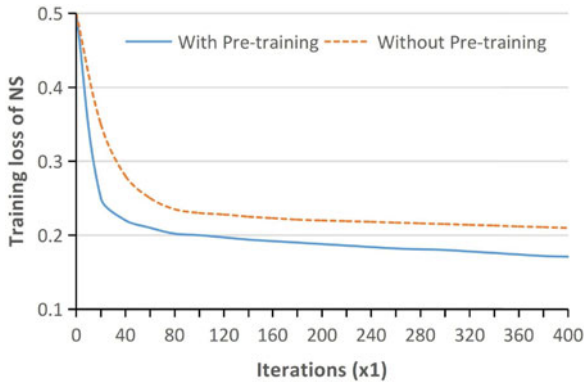

Fig. 4. Convergence of node scheduler across 3000 tasks with and without pre-training.

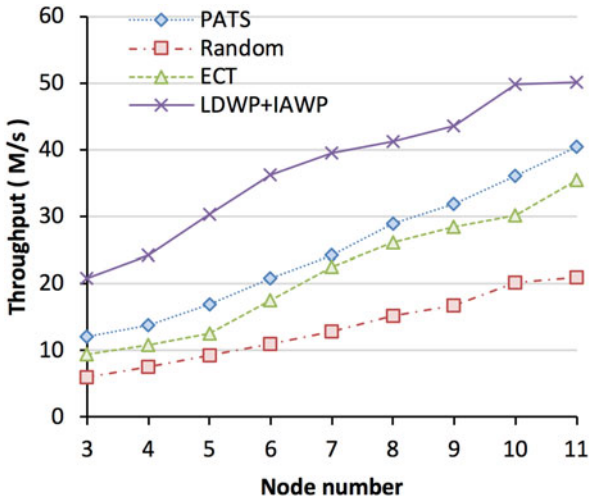

Fig. 5. Cluster throughput of 1500 tasks with different number of nodes.

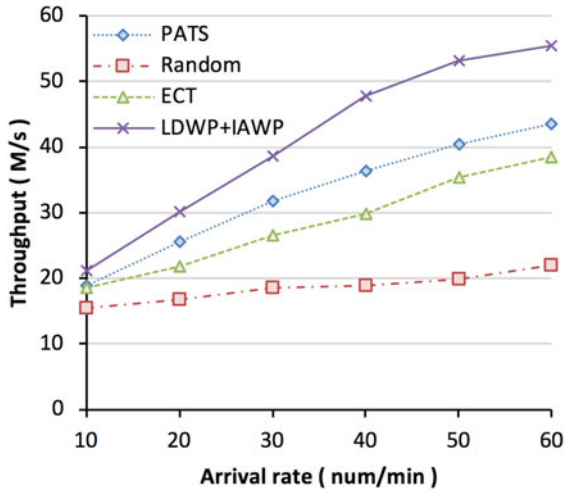

Fig. 7. Cluster throughput of 1500 tasks with different task arrival rate (10/min, 20/min, 30/min, 40/min, 50/min, and 60/min).

the minimal estimated MCT. (3) In the PATS method [31], [32], the scheduler assigns tasks to CPUs or GPUs based on an estimate of the relative performance of each task on a GPU compared to its performance on a CPU core as well as the computational loads of the CPUs and GPUs. We use 1500 tasks to validate our method, and each task is randomly selected from 500 video synopsis tasks, 500 intrusion detection tasks and 500 face recognition tasks.

5.3.1 Impact of Cluster Node Number

This subsection investigates the performance impact of our two-stage task scheduling approach at different cluster scales. We continuously submit the above 1500 tasks at a task arrival rate of 50/min, which will be executed in the hybrid cluster with different number of nodes. We first use LDWP method to schedule each task to appropriate worker node, and then we use IAWP method to assign subtasks to appropriate heterogeneous computing unit. During the execution of these tasks, we calculate the throughputs of cluster and single worker node.

Fig. 5 shows the cluster throughput change in the experiment. As shown in the figure, we can find the cluster throughputs of the four scheduling methods increase as the number of cluster nodes increases, because there are more computing resources can be used to process tasks. The two-stage task scheduling approach obviously outperforms the other three methods. In our experiments, the system throughput is increased about 38.8 percent by using our LDWP method and IAWP method when the node number is 10. In addition, the throughput of the LDWP and IAWP method continues to grow until the number of cluster nodes is 10 and then begins to achieve a relatively stable phase. This is because the average task arrival rate is a constant, and the cluster throughput cannot be improved significantly once there are sufficient computing resources for the current tasks. Therefore, the arrival task can be processed immediately without long queue waiting time.

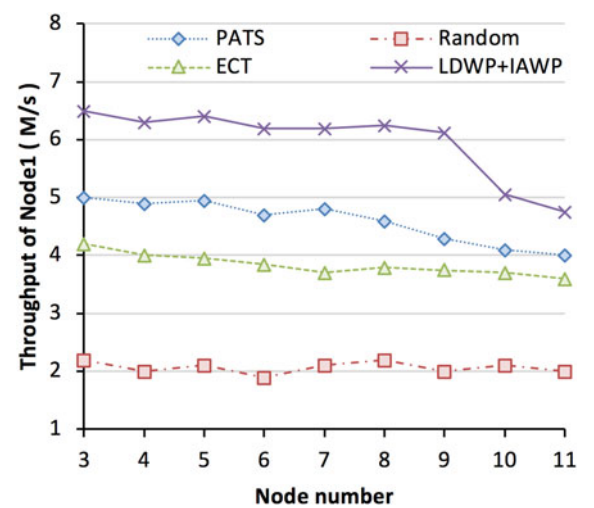

Fig. 6. Node throughput of 1500 tasks with different number of nodes.

Fig. 6 shows the throughput change of the worker node 1 in the experiment. As shown in the figure, we can find the throughput of the four scheduling methods is relatively stable when the number of the cluster nodes increases from 3 to 9, and then begins to gradually decline when the node number is 10. This is because the processing speed of the cluster is greater than the task arrival rate, which causes the computing units to be idle for a period of time. In addition, the performance of the LDWP and IAWP method is higher than those of the other methods. The throughput of node 1 is increased about 31.9 percent by using the LDWP and IAWP method when the node number is 6.

5.3.2 Performance With Different Task Arrival Rate

This subsection presents the effectiveness of our two-stage task scheduling approach at different task arrival rate. We continuously submit the above 1500 tasks with the changing task arrival rate. The number of the worker nodes is 11, and the task arrival rates are respectively 10/min, 20/min, 30/min, 40/min, 50/min, and 60/min in the experiment. ~~We first use LDWP method to schedule each task to appropriate worker node, and then we use IAWP method to assign subtasks to appropriate heterogeneous computing unit~~. Then we calculate the throughputs of cluster and single worker node with different task arrival rate.

Fig. 7 shows the cluster throughput change of the four scheduling methods in the experiment. As shown in the

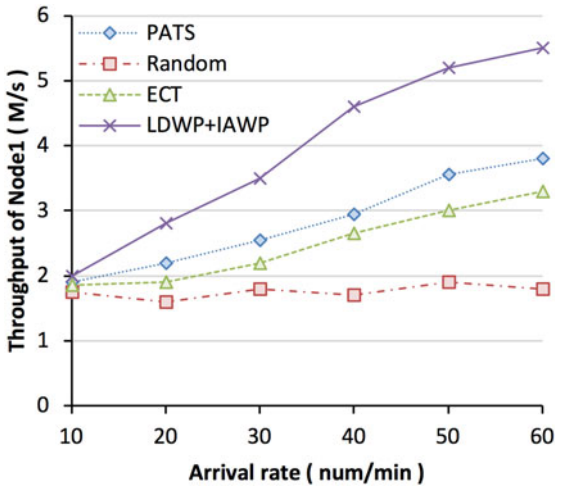

Fig. 8. Node throughput of 1500 tasks with different task arrival rate (10/min, 20/min, 30/min, 40/min, 50/min, and 60/min).

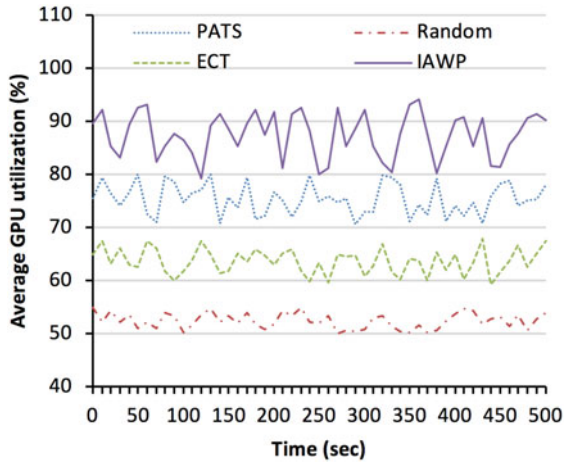

Fig. 10. GPU utilization of worker node 1 for 1500 tasks with task arrival rate 50/min.

figure, the cluster throughput of each method increases as the task arrival rate increases. When the task arrival rate is 10/min, the performances of all methods are similar. This is because the task arrival rate is slower than the task processing speed in the cluster. When the task arrival rate increases, the performance of the LDWP method and IAWP method is obviously better than those of the other methods. In our experiments, the system throughput is increased about 27.9 percent by using our LDWP and IAWP method when the task arrival rate is 60.

Fig. 8 shows the throughput change of the worker node 1 in the experiment. As shown in the figure, the node throughput of each method increases as the task arrival rate increases. Similarly, our LDWP and IAWP method obviously outperforms the other three methods at node level with the increase of the task arrival rate. And the throughput of node 1 is increased about 41.0 percent by using the LDWP and IAWP method when the task arrival rate is 60.

5.3.3 Evaluation of IAWP Method in Terms of GPU Utilization

This subsection tests the ability of the node scheduling method to handle subtask interference. We continuously submit the 1500 tasks with task arrival rate 50/min. We use LDWP method to schedule each task to appropriate worker node and then we analyze GPU utilization of cluster and single worker node with the four scheduling methods.

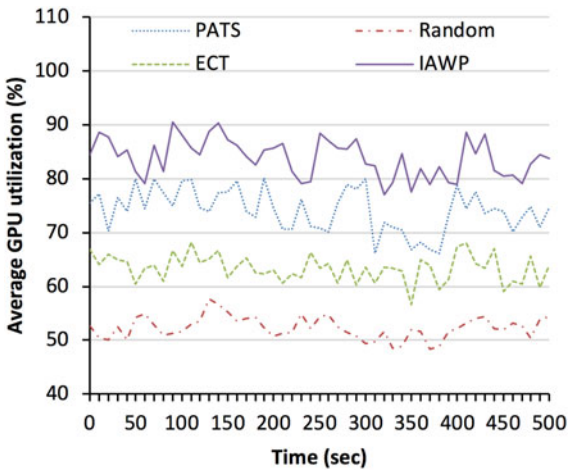

Fig. 9. GPU utilization of cluster for 1500 tasks with task arrival rate 50/min.

Fig. 9 shows the cluster GPU utilization of the four scheduling methods in the experiment starting from the basic stability of the system. As shown in the figure, the GPU utilization curve fluctuates slightly with the continuous execution and completion of subtasks. We observe that the IAWP is able to achieve an average GPU utilization of 83 percent, which obviously outperforms the other three methods. In our experiments, the average GPU utilization is increased about 14.7 percent by using our IAWP method.

Fig. 10 shows the node GPU utilization of the four scheduling methods in the experiment starting from the basic stability of the system. As shown in the figure, the GPU utilization curve fluctuates slightly with the continuous execution and completion of subtasks. We observe that the IAWP method is able to achieve an average GPU utilization of 87 percent, which obviously outperforms the other three methods. This is because the node scheduler is interference-aware, which assigns a subtask to a GPU considering the subtasks interference and the device utilization.

5.3.4 Evaluation of IAWP Method in Terms of Load Balancing Degree

This subsection tests the ability of the node scheduling method to handle load balancing. We continuously submit the 1500 tasks with task arrival rate 50/min. We use LDWP method to schedule each task to appropriate worker node and then we analyze the load balancing degree with the four scheduling methods.

We use the resource utilization of a node to indicate the load balancing degree of the node, and we use the standard deviation of load balancing degree between the nodes to indicate the load balancing degree of the cluster. We define the load balancing degree of worker node M_i as

$$Load_i = (U^{CPU} + U^{hMem} + U^{GPU} + U^{dMem})/4, \quad (7)$$

where U^{CPU}, U^{hMem}, U^{GPU}, U^{dMem} are respectively the CPU utilization, host memory utilization, GPU utilization and

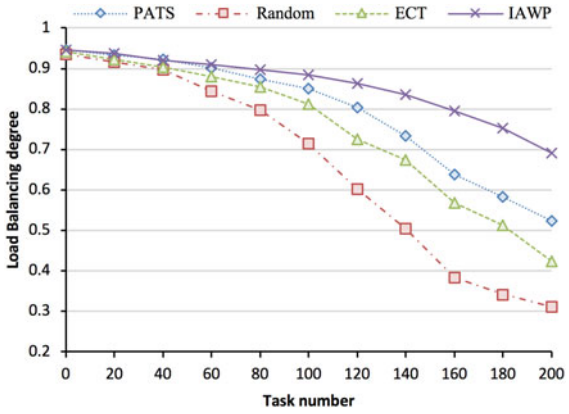

Fig. 11. Degree of load balancing with different number of tasks.

GPU memory utilization. Then we calculate the average load balancing degree as

$$Load_{avg} = \sum_{i=1}^{M} Load_i / M, \tag{8}$$

We define the standard deviation of load balancing degree as

$$Load_{std} = \sqrt{\sum_{i=1}^{M} (Load_i - Load_{avg})^2 / M}, \tag{9}$$

where $Load_{std}$ represents the load difference between all nodes in the cluster. The larger the $Load_{std}$, the larger the load difference between the nodes in the cluster; otherwise, the load of the cluster is more balancing. We define the load balancing degree as

$$Degree_{LoadBalancing} = 1 - Load_{std}. \tag{10}$$

Fig. 11 shows the degree of cluster load balancing corresponding to the four methods. It can be seen from the figure that nodes in the cluster have enough resources when the number of tasks is small, and load balancing degree of the four methods is good. As the number of tasks continues to increase, the balance of our IAWP method is better. This is because our IAWP method considers the interference between the co-executing subtasks, and can achieve a balanced use of resources.

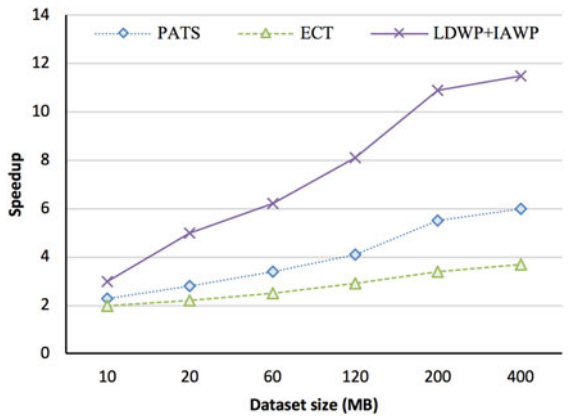

Fig. 12. Speedup compared to Random with different dataset sizes.

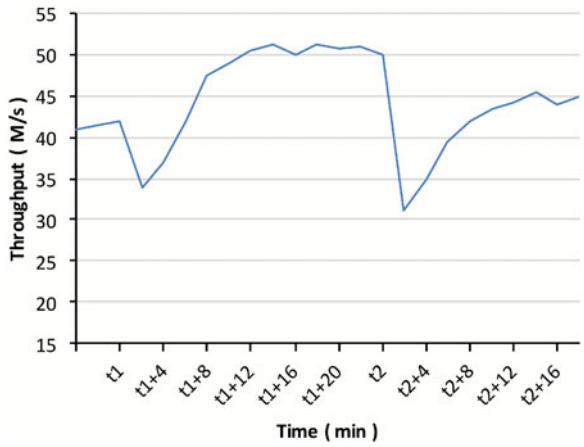

Fig. 13. Throughput of our scheduling approach with cluster changing.

5.3.5 Evaluation of LDWP and IAWP Method in Terms of Speedup

This subsection evaluates the effectiveness of our scheduling approach in items of speedup. We use video synopsis task to complete comparison experiments with dataset sizes of 10 MB, 20 MB, 60 MB, 120 MB, 200 MB, 400 MB, respectively. We define the speedup as the processing time of the Random method divided by the processing time of the other three methods.

Fig. 12 shows the speedup compared to Random with different dataset sizes. As shown in this figure, our LDWP and IAWP method can achieve a speedup of 5 to 10 times, which is much higher than other methods. And as the size of the dataset increases, the speedup of our LDWP and IAWP methods become more obvious.

5.3.6 Performance With Cluster Changing

This subsection evaluates the effectiveness of our scheduling approach when the cluster configuration changes. We continuously submit tasks whose type are randomly selected from Table 5 with task arrival rate 50/min. In the beginning, the number of the worker nodes in the cluster is 8.

The experimental result is shown in Fig. 13. At the time t1, we add three worker nodes to the cluster. After a short period of scheduling model relearning, the cluster throughput starts to rise. After about 12 minutes, the cluster throughput reaches a plateau. We reduce two nodes at the time t2, and then the cluster throughput decreases immediately. After about 10 minutes, the cluster throughput begins to reach a plateau again. As can be seen from the figure, our two-stage task scheduling approach can rebuild the learning model to improve system performance rapidly when the cluster configuration changes.

6 Related Work

6.1 Task Scheduling for Hybrid Architectures

In recent years, many researches focus on task scheduling for hybrid computing devices. In [6], Bleuse *et al.* proposed a method for scheduling independent sequential tasks on hybrid CPU-GPU architectures, where each task can be processed either on a CPU or a GPU. And the objective is to minimize the maximum completion time of the whole schedule.

This work assumes that the execution times of independent tasks on CPU and GPU are known in advance, and it works for sequential tasks only. To solve the drawbacks, Bleuse *et al.* [7] then proposed a new scheduling algorithm using a generic methodology for scheduling independent tasks on hybrid architectures. And tasks are considered to be moldable when assigned to the CPUs. However, the above work assumes that the processing times of tasks are known in advance, which can be time-consuming to obtain in the distributed computing platform.

In [8], the authors proposed a parallel implementation on CPU/GPU of two variants of a stochastic local search method to efficiently solve the scheduling problem in heterogeneous computing systems. This work adopts the Expected Time to Compute (ETC) performance estimation modeled by Ali *et al.* [30], which provides an estimation for the execution time of tasks in heterogeneous computing system. In [9], Zhang *et al.* proposed an online reinforcement learning based task scheduler for hybrid CPU-GPU devices, which can find the optimal execution path and computing devices within the current scheduling scope to optimize the task execution time. Currently, energy consumption plays a key role in high performance computing. In [33], a predictive power-aware scheduling algorithm was presented to distribute computing workload to the resources on heterogeneous CPU-GPU architectures efficiently and further reduce the computing cost in the system. In [34], the authors focused on exploring the relationship between task scheduling algorithms and energy constraints to minimize the energy usage of the workloads on heterogeneous architectures.

Unlike our work, all of the above work does not take the runtime status of system and the interference of tasks performed on the GPU into account. This leads to an unbalanced utilization of cluster resources [35], [36], [37]. In [14], the authors proposed an interference-driven cluster management framework to predict and handle interference for GPU-based heterogeneous clusters, which detects excessive interference and restarts the interfering jobs on different nodes. Mystic [17] can predict the interference between incoming applications and the executing applications on the GPU using Collaborative Filtering (CF), and guide the scheduler to minimize the interference and improve system throughput. However, it only considers task scheduling on the GPU without considering the heterogeneity of the hybrid CPU-GPU architecture.

6.2 Task Scheduling in Heterogeneous Cluster

There are many literatures on task scheduling in heterogeneous cluster. In [38], a heuristic energy-aware stochastic task scheduling strategy was proposed to improve the weighted probability that both the deadline and the energy consumption budget constraints can be met, and the strategy has the capability of balancing between schedule length and energy consumption. In [39], the authors proposed a novel reliability maximization method with energy constraint, which can satisfy the requirement of precedence constrained tasks, while pursuing high performance as well as achieving energy efficiency. The aim of the heuristics scheduling is to improve the task reliability and achieve low power consumption in a heterogeneous computing system. Kumar *et al.* [40] presented an energy efficient scheduling method using Dynamic Voltage and Frequency Scaling (DVFS) technique to execute independent jobs on a heterogeneous cluster system. This work incorporates a resource replacement strategy to generate a cost effective schedule without sacrificing its performance. However, the above work assumes that the tasks are independent of each other, but there are dependencies between tasks in a practical application system sometimes.

In [10], Vasile *et al.* proposed a resource-aware hybrid scheduling algorithm for different types of applications in heterogeneous distributed system. The proposed algorithm considers hierarchical clustering of the available resources into groups in the allocation phase. In [11], the authors proposed a stochastic dynamic level scheduling algorithm to deal with various random variables on heterogeneous cluster systems. The proposed algorithm is able to effectively handle task dependency, time randomness, and processor heterogeneity simultaneously. DRS [12] is a dependency-aware and resource-efficient scheduler which can improve the resource utilization by considering task dependency and tasks' resource requirements. And DRS can reduce the response time of jobs among the CPU-GPU clusters by using the mutual reinforcement learning to estimate the task's waiting time and assign tasks to workers with the consideration of tasks' waiting time. While the previous effort has lent support to the consideration of task dependency, they only consider task scheduling at the cluster level without considering the node level scheduling.

In [13], the authors proposed a two-stage resource provisioning and task scheduling method based on deep reinforcement learning. The proposed strategy considers task dependencies and can achieve low energy cost, low reject rate as well as low runtime with fast convergence. In our previous work [18], we first proposed a two-stage video task scheduling approach based on deep reinforcement learning, which takes into account fine-grained task division and dependencies of each subtask. Then we proposed a transfer learning based generalization method for the proposed task scheduling approach to address the scheduling model relearning issue. Unlike the above work, we also consider multi-tasking parallelism and interference avoidance on the GPU in this paper, which can improve co-execution performance and achieve high system throughput.

7 Conclusion

In this paper, we address the issue of efficient workload parallelization in CPU-GPU heterogeneous cluster and the interference of fine-grained tasks on the GPU. For improving resource utilization and system throughput, we propose a novel two-stage task scheduling approach for streaming applications based on deep reinforcement learning and neural collaborative filtering. First, a cluster-level scheduler assigns task to the appropriate worker node according to the runtime system status and the characteristics of each task. Second, a node-level scheduler distributes subtask to the appropriate computing unit according to the estimated speedup. The subtasks assigned to the GPU are first added to a queue, and then the scheduler selects appropriate subtasks from the queue to be executed simultaneously with the existing subtasks on the GPU considering the subtask

interference. Furthermore, we use pre-training to make the learning of neural network more efficient and propose a transfer learning based generalization strategy to quickly rebuild an effective scheduling model when the computing power of the cluster changes. The extensive experimental results show that our scheduling approach can significantly improve the throughput of distributed computing platform and the utilization of GPUs compared with other widely used algorithms.

Acknowledgments

The work was supported in part by the NSFC under Grant No.61720106007, the Innovation Research Group Project of NSFC (61921003), and the 111 Project (B18008).

References

[1] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs," in *Proc IEEE Symp. Appl. Specific Processors*, 2008, pp. 101–107.

[2] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization*, 2009, pp. 44–54.

[3] G. Teodoro *et al.*, "Coordinating the use of GPU and CPU for improving performance of compute intensive applications," in *Proc. IEEE Int. Conf. Cluster Comput. Workshops*, 2009, pp. 1–10.

[4] Y. Liang, H. P. Huynh, K. Rupnow, R. S. M. Goh, and D. Chen, "Efficient GPU spatial-temporal multitasking," *IEEE Trans. Parallel Distrib. Syst*, vol. 26, no. 3, pp. 748–760, Mar. 2015.

[5] C. Hong, I. Spence, and D. S. Nikolopoulos, "FairGV: Fair and fast GPU virtualization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3472–3485, Dec. 2017.

[6] R. Bleuse, S. Kedad-Sidhoum, F. Monna, G. Mounié, and D. Trystram, "Scheduling independent tasks on multi-cores with GPU accelerators," *Concurrency Comput: Pract. Exper.*, vol. 27, no. 6, pp. 1625–1638, 2015.

[7] R. Bleuse, S. Hunold, S. Kedad-Sidhoum, F. Monna, G. Mounié, and D. Trystram, "Scheduling independent moldable tasks on multi-cores with GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2689–2702, Sep. 2017.

[8] S. Iturriaga, S. Nesmachnow, F. Luna, and E. Alba, "A parallel local search in CPU/GPU for scheduling independent tasks on large heterogeneous computing systems," *J. Supercomput.*, vol. 71, no. 2, pp. 648–672, 2015.

[9] T. Zhang and J. Li, "Online task scheduling for LiDAR data preprocessing on hybrid GPU/CPU devices: A reinforcement learning approach," *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 8, no. 1, pp. 386–397, Jan. 2015.

[10] M.-A. Vasile, F. Pop, R.-L. Tutueanu, V. Cristea, and J. Kołodziej, "Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing," *Future Gener. Comput. Syst.*, vol. 51, pp. 61–71, 2015.

[11] K. Li, X. Tang, B. Veeravalli, and K. Li, "Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 191–204, Jan. 2015.

[12] J. Liu and H. Shen, "Dependency-aware and resource-efficient scheduling for heterogeneous jobs in clouds," in *Proc. IEEE Int. Conf. Cloud Comput. Tech. Sci.*, 2016, pp. 110–117.

[13] M. Cheng, J. Li, and S. Nazarian, "DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *Proc. 23rd Asia South Pacific Des. Autom. Conf.*, 2018, pp. 129–134.

[14] R. Phull, C.-H. Li, K. Rao, H. Cadambi, and S. Chakradhar, "Interference-driven resource management for GPU-based heterogeneous clusters," in *Proc. 21st Int. Symp. High-Perf. Parallel Distrib. Comput.*, 2012, pp. 109–120.

[15] D. Sengupta, A. Goswami, K. Schwan, and K. Pallavi, "Scheduling multi-tenant cloud workloads on accelerator-based systems," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2014, pp. 513–524.

[16] Q. Hu, J. Shu, J. Fan, and Y. Lu, "Run-time performance estimation and fairness-oriented scheduling policy for concurrent GPGPU applications," in *Proc. 45th Int. Conf. Parallel Process.*, 2016, pp. 57–66.

[17] Y. Ukidave, X. Li, and D. Kaeli, "Mystic: Predictive scheduling for GPU based cloud servers using machine learning," in *Proc. 30th IEEE Int. Parallel & Distrib. Proc. Symp.*, 2016, pp. 353–362.

[18] H. Zhang, B. Tang, X. Geng, and H. Ma, "Learning driven parallelization for large-scale video workload in hybrid CPU-GPU cluster," in *Proc. 47th Int. Conf. Parallel Process.*, 2018.

[19] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the Trade*. Berlin, Germany: Springer, 2012, pp. 421–436.

[20] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[21] S. Li, J. Kawale, and Y. Fu, "Deep collaborative filtering via marginalized denoising auto-encoder," in *Proc. 24th ACM Int. Conf. Info. Knowl. Manag.*, 2015, pp. 811–820.

[22] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proc. 26th Int. World Wide Web Conf.*, 2017, pp. 173–182.

[23] X. He and T.-S. Chua, "Neural factorization machines for sparse predictive analytics," in *Proc. 40th ACM Int. Conf. Res. Develop. Inf. Retrieval*, 2017, pp. 355–364.

[24] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," in *Proc. Artif. Intell. Statist.*, 2009, pp. 153–160.

[25] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, 2010.

[26] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015.

[28] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. 7, pp. 2121–2159, 2011.

[29] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," 2012. [Online]. Available: https://www.cs.toronto.edu/tijmen/csc321/slides/lecture_slides_lec6.pdf

[30] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, "Task execution time modeling for heterogeneous computing systems," in *Proc. 9th Heterogeneous Comput. Workshop*, 2000, pp. 185–199.

[31] G. Teodoro *et al.*, "Accelerating large scale image analyses on parallel, CPU-GPU equipped systems," in *Proc. 26th IEEE Int. Parallel Distrib. Proc. Symp.*, 2012, pp. 1093–1104.

[32] G. Teodoro *et al.*, "High-throughput analysis of large microscopy image datasets on CPU-GPU cluster platforms," in *Proc. 27th IEEE Int. Parallel Distrib. Proc. Symp.*, 2013, pp. 103–114.

[33] M. Chiesi, L. Vanzolini, C. Mucci, E. F. Scarselli, and R. Guerrieri, "Power-aware job scheduling on heterogeneous multicore architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 867–877, Mar. 2015.

[34] M. Ciznicki, K. Kurowski, and J. Weglarz, "Energy aware scheduling model and online heuristics for stencil codes on heterogeneous computing architectures," *Cluster Comput.*, vol. 20, no. 3, pp. 2535–2549, 2017.

[35] J. Zhong and B. He, "Kernelet: High-throughput GPU kernel executions with dynamic slicing and scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1522–1532, Jun. 2014.

[36] C. Zhang, J. Yao, Z. Qi, M. Yu, and H. Guan, "vGASA: Adaptive scheduling algorithm of virtualized GPU resource in cloud gaming," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 3036–3045, Nov. 2014.

[37] H. P. Huynh, A. Hagiescu, O. Z. Liang, W.-F. Wong, and R. S. M. Goh, "Mapping streaming applications onto GPU systems," *IEEE Trans. Parallel Distrib. Syst*, vol. 25, no. 9, pp. 2374–2385, Sep. 2014.

[38] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2867–2876, Nov. 2014.

[39] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, and K. Li, "Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster," *Info. Sci.*, vol. 319, pp. 113–131, 2015.

[40] N. Kumar and D. P. Vidyarthi, "An energy aware cost effective scheduling framework for heterogeneous cluster system," *Future Gener. Comput. Syst.*, vol. 71, pp. 73–88, 2017.

Haitao Zhang received the BS degree in mathematics from the Dalian University of Technology, China, in 2006, the MS degree in computer science from Northeastern University, China, in 2008, and the PhD degree in computer science from the Beijing University of Posts and Telecommunications, China, in 2012. From 2010 to 2011, he was a visiting scholar with the Department of Computer Science, Illinois Institute of Technology, Chicago. He is currently an associate professor with the School of Computer Science, Beijing University of Posts and Telecommunications, China. He is the editor-in-chief of more than 10 ITU-T standards related to visual surveillance and cloud computing. His research interests include parallel and distributed computing, cloud computing, big data, and image/video analysis.

Xin Geng received the BS degree in software engineering from Xidian University, China, in 2017. She is currently working toward the MS degree in computer science and technology at the Beijing University of Posts and Telecommunications, China. Her research interests include parallel and distributed computing, heterogeneous computing, cloud computing, resource management and scheduling in distributed systems.

Huadong Ma (Senior Member, IEEE) received the BS degree in mathematics from Henan Normal University, Xinxiang, China, in 1984, the MS degree in computer science from the Shenyang Institute of Computing Technology, Chinese Academy of Science, in 1990, and the PhD degree in computer science from the Institute of Computing Technology, Chinese Academy of Science, Beijing, China, in 1995. He is a Chang Jiang Scholar Professor, and executive dean of Institute of Networking Technology, Beijing University of Posts and Telecommunications (BUPT), China. From 1999 to 2000, he held a visiting position with the University of Michigan, Ann Arbor. He was a visiting professor with the University of Texas at Arlington, Texas from July to September 2004. His current research focuses on multimedia system and networking, Internet of things and sensor networks, and he has published more than 300 papers in prestigious journals (such as the *IEEE/ACM Transactions on Networking*, the *IEEE Transactions on Parallel Distributed Systems*) or Conferences (such as ACM MobiCom/MM, IEEE INFOCOM) and five books. He was awarded the Natural Science Award of the Ministry of Education, China in 2017. He was a recipient of the 2019 Prize Paper Award of *IEEE Transactions on Multimedia*, the 2018 Best Paper Award from IEEE MultiMedia, the Best Paper Award in IEEE ICPADS2010 and the Best Student Paper Award in IEEE ICME2016 for his coauthored papers. He received the National Funds for Distinguished Young Scientists in 2009. He is an editorial board member of the *IEEE Transactions on Multimedia*, the *IEEE Internet of Things Journal*, ACM T-IoT, and MTAP. He serves for chair of ACM SIGMOBILE China.