Cooperative Distributed GPU Power Capping for Deep Learning Clusters

Dong-Ki Kang, Yun-Gi Ha, Limei Peng, and Chan-Hyun Youn, *Senior Member, IEEE*

***Abstract*—The recent GPU-based clusters that handle deep learning (DL) tasks have the features of GPU device heterogeneity, a variety of deep neural network (DNN) models, and high computational complexity. Thus, the traditional power capping methods for CPU-based clusters or small-scale GPU devices cannot be applied to the GPU-based clusters handling DL tasks. This article develops a cooperative distributed GPU power capping (CD-GPC) system for GPU-based clusters, aiming to minimize the training completion time of invoked DL tasks without exceeding the limited power budget. Specifically, we first design the frequency scaling approach using the online model estimation based on the recursive least square method. This approach achieves the accurate tuning for DL task training time and power usage of GPU devices without needing offline profiling. Then, we formulate the proposed FS problem as a Lagrangian dual decomposition-based economic model predictive control problem for large-scale heterogeneous GPU clusters. We conduct both the NVIDIA GPU-based lab-scale real experiments and real job trace-based simulation experiments for performance evaluation. Experimental results validate that the proposed system improves the power capping accuracy to have a mean absolute error of $<1\%$, and reduces the deadline violation ratio of invoked DL tasks by 21.5% compared with other recent counterparts.**

***Index Terms*—Deep learning (DL) cluster, economic model predictive control (EMPC), GPU power capping, Lagrangian dual decomposition, Lipschitz continuity.**

Manuscript received August 17, 2019; revised March 5, 2020, September 3, 2020, December 2, 2020, and March 1, 2021; accepted June 21, 2021. Date of publication July 14, 2021; date of current version February 9, 2022. This work was supported in part by the Defense Challengeable Future Technology Program of the Agency for Defense Development (ADD), South Korea, and in part by the Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean Government, (Core Technology Research for Self-Improving Integrated Artificial Intelligence System) under Grant 20ZS1100. *(Corresponding author: Chan-Hyun Youn.)*

Dong-Ki Kang is with the Department of Information Technology, Jeonbuk National University, Jeonju 54896, South Korea (e-mail: dongkikang@jbnu.ac.kr).

Yun-Gi Ha and Chan-Hyun Youn are with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea (e-mail: yungi.ha@kaist.ac.kr; chyoun@kaist.ac.kr).

Limei Peng is with the School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, South Korea (e-mail: auroraplm@knu.ac.kr).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIE.2021.3095790.

Digital Object Identifier 10.1109/TIE.2021.3095790

I. Introduction

In recent years, the pervasive use of parallel-computing GPUs has accelerated the development of diverse applications based on deep learning (DL). Equipped with highly paralleled streaming multiprocessors (SMs), GPU devices can achieve very rapid distributed computing in the training process for data-rich deep neural networks (DNN). Therefore, research on GPU cluster-assisted DL task processing has received great attention in the past few years. For example, Microsoft and NVIDIA have launched the DGX-1 system which deploys 8 Tesla P100 GPUs (each can yield over 80 teraflops) per server chassis. Facebook has built the GPU cluster, namely, Big Basin, which is similar to DGX-1.

On the other hand, the outstanding computing capability of GPU devices is achieved at the cost of remarkably high power consumption. Even with great advances in transistor density, i.e., 16 nm, the recent NVIDIA Pascal architecture (P100) still requires high thermal design power (TDP) up to 300 W, which is much larger than the modern CPUs. Such high power consumption may cause the power budget overloading and the sudden circuit breakdown [1].

To resolve the above concerns, power capping based on throttling server capability has been discussed in the past years [2]. The main goal of power capping is to adjust the power usage of servers, to maintain the peak power consumption below the given power budget. For CPU-based clusters, power capping methods based on CPU core frequency scaling (FS) have been developed. For instance, a multiinput–multioutput (MIMO) control theoretic approach was proposed to guarantee the control accuracy and stability [3]. Furthermore, dynamic workload-, priority-, and electricity price-aware power capping approaches have been proposed for Internet data centers [4]. The proposed adaptive power capping schemes can efficiently respond to the varying environmental factors. Recently, power control schemes for GPU devices have also been developed. Empirical modeling-based GPU power and performance estimation approaches have been discussed in [5]. They proposed to maximize the overall performance with acceptable power usage on top of a single GPU device. Moreover, offline-profiling-based approaches have been explored in [6] and [7], so as to find the power-efficient frequency setting for a single GPU device.

However, to the best of our knowledge, no generalized approach or model has been developed to implement power capping for GPU-based clusters handling DL tasks. In contrast to a single GPU device, the power capping for GPU-based clusters that handle DL tasks has the following main challenges. First, due to continuous upgrade and replacement [8], GPU devices in the cluster are heterogeneous in having different power and performance characteristics. Therefore, developing a specific power model dedicated to a particular GPU device does not take a good effect on a GPU cluster. Second, it is difficult to make an optimal power capping decision when various different DNN models are simultaneously trained to handle multiple DL tasks. Finally, the power capping is a nonseparable coupled problem throughout entire heterogeneous GPU servers, therefore it is not easy to achieve on-time power capping for large-scale GPU clusters.

Research on power capping for GPU clusters is still in the infancy stage. GPU power and performance estimation approaches have been discussed [5], [9], which highly depend on offline profiling for a specific GPU architecture. The long latency and high cost of offline profiling prevent us from applying such approaches to power capping for a GPU cluster consisting of heterogeneous GPU devices. Recently, the effect of GPU FS on DL task processing has been investigated in [10] and [11] which just exhibits the empirical results without an explicit management scheme for GPU clusters. Runtime power control schemes for small-scale GPU clusters have been explored in [6] and [7], but they are not applicable to large-scale GPU clusters consisting of hundreds of servers, due to the unacceptable computation time for optimal capping decision-making.

To tackle the above challenges, we propose a novel cooperative distributed GPU power capping (CD-GPC) system in this article. The goal of the proposed CD-GPC is to minimize the training performance degradation for invoked DL tasks on GPU clusters by means of on-time power capping that does not exceed the power budget. To do this, We first design an online power and performance model estimator for GPU servers based on the recursive least square (RLS) [12] method and FS (FS) [13]. The estimator enables an accurate runtime tuning for GPU FS without needing the time and cost-consuming offline profiling. Then we formulate the power capping for GPU clusters as an economic model predictive control (EMPC) problem [14], aiming to achieve rigorous and stable power control. Specifically, we exploit the Lagrangian dual decomposition method [15] to divide the entire EMPC problem into multiple subproblems. Thanks to the strong convexity of our EMPC problem, we can apply the tight Lipschitz constant [16] to the updating of the associated dual variable, so as to efficiently accelerate the convergence to the optimal power capping solution.

To evaluate the performance of the proposed CD-GPC system, we conduct the NVIDIA GPUs-based lab-scale real experiments. By training well-known DNN models such as AlexNet [17], GoogleNet InceptionV3 [18], ResNet152 [19], and VGG19Net [20] upon the CD-GPC system, we verify the practicality of our work. Furthermore, in order to strengthen the reliability of the performance evaluation for the CD-GPC system, we conduct the Microsoft real job-trace [21]-based large-scale (200+ GPU servers) simulation experiments. Both lab-scale real experiments and large-scale simulation experiments validate that the proposed system improves the power capping accuracy to have a mean absolute error of $<1\%$, and reduces the deadline violation ratio of invoked DL tasks by 21.5% compared with other recent counterparts.

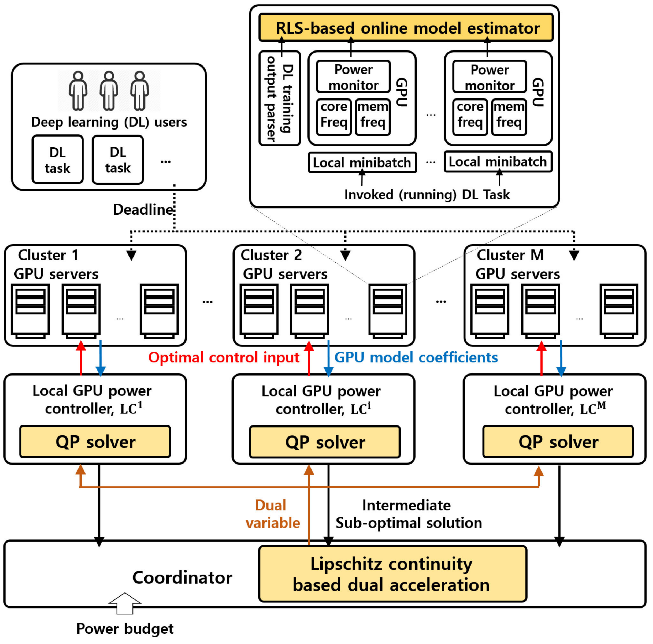

RLS-based online model estimator
DL training output parser
Power monitor
core Freq
mem freq
GPU
Local minibatch
Invoked (running) DL Task
Deep learning (DL) users
DL task
Deadline
Cluster 1 GPU servers
Cluster 2 GPU servers
Cluster M GPU servers
Optimal control input
GPU model coefficients
Local GPU power controller, LC¹
Local GPU power controller, LCⁱ
Local GPU power controller, LCᴹ
QP solver
Dual variable
Intermediate Sub-optimal solution
Coordinator
Lipschitz continuity based dual acceleration
Power budget

Fig. 1. Proposed CD-GPC system for GPU-based cluster handling DL tasks.

II. Overview of CD-GPC System

In this section, we describe the structure of the proposed CD-GPC system and the detailed control procedures. As shown in Fig. 1, the CD-GPC system consists of three major layers, saying local GPU clusters, local GPU power controllers, and a coordinator from top to bottom. A local GPU cluster is usually composed of tens to hundreds of GPU servers, each of which then consists of a single or multiple GPU devices. When a DL task with its associated deadline is invoked on a certain server, the iteration-by-iteration DNN training process is triggered. DNNs are typically trained using stochastic-gradient-descent (SGD), where network model parameters are iteratively updated for every randomly selected training subdataset called *minibatch* [22]. In the SGD method, an *iteration* is considered finished when a *feed-forward* and a *back-propagation* for a single minibatch are completed. An *epoch* consists of multiple iterations and one epoch is completed when all iterations (i.e., for the entire input dataset) are completed. A DL task usually includes hundreds of epochs in order to achieve a satisfactory estimation accuracy based on a predetermined loss function. This hierarchical structure of a DL task is shown in Fig. 2.

Each GPU server has an the internal online model estimator. Whenever an iteration is completed, the corresponding iteration time is reported to the estimator through the training output parser. Meanwhile, the GPU power consumption is monitored by utilities (e.g., NVIDIA-SMI) and then reported to the estimator. Based on the current GPU frequency setting and reported values, the online model estimator repeatedly updates the model coefficients for iteration time and power consumption, by using

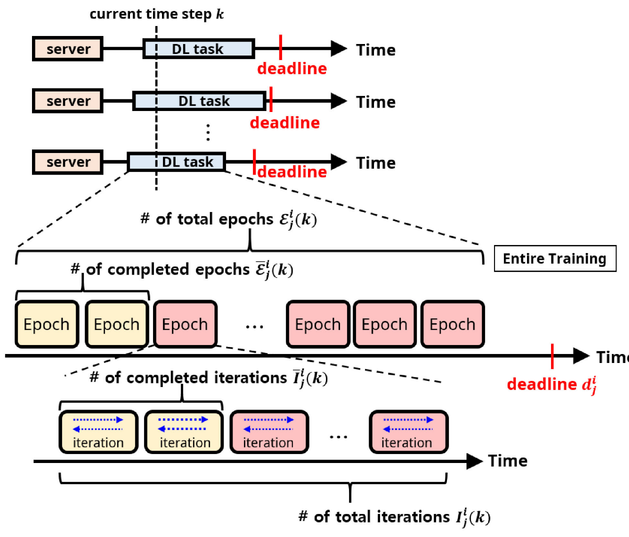


Fig. 2. Epoch- and iteration-based entire deep neural network model training.

the RLS regression method (presented in Section III-C). The model coefficients from all GPU servers in a local cluster are delivered to the corresponding local GPU power controller.

A local GPU power controller elaborates on finding the optimal GPU frequency setting for all GPU servers of a cluster so as to minimize the deadline violation of a DL task. Based on EMPC formulation, the controller adjusts the GPU frequency appropriately for all GPU servers in a local cluster. Meanwhile, since the total power consumed by all local clusters cannot exceed the given power budget, each local GPU power controller receives the *dual variable*, i.e., the price of the power capacity, from the coordinator through the Lagrangian dual decomposition method. Based on the proper dual variable, collected model coefficients, and the deadline, the quadratic programming (QP) solver in the local GPU controller calculates the optimal control input for each GPU server in a local cluster. The details are presented in Sections III and IV.

The coordinator retrieves the system-wide information (i.e., the suboptimal control input of each local GPU power controller and power budget) and provides the dual variable to each local controller. The coordinator gradually updates the optimal dual variable through the repeated interactions with local controllers. Specifically, it exploits the *Lipschitz continuity* of the control problem, so as to accelerate the convergence in dual variable updating. The sufficiently updated dual variable is fed back to the local GPU power controllers as the optimal dual variable, and each local GPU power controller generates optimal control input for GPU servers in each local cluster accordingly.

III. Model and Online Estimator

In this section, we first present the power and performance models based on iteration time of DL tasks and FS of GPUs. Then, we design the RLS method-based online model estimator which enables accurate runtime tuning of GPU core/memory frequency values.

A. Deep Learning Task Model

All invoked DL tasks have their deadline within which all the associated iterations should be completed as shown in Fig. 2. Let d_j^i denote the deadline for a DL task invoked on the GPU server j of the local cluster i. We define the actual completion time of an iteration as *iteration time*, and use it as a performance factor for DL task processing. Let μ_j^i denote the iteration time of a DL task invoked on the GPU server j of the local cluster i. Let T and k denote the unit control period and the index of the current time step, respectively. Then, we present the following definition.

Definition 1: (Iteration Time Requirement)

$$\lambda_j^i(k) = \frac{d_j^i - Tk}{(\mathcal{E}_j^i(k) - \overline{\mathcal{E}}_j^i(k) - 1)\mathcal{I}_j^i(k) + \mathcal{I}_j^i(k) - \overline{\mathcal{I}}_j^i(k)}. \quad (1)$$

The iteration time requirement $\lambda_j^i(k)$ represents the subdeadline of an iteration on the GPU server j of the local cluster i at time step k. Here, $d_j^i - Tk$ denotes the remaining time for completing a DL task without deadline violation. $\mathcal{E}_j^i(k)$ and $\overline{\mathcal{E}}_j^i(k)$ denote the total numbers of required epochs and completed epochs, respectively. $\mathcal{I}_j^i(k)$ is the total number of iterations in an epoch (all epochs contain the same number of iterations) and $\overline{\mathcal{I}}_j^i(k)$ is the number of completed iterations for the current epoch. If the current iteration time is larger than the required one, i.e., $\lambda_j^i < \mu_j^i$, then we deem the deadline for training the DL task is violated. In contrast, if $\lambda_j^i > \mu_j^i$, a part of the assigned computing capability for the DL task is deemed to be wasted, i.e., the power capacity is dissipated. Ideally, they should be the same, i.e., $\lambda_j^i = \mu_j^i$, in order to achieve both power efficiency and performance insurance. Therefore, for a given power budget, the objective of power capping for DL task processing is to minimize the gap between λ_j^i and μ_j^i.

B. Frequency Scaling-Based Power and Performance Model

We exploit the GPU FS approach [13], which adjusts the clock rate of GPU SMs and the access rate of GPU memory controllers. By using the FS approach, we can tune the power usage of GPU servers and iteration time of invoked DL tasks in a fine-grained manner. Specifically, we adopt the regression-based statistical approach, for it is device-agnostic and more suitable for heterogeneous GPU devices than the empirical approaches that are dedicated to certain GPU architectures.

Let $\mathbf{x}_j^i(k) = (x_{j,c}^i(k), x_{j,m}^i(k)) \in \mathbb{R}^2$ denote the GPU core and memory frequency pair of the GPU server j of the local cluster i at time step k. If there are multiple GPU devices in the server, they use the equal frequency setting. Let $\mathrm{p}_j^i(k)$ denote the power consumption of the GPU server j of the local cluster i at time step k and it can be estimated as follows:

$$p_j^i(k) = N_j^i(\alpha_{j,c}^i(k)x_{j,c}^i(k) + \alpha_{j,m}^i(k)x_{j,m}^i(k) + \alpha_{j,e}^i(k)). \quad (2)$$

Here, N_j^i denotes the number of GPU devices in the GPU server j of the local cluster i. $\alpha_{j,c}^i(k), \alpha_{j,m}^i(k), \alpha_{j,e}^i(k)$ denote power consumption model coefficients at time k for associated

GPU core instruction-set, memory instruction-set, and remainder operations, respectively [13]. The current iteration time at time step k, i.e., $\mu_j^i(k)$ can be estimated similar to $\mathrm{p}_j^i(k)$ as follows:

$$\mu_j^i(k) = \frac{1}{N_j^i}\left(\frac{\beta_{j,c}^i(k)}{x_{j,c}^i(k)} + \frac{\beta_{j,m}^i(k)}{x_{j,m}^i(k)} + \beta_{j,\mathrm{e}}^i(k)\right). \tag{3}$$

Note that each GPU server executes only one DL task at a time. In [23], we assume that both the power consumption and the DL performance are linearly proportional to the number of GPU devices, as defined in (2) and (3). $\beta_{j,c}^i(k), \beta_{j,m}^i(k), \beta_{j,\mathrm{e}}^i(k)$ denote iteration time model coefficients at time k, respectively, similar to $\alpha_{j,c}^i(k), \alpha_{j,m}^i(k), \alpha_{j,\mathrm{e}}^i(k)$ [13].

Now, we present the control system dynamics based on the GPU power and performance model (2) and (3). By [3], we present the discrete-time system as follows:

$$\mathbf{x}_j^i(k+1) = \mathbf{x}_j^i(k) + \Delta\mathbf{x}_j^i(k) \tag{4}$$

$$p_j^i(k+1) = p_j^i(k) + \nabla p_j^i(k)^T \Delta\mathbf{x}_j^i(k) \tag{5}$$

$$\mu_j^i(k+1) \approx \mu_j^i(k) + \nabla \mu_j^i(k)^T \Delta\mathbf{x}_j^i(k) \tag{6}$$

where $\Delta\mathbf{x}_j^i(k) = \mathbf{x}_j^i(k+1) - \mathbf{x}_j^i(k)$, $\nabla p_j^i(k) = (N_j^i\alpha_{j,c}^i(k), N_j^i\alpha_{j,m}^i(k)) \in \mathbb{R}^2$, and $\nabla\mu_j^i(k) = (\frac{-\beta_{j,c}^i(k)}{N_j^i(x_{j,c}^i(k))^2}, \frac{-\beta_{j,m}^i(k)}{N_j^i(x_{j,m}^i(k))^2}) \in \mathbb{R}^2$. Due to the nonlinearity of (3), the dynamic model for μ_j^i is formulated as the approximated first-order Taylor equation as shown in (6). Note that it is necessary to find the proper coefficients, saying $\alpha_{j,c}^i, \alpha_{j,m}^i, \alpha_{j,e}^i, \beta_{j,c}^i, \beta_{j,m}^i$, and $\beta_{j,e}^i$, in order to accurately tune the target GPU power consumption and iteration time. Instead of using traditional regression methods, we adopt the RLS-based method which is suitable for online model coefficient estimation.

C. Recursive Least Square-Based Online Model Estimator

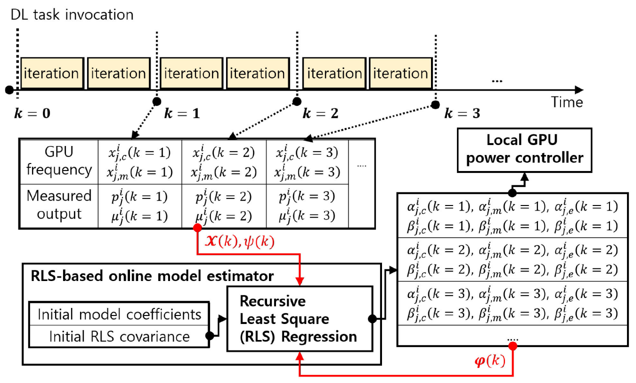
The RLS regression method [12] updates the model coefficients by using both previous model coefficients and newly retrieved sample data. The RLS method fits the model on the fly without requiring fully sampled data and any recalculation for the entire sample data. Due to the low computational burden, it is suitable for the online GPU model estimation.

In the standard RLS regression formula at time step k, the model coefficients $\boldsymbol{\varphi}(k)$ can be estimated based on the measured sample output $\psi(k)$ and the sample input $\boldsymbol{\chi}(k)$ as

$$\boldsymbol{\varphi}(k) = \boldsymbol{\varphi}(k-1) - \frac{\mathbf{O}(k-1)\boldsymbol{\chi}(k)(\boldsymbol{\chi}(k)^T\boldsymbol{\varphi}(k-1) - \psi(k))}{1+\boldsymbol{\chi}(k)^T\mathbf{O}(k-1)\boldsymbol{\chi}(k)} \tag{7}$$

$$\mathbf{O}(k) = \mathbf{O}(k-1) - \frac{\mathbf{O}(k-1)\boldsymbol{\chi}(k)\boldsymbol{\chi}(k)^T\mathbf{O}(k-1)}{1+\boldsymbol{\chi}(k)^T\mathbf{O}(k-1)\boldsymbol{\chi}(k)} \tag{8}$$

where $\mathbf{O}(k)$ is the covariance diagonal matrix for updating model coefficients. The sample output data $\psi(k)$ for the GPU server j of the local cluster i, is measured during the interval $[k-1, k]$, respectively. For the estimation of the GPU power model coefficients, we set $\boldsymbol{\varphi}(k) \leftarrow (\alpha_{j,c}^i(k), \alpha_{j,m}^i(k), \alpha_{j,e}^i(k))$, $\psi(k) \leftarrow \frac{p_j^i(k)}{N_j^i}$, and $\boldsymbol{\chi}(k) \leftarrow (x_{j,c}^i(k), x_{j,m}^i(k), 1)$. For the estimation of the iteration time model coefficients, we set $\boldsymbol{\varphi}(k) \leftarrow (\beta_{j,c}^i(k), \beta_{j,m}^i(k), \beta_{j,e}^i(k))$, $\psi(k) \leftarrow N_j^i\mu_j^i(k)$, and $\boldsymbol{\chi}(k) \leftarrow (x_{j,c}^i(k)^{-1}, x_{j,m}^i(k)^{-1}, 1)$. The initial model coefficients can be estimated based on the historical data. The initial diagonal matrix $\mathbf{O}(k)$ has large diagonal values ($>10^8$). Fig. 3 shows the procedures of the RLS regression from the viewpoint of a single GPU server. For every time step k, our RLS-based online model estimator receives the current GPU frequency vector $\boldsymbol{\chi}(k)$ and the associated output data $\psi(k)$. By using (7) and (8) together with the previous model coefficient vector $\boldsymbol{\varphi}(k-1)$, the estimator derives the updated model coefficient vector $\boldsymbol{\varphi}(k)$ and report it to the local GPU power controller. Then, $\boldsymbol{\varphi}(k)$ is fed back to the estimator as the next feedback value.



DL task invocation
iteration
iteration
iteration
iteration
iteration
iteration
...
Time
$k = 0$
$k = 1$
$k = 2$
$k = 3$
GPU frequency
Measured output
$x_{j,c}^i(k=1)$ $x_{j,m}^i(k=1)$
$x_{j,c}^i(k=2)$ $x_{j,m}^i(k=2)$
$x_{j,c}^i(k=3)$ $x_{j,m}^i(k=3)$
$p_j^i(k=1)$ $\mu_j^i(k=1)$
$p_j^i(k=2)$ $\mu_j^i(k=2)$
$p_j^i(k=3)$ $\mu_j^i(k=3)$
....
$\boldsymbol{\chi}(k), \psi(k)$
Local GPU power controller
$\alpha_{j,c}^i(k=1)$, $\alpha_{j,m}^i(k=1)$, $\alpha_{j,e}^i(k=1)$ $\beta_{j,c}^i(k=1)$, $\beta_{j,m}^i(k=1)$, $\beta_{j,e}^i(k=1)$
$\alpha_{j,c}^i(k=2)$, $\alpha_{j,m}^i(k=2)$, $\alpha_{j,e}^i(k=2)$ $\beta_{j,c}^i(k=2)$, $\beta_{j,m}^i(k=2)$, $\beta_{j,e}^i(k=2)$
$\alpha_{j,c}^i(k=3)$, $\alpha_{j,m}^i(k=3)$, $\alpha_{j,e}^i(k=3)$ $\beta_{j,c}^i(k=3)$, $\beta_{j,m}^i(k=3)$, $\beta_{j,e}^i(k=3)$
....
RLS-based online model estimator
Initial model coefficients
Initial RLS covariance
Recursive Least Square (RLS) Regression
$\boldsymbol{\varphi}(k)$

Fig. 3. Recursive least square (RLS) regression-based online model estimation.

IV. Economic MPC-Based Formulation

In this section, we design the power controller for GPU clusters using control dynamics (4)–(6). We adopt the EMPC theory, which can find the trajectory of optimal frequency setting among multiple GPU servers [14]. The EMPC-based design generally incorporates a general cost function which consists of two terms, i.e., a *tracking error* and a *control penalty*. Our controller optimizes the cost function over a certain time interval, called the *control horizon*. Let H^c denote the length of the control horizon. Then we formulate the EMPC cost function $J(k)$ as follows:

$$J(k) = \sum_{i=1}^{M} J^i(k) \tag{9}$$

where

$$J^i(k) = \sum_{j=1}^{N^i}\sum_{t=1}^{H^c} \tau_j^i \|\mu_j^i(k+t|k) - \lambda_j^i\|^2 + \sum_{j=1}^{N^i}\sum_{t=0}^{H^c-1} \|(\Delta\mathbf{x}_j^i(k+t|k))\|^2_{\mathbf{R}_j^i}. \tag{10}$$

Based on (4), we also set $\Delta\mathbf{x}_j^i(k+t|k) = \mathbf{x}_j^i(k+t+1|k) - \mathbf{x}_j^i(k+t|k)$, where $\mathbf{x}_j^i(k+t|k)$ denotes the *predicted* GPU frequency vector at time step $k+t$. Subsequently, $\mu_j^i(k+t|k) = \frac{1}{N_j^i}\left(\frac{\beta_{j,c}^i(k)}{x_{j,c}^i(k+t|k)} + \frac{\beta_{j,m}^i(k)}{x_{j,m}^i(k+t|k)} + \beta_{j,\mathrm{e}}^i(k)\right)$ denotes the predicted iteration time derived by $\mathbf{x}_j^i(k+t|k)$. Based on (6) and a successive linearization around the current state $\mathbf{x}_j^i(k|k) = \mathbf{x}_j^i(k)$, we set $\mu_j^i(k+t|k) \approx \mu_j^i(k) + \nabla\mu_j^i(k)^T \sum_{l=0}^{t} \Delta\mathbf{x}_j^i(k+l|k)$. The scalar τ_j^i and the positive definite matrix $\mathbf{R}_j^i \in \mathbb{R}^{2\times 2}$ denote the weight for iteration time requirement violation and the weight for control penalty, respectively. The first term of (10) penalizes the violation of iteration time requirement as described in Section III-A. The second term is used to avoid the sudden change of state (i.e., GPU frequency value) [4]. Let $[X]$ denote the integer set $\{1,\ldots,X\}$, then the associated constraints are given as follows:

$$\underline{\mathbf{x}}_j^i \preceq \mathbf{x}_j^i(k+t|k) \preceq \overline{\mathbf{x}}_j^i,\ \forall j\in[N^i], \forall i\in[M], \forall t\in[H^c] \quad (11)$$

$$\sum_{i=1}^{M}\sum_{j=1}^{N^i} p_j^i(k+t|k) \le pb(k),\ \forall t\in[H^c] \quad (12)$$

where $\underline{\mathbf{x}}_j^i, \overline{\mathbf{x}}_j^i \in \mathbb{R}^2$ denote the lower and upper bounds of the GPU frequency vector, respectively. $pb(k)$ denotes the given power budget at time step k. Similar to $\mu_j^i(k+t|k)$, we set $p_j^i(k+t|k) = N_j^i(\alpha_{j,c}^i(k)x_{j,c}^i(k+t|k) + \alpha_{j,m}^i(k)x_{j,m}^i(k+t|k) + \alpha_{j,\mathrm{e}}^i(k)) = p_j^i(k) + \nabla p_j^i(k)^T \sum_{l=0}^{t}\Delta\mathbf{x}_j^i(k+l|k)$. Constraint (11) ensures the feasible frequency range of each GPU server. Constraint (12) represents the total power consumption of the entire cluster should not exceed the power budget at every time step.

Now, based on cost function (9) and constraints (11)–(12), we formulate the *primal quadratic programming (QP)* problem as follows.

Problem 1: (Primal QP control problem)

$$\min_{\Delta\dot{\mathbf{x}}(k)} J(k) = \frac{1}{2}\Delta\dot{\mathbf{x}}(k)^T\mathbf{H}(k)\Delta\dot{\mathbf{x}}(k) + \mathbf{g}^T(k)\Delta\dot{\mathbf{x}}(k) + e(k) \quad (13)$$

subject to

$$\mathbf{A}^{x,i}(k)\Delta\dot{\mathbf{x}}^i(k) \preceq \mathbf{b}^i(k), \quad \forall i\in[M] \quad (14)$$

$$\mathbf{A}^p(k)\Delta\dot{\mathbf{x}}(k) \preceq \mathbf{b}^p(k). \quad (15)$$

Here, $\Delta\dot{\mathbf{x}}(k) = (\Delta\dot{\mathbf{x}}^1(k),\ldots,\Delta\dot{\mathbf{x}}^M(k)) \in \mathbb{R}^{\sum_{i=1}^M 2N^i\cdot H^c}$ denotes the input trajectory over control horizon H^c, where $\Delta\dot{\mathbf{x}}^i(k) = (\Delta\mathbf{x}^i(k|k),\ldots,\Delta\mathbf{x}^i(k+H^c-1|k)) \in \mathbb{R}^{2N^i\cdot H^c}$ and $\Delta\mathbf{x}^i(k+t|k) = (\Delta\mathbf{x}_1^i(k+t|k),\ldots,\Delta\mathbf{x}_{N^i}^i(k+t|k)) \in \mathbb{R}^{2N^i}$. The positive symmetric definite matrix $\mathbf{H}(k)$ and the vector $\mathbf{g}(k)$ are corresponding to Hessian and gradient of (9), respectively. $\mathbf{A}^{x,i}(k)$ and $\mathbf{b}^i(k)$ are corresponding matrix and vector for constraint (11), respectively. $\mathbf{A}^p(k)$ and $\mathbf{b}^p(k)$ are corresponding matrix and vector for power budget constraint (12), respectively. For details, readers of interests are referred to [3]. The scalar $e(k)$ denotes the corresponding constant term for (9) and does not affect the optimal solution for Problem 1.

V. Accelerated Dual Decomposition

A. Lagrangian Dual Decomposition

Due to the high complexity caused by the large dimension of the input trajectory $\Delta\dot{\mathbf{x}}(k)$, i.e., $\sum_{i=1}^M 2N^i\cdot H^c$, it is pretty tough to solve Problem 1 in a centralized fashion. As an alternative, we apply the Lagrangian dual decomposition approach [14] to Problem 1 as follows.

Problem 2: (Dual control problem)

$$\max_{\boldsymbol{\gamma}} D(\boldsymbol{\gamma}) \quad (16)$$

where

$$D(\boldsymbol{\gamma}) = \min_{\Delta\dot{\mathbf{x}}(k)} J(k) + \boldsymbol{\gamma}^T(\mathbf{A}^p(k)\Delta\dot{\mathbf{x}}(k) - \mathbf{b}^p(k)) \quad (17)$$

subject to (14) and

$$\boldsymbol{\gamma} \succeq \mathbf{0}. \quad (18)$$

The vector $\boldsymbol{\gamma} = (\gamma^1,\ldots,\gamma^{H^c}) \in \mathbb{R}^{H^c}$ is the *Lagrangian dual variable* for constraint (15). The term $\boldsymbol{\gamma}^T\mathbf{A}^p(k)\Delta\dot{\mathbf{x}}(k)$ can be decomposed as follows:

$$\begin{aligned}\boldsymbol{\gamma}^T&\mathbf{A}^p(k)\Delta\dot{\mathbf{x}}(k)\\ &= \boldsymbol{\gamma}^T[\mathbf{A}^{p,1}(k)|\cdots|\mathbf{A}^{p,M}(k)][\Delta\dot{\mathbf{x}}^1(k)^T\cdots\Delta\dot{\mathbf{x}}^M(k)^T]^T\\ &= \sum_{i=1}^{M}\boldsymbol{\gamma}^T\mathbf{A}^{p,i}(k)\Delta\dot{\mathbf{x}}^i(k).\end{aligned} \quad (19)$$

Therefore, the dual control problem becomes separable and can be solved in a distributed manner. Based on (19), the dual subfunction $D^i(\boldsymbol{\gamma})$ for LC^i is formulated as follows:

$$D^i(\boldsymbol{\gamma}) = \min_{\Delta\dot{\mathbf{x}}^i(k)} J^i(k) + \boldsymbol{\gamma}^T\mathbf{A}^{p,i}(k)\Delta\dot{\mathbf{x}}^i(k). \quad (20)$$

By using the concept of a conjugate function [24], we can derive the gradient of $D^i(\boldsymbol{\gamma})$ as follows:

$$\nabla_{\boldsymbol{\gamma}}D^i(\boldsymbol{\gamma}) = \mathbf{A}^{p,i}(k)\Delta\dot{\mathbf{x}}^{i*}(\boldsymbol{\gamma},k), \forall i\in[M]. \quad (21)$$

Finally, the unique optimum $\Delta\dot{\mathbf{x}}^{i*}(\boldsymbol{\gamma},k)$ for $D^i(\boldsymbol{\gamma})$ is derived as follows:

$$\Delta\dot{\mathbf{x}}^{i*}(\boldsymbol{\gamma},k) = \operatorname*{argmin}_{\Delta\dot{\mathbf{x}}^i(k)} J^i(k) + \boldsymbol{\gamma}^T\mathbf{A}^{p,i}(k)\Delta\dot{\mathbf{x}}^i(k), \forall i\in[M]. \quad (22)$$

The *strong duality* holds for the primal and dual control problems if there is at least one feasible solution within the *relative interior* of constraints (14) and (15). This is called *Slater's condition* [24]. In such a case, $\min J = \max D$ holds. We can derive the optimal solution $\boldsymbol{\gamma}^*$ for the dual control problem by using the iterative numerical method. Let $\boldsymbol{\gamma}_{(h)}$ denote the interim dual variable at dual step h. Then $\boldsymbol{\gamma}_{(h)}$ is updated by the gradient projection method as follows:

$$\begin{aligned}\boldsymbol{\gamma}_{(h+1)} &= \left[\boldsymbol{\gamma}_{(h)} + \varepsilon\left(\sum_{i=1}^{M}\nabla_{\boldsymbol{\gamma}}D^i(\boldsymbol{\gamma}_{(h)}) - \mathbf{b}^p(k)\right)\right]_+\\ &= \left[\boldsymbol{\gamma}_{(h)} + \varepsilon\left(\sum_{i=1}^{M}\mathbf{A}^{p,i}(k)\Delta\dot{\mathbf{x}}^{i*}(\boldsymbol{\gamma}_{(h)},k) - \mathbf{b}^p(k)\right)\right]_+\end{aligned} \quad (23)$$

where $\varepsilon > 0$ is the gradient step length, and $[\cdot]_+$ represents the projection into the nonnegative orthant. At max count $h_{\max}$, we can derive the (approximated) optimum $\boldsymbol{\gamma}_{(h_{\max})} \approx \boldsymbol{\gamma}^*$.

B. Acceleration With Tight Lipschitz Continuity

Note that the convergence rate for the optimum $\boldsymbol{\gamma}^*$ of (23) depends on the gradient step size ϵ. Instead of using an arbitrary value, we exploit the *tight Lipschitz continuity* in dual functions to find a proper ε for fast dual convergence. Since $J(k)$ is strongly convex and $\mathbf{H}(k)$ is positive symmetric definite, the dual function $D(\boldsymbol{\gamma})$ satisfies the following bound [16]:

$$D(\boldsymbol{\gamma}) \geq D(\boldsymbol{\gamma}') + < \bigtriangledown_{\boldsymbol{\gamma}} D(\boldsymbol{\gamma}'), \boldsymbol{\gamma} - \boldsymbol{\gamma}' > - \frac{1}{2} \|\boldsymbol{\gamma} - \boldsymbol{\gamma}'\|^2_{\mathbf{L}} \quad (24)$$

where for all $\boldsymbol{\gamma}, \boldsymbol{\gamma}' \in \mathbb{R}^{H^c}$, $\mathbf{L} \in \mathbb{R}^{H^c \times H^c}$ is the associated *Lipschitz constant*. Then, (23) can be reformulated as follows:

$$\begin{aligned} \boldsymbol{\gamma}_{(h+1)} &= \left[\underset{\boldsymbol{\gamma}}{\operatorname{argmax}} \left\{ D(\boldsymbol{\gamma}_{(h)}) + \langle \bigtriangledown_{\boldsymbol{\gamma}} D(\boldsymbol{\gamma}), \boldsymbol{\gamma} - \boldsymbol{\gamma}_{(h)} \rangle \right. \right. \\ &\qquad \left. \left. - \frac{1}{2} \|\boldsymbol{\gamma} - \boldsymbol{\gamma}_{(h)}\|^2_{\mathbf{L}} \right\} \right]_+ \\ &= \left[\boldsymbol{\gamma}_{(h)} + \mathbf{L}^{-1} \left(\sum_{i=1}^{M} \mathbf{A}^{p,i}(k) \Delta \dot{\mathbf{x}}^{i*}(\boldsymbol{\gamma}_{(h)}, k) - \mathbf{b}^p(k) \right) \right]_+ . \end{aligned} \quad (25)$$

Obviously, the minimization for L (i.e., maximization for ε) accelerates the convergence rate most rapidly for dual decomposition. Then, we present the following proposition for minimum $\mathbf{L}$, i.e., *tight Lipschitz constant*.

Proposition 1: When the power budget $pb(k)$ is not sufficient to meet the iteration time requirement of all invoked DL tasks, the minimum $\mathbf{L}$, i.e., *tight Lipschitz constant* in (24) is defined as $\mathbf{L}(k) = \mathbf{A}^p(k) \cdot \mathbf{H}^{-1}(k) \cdot \mathbf{A}^p(k)^T$.

Proof: Note that $J(k)$ is the strongly convex QP function. If $pb(k)$ is not sufficient, then $\mathbf{A}^p(k)\Delta\dot{\mathbf{x}}^*(\boldsymbol{\gamma}^*, k) = \mathbf{b}^p(k)$ holds (it means that the total power consumption by the optimal power capping exactly reaches out to the given power budget). And then, we regard the constraint (15) as the equality constrained form and derive the tightest value for $\mathbf{L}$ as $\mathbf{A}^p(k) \cdot \mathbf{H}^{-1}(k) \cdot \mathbf{A}^p(k)^T$ by using *Theorem 10* in [16]. ■

It may be highly time-consuming to directly derive $\mathbf{L}(k)^{-1}$ due to the calculation for $\mathbf{H}^{-1}(k)$. Let $\mathbf{H}(k) = \text{blkdiag}\{\mathbf{H}^1(k), \ldots, \mathbf{H}^M(k)\}$, where $\mathbf{H}^i(k)$ is the Hessian for $J^i(k)$. Then, we present the following proposition.

Proposition 2: Let $\mathbf{A}^{p,i}(k) \cdot \mathbf{H}^{i^{-1}}(k) \cdot \mathbf{A}^{p,i}(k)^T$ denote the local tight Lipschitz constant. The inverse of the tight Lipschitz constant is $\mathbf{L}(k)^{-1} = (\sum_{i=1}^{M} \mathbf{A}^{p,i}(k) \cdot \mathbf{H}^{i^{-1}}(k) \cdot \mathbf{A}^{p,i}(k)^T)^{-1}$.

Proof: Without loss of generality, $\mathbf{H}^i \in \mathbb{R}^{2N^iH^c \times 2N^iH^c} \forall i$ is the invertible square matrix. Then, based on the characteristics of a block diagonal matrix, we can obtain the following:

$$\begin{aligned} \mathbf{L}(k)^{-1} &= (\mathbf{A}^p(k) \cdot \mathbf{H}^{-1}(k) \cdot \mathbf{A}^{pT}(k))^{-1} \\ &= (\mathbf{A}^p \cdot \text{blkdiag}\{\mathbf{H}^{1^{-1}}(k), \ldots, \mathbf{H}^{M^{-1}}(k)\} \cdot \mathbf{A}^p(k)^T)^{-1} \\ &= \left(\sum_{i=1}^{M} \mathbf{A}^{p,i}(k) \cdot \mathbf{H}^{i^{-1}}(k) \cdot \mathbf{A}^{p,i}(k)^T \right)^{-1} . \end{aligned}$$

■

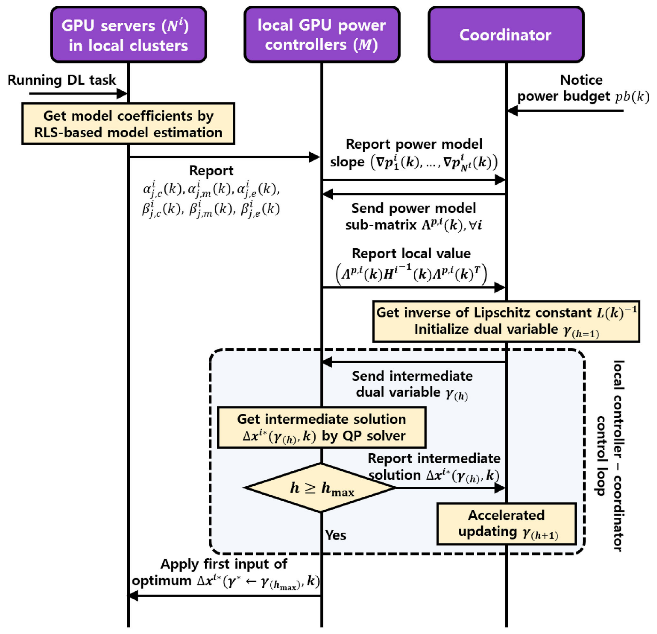

GPU servers (N^i) in local clusters
local GPU power controllers (M)
Coordinator
Running DL task
Notice power budget $pb(k)$
Get model coefficients by RLS-based model estimation
Report $\alpha^i_{j,c}(k), \alpha^i_{j,m}(k), \alpha^i_{j,e}(k), \beta^i_{j,c}(k), \beta^i_{j,m}(k), \beta^i_{j,e}(k)$
Report power model slope $(\nabla p^i_1(k), \ldots, \nabla p^i_{N^i}(k))$
Send power model sub-matrix $A^{p,i}(k), \forall i$
Report local value $(A^{p,i}(k)H^{i^{-1}}(k)A^{p,i}(k)^T)$
Get inverse of Lipschitz constant $L(k)^{-1}$ Initialize dual variable $\gamma_{(h=1)}$
Send intermediate dual variable $\gamma_{(h)}$
Get intermediate solution $\Delta x^{i*}(\gamma_{(h)}, k)$ by QP solver
Report intermediate solution $\Delta x^{i*}(\gamma_{(h)}, k)$
$h \geq h_{\max}$
Accelerated updating $\gamma_{(h+1)}$
Yes
Apply first input of optimum $\Delta x^{i*}(\gamma^* \leftarrow \gamma_{(h_{\max})}, k)$
local controller – coordinator control loop

Fig. 4. Control flow diagram of the proposed CD-GPC system.

Proposition 2 enables the distributed computation for $\mathbf{L}(k)^{-1}$ over multiple local power controllers.

We present the procedures of the CD-GPC, in Algorithms 1 and 2. Based on Proposition 2, the coordinator iteratively updates the dual variable $\boldsymbol{\gamma}_{(h)}$ by interacting with local controllers (line 06–10 in Algorithm 1). The coordinator sends the final dual variable as the optimum to local controllers (line 11 in Algorithm 1). Based on $\boldsymbol{\gamma}_{(h)}$, the QP solver in each local controller calculates the intermediate suboptimum $\Delta\dot{\mathbf{x}}^{i*}(\boldsymbol{\gamma}_{(h)}, k)$ (line 06–10 in Algorithm 2). At $h_{\max}$, the controller applies the first input $\Delta\mathbf{x}^{i*}(k|k)$ from $\Delta\dot{\mathbf{x}}^{i*}$ as the optimal input to all GPU servers (line 13 in Algorithm 2). The control sequences of the CD-GPC system over time are shown in Fig. 4.

VI. Lab-Scale Real Experiments

In this section, we investigate the performance of the proposed CD-GPC approach upon a lab-scale GPU cluster.

A. NVIDIA GPU Based Experimental Environment

The experimental environment of our GPU cluster is shown in Table I. Our experimental environment consists of 1 coordinator, 3 local GPU power controllers, and 11 heterogeneous GPU servers (one of them contains 4 GPU devices). The GPU chipsets are DVFS-capable GTX1060/1080 using NVIDIA PASCAL

Algorithm 1: Coordinator of CD-GPC.

```
INPUT : Power budget for the cluster, $pb(k)$,
OUTPUT : Optimal dual variable, $\gamma^*$,
01 : receive power model slope $\nabla p_j^i(k)$ of (5), from local GPU power controllers $LC^i\ \forall i, j$
02 : form matrix $\mathbf{A}^p(k)$ for power budget constraint of (15) and send submatrix $\mathbf{A}^{p,i}(k)$ to $LC^i \forall i$
03 : receive $\mathbf{A}^{p,i}(k)\cdot \mathbf{H}^{i^{-1}}(k)\cdot \mathbf{A}^{p,i}(k)^T$ from $LC^i \forall i$ (Proposition 2)
04 : get the tight Lipschitz constant $\mathbf{L}(k)^{-1}$ (Proposition 2)
05 : initialize the dual variable $\boldsymbol{\gamma}_{(h=1)}$ of (25)
06 : for dual steps, $h = 1 \rightarrow [h_{\max} - 1]$ do
07 :     send $\boldsymbol{\gamma}_{(h)}$ to $LC^i\ \forall i$
08 :     receive the intermediate suboptimum $\Delta\dot{\mathbf{x}}^{i*}(\boldsymbol{\gamma}_{(h)}, k)$ of (22) from $LC^i\ \forall i$
09 :     get $\boldsymbol{\gamma}_{(h+1)}$ of (25)
10 : end for
11 : $\boldsymbol{\gamma}^* \leftarrow \boldsymbol{\gamma}_{(h_{\max})}$ and send $\boldsymbol{\gamma}^*$ to $LC^i \forall i$
```

Algorithm 2: Local GPU Power Controller LC^i of CD-GPC.

```
INPUT : Iteration time requirement, $\lambda_j^i(k) \forall j$,
OUTPUT : Optimal control input, $\Delta\mathbf{x}^{i*}(k)$,
01 : get model coefficients $\alpha_{j,c}^i(k), \alpha_{j,m}^i(k), \alpha_{j,e}^i(k), \beta_{j,c}^i(k), \beta_{j,m}^i(k), \beta_{j,e}^i(k)$ of (2)–(3) via RLS estimator
02 : get $\nabla p_j^i(k), \nabla \mu_j^i(k) \forall j$ of (5)–(6)
03 : send power model slope $\nabla p_j^i(k), \forall j$ of (5), to the coordinator
04 : receive submatrix $\mathbf{A}^{p,i}(k)$ of (15) from the coordinator
05 : send $\mathbf{A}^{p,i}(k)\cdot \mathbf{H}^{i^{-1}}(k)\cdot \mathbf{A}^{p,i}(k)^T$ to the coordinator (Proposition 2)
06 : for dual steps, $h = 1 \rightarrow [h_{\max} - 1]$ do
07 :     receive dual variable $\boldsymbol{\gamma}_{(h)}$ from the coordinator
08 :     QP solver finds the intermediate suboptimum $\Delta\dot{\mathbf{x}}^{i*}(\boldsymbol{\gamma}_{(h)}, k)$ of (22)
09 :     send $\Delta\dot{\mathbf{x}}^{i*}(\boldsymbol{\gamma}_{(h)}, k)$ to the coordinator
10 : end for
11 : receive optimal dual variable $\boldsymbol{\gamma}^*$ from the coordinator
12 : $\Delta\dot{\mathbf{x}}^{i*}(k) \leftarrow \Delta\dot{\mathbf{x}}^{i*}(\boldsymbol{\gamma}^*, k)$
13 : apply first input $\Delta\mathbf{x}^{i*}(k|k)$ from $\Delta\dot{\mathbf{x}}^{i*}(k)$ to all associated servers
```

architecture that supports CUDA. We use the object-based database framework and MongoDB as the control message queue and use computation network toolkit (CNTK) [25] as the DL framework.

Upon CNTK, we train the well-known DNN models including Resnet152 [19], VGG19net [20], Alexnet [17], and Googlenet InceptionV3 [18]. Table II shows the DL task mapping for the experiment. The iteration time requirement, i.e., IterTR, for each DL task is the median of the processing time range available for each GPU server. All GPU servers initiate the FS at the starting value of the range. The control horizon is $H^c = 10$. We select the max count for dual steps, $h_{\max}$ as 7 based on our empirical analysis. The time interval for GPU power measurement is set as 1 s. The iteration time (IterT) of each DL task is reported whenever the iteration is finished.

In order to evaluate the performance of the CD-GPC system, we use the following metrics, saying, violation of iteration time requirement (sec) denoted by $\max\{\mu_j^i - \lambda_j^i, 0\}, \forall i, j,$ power consumption (W) denoted by $\sum_{\forall i}\sum_{j=1}^{N^i} p_j^i(k)$, dual convergence rate, and control computation time (sec). The curves of the violation of iteration time requirement and the power consumption are shown in Figs. 5 –7, and 10. The curves of the dual convergence rate are shown in Fig. 8. The control computation time for our lab-scale small GPU cluster (which contains 11 GPU servers and three local controllers) is not so long. Therefore we put the evaluation of the control computation time for the large-scale GPU cluster (contains 200 GPU servers) to the next Section VII of simulation experiments.

To the best of our knowledge, our work is the first to study the power control for GPU-based cluster handling DL tasks, and thus, it is difficult to find proper benchmark algorithms. Regarding this, we implement the MIMO CPU power control algorithm [3] as a comparison. For the MIMO algorithm, we use the cost function (5) in [3], instead of using our function

TABLE I
GPU CONTROLLER/SERVER SPECIFICATION, 3 LOCAL CLUSTERS, AND 11 GPU SERVERS, GTX1060(CORE RANGE (MHZ) [580-1850], MEM RANGE (MHZ) [2850-4440]), GTX1080(CORE RANGE (MHZ) [580-1950], MEM RANGE (MHZ) [3515–5414])

Controller	CPU	MEM
Coordinator	Intel i7-3770, 3.4Ghz	16GB
LC^1	Intel Xeon E5-2609, 2.5Ghz	8GB
LC^2	Intel Xeon E5-2609, 2.5Ghz	8GB
LC^3	Intel Xeon E5-2609, 2.5Ghz	8GB

Cluster	Server	CPU	MEM	GPU
Cluster1	s_1^1	Xeon 4110, 2.1Ghz	64GB	1080 8GB (4ea)
Cluster2	s_1^2	i7-6700, 3.4Ghz	16GB	1080 8GB
	s_2^2	i7-4790, 3.6Ghz	8GB	1080 8GB
	s_3^2	i5-4590, 3.3Ghz	8GB	1060 6GB
	s_4^2	E5-2620, 2.1Ghz	16GB	1060 6GB
	s_5^2	i5-2400, 3.1Ghz	8GB	1060 6GB
Cluster3	s_1^3	i7-3770, 3.4Ghz	16GB	1060 6GB
	s_2^3	i7-3770, 3.4Ghz	16GB	1060 6GB
	s_3^3	i7-3770, 3.4Ghz	16GB	1060 6GB
	s_4^3	i7-3770, 3.4Ghz	16GB	1060 6GB
	s_5^3	i7-4790, 3.6Ghz	16GB	1080 8GB

TABLE II
INVOKED DL TASKS, *P*:PARALLEL, *NP*:NONPARALLEL, *ITERTR:* ITERATION TIME REQUIREMENT

Type	DL Task	Server	DNN Model	*IterTR* λ (sec)
P	$a1$	$j = 1, i = 1$	ResNet152	2.5
NP	$a2$	$j = 1, i = 2$	InceptionV3	4
NP	$a3$	$j = 2, i = 2$	InceptionV3	4
NP	$a4$	$j = 3, i = 2$	InceptionV3	4
NP	$a5$	$j = 4, i = 2$	InceptionV3	5.7
NP	$a6$	$j = 5, i = 2$	InceptionV3	4
NP	$a7$	$j = 1, i = 3$	AlexNet	5
NP	$a8$	$j = 2, i = 3$	AlexNet	3.5
NP	$a9$	$j = 3, i = 3$	VGG19Net	4.5
NP	$a10$	$j = 4, i = 3$	VGG19Net	4.2
NP	$a11$	$j = 5, i = 3$	VGG19Net	3.5

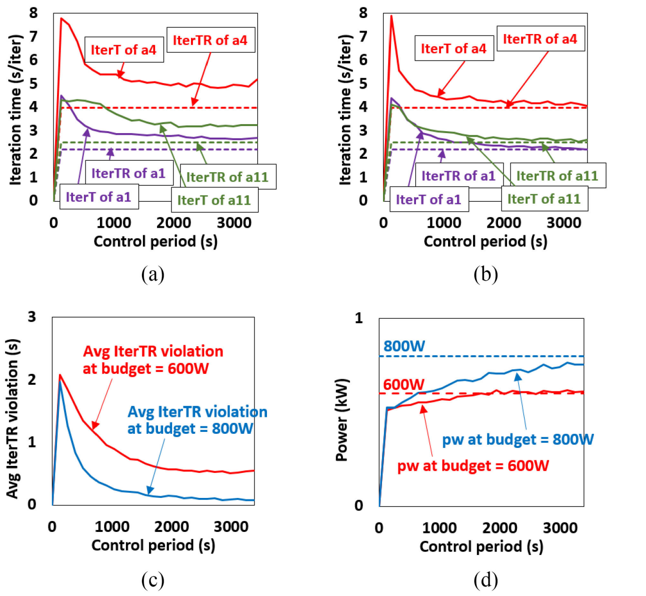


Fig. 5. Control accuracy of the proposed CD-GPC method at low power budget $pb = 600$ W, and enough one $pb = 800$ W. (a) IterT curves of 3 DL tasks a1, a4, a11 with $pb = 600$ W. (b) IterT curves of 3 DL tasks a1, a4, a11 with $pb = 800$ W. (c) Average IterTR violation curves of 11 DL tasks. (d) Total power consumption curves of 11 servers.

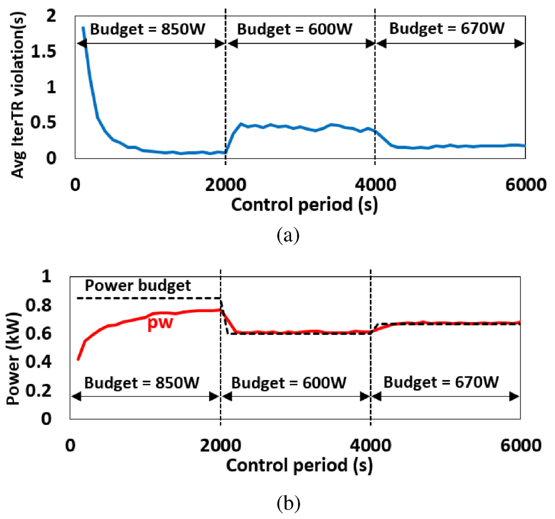


Fig. 6. Control accuracy given dynamic power budget. (a) Average IterTR violation curve. (b) Total power consumption curve.

(13). In addition, we consider three types of fixed GPU frequency settings, i.e., minimum, medium, and maximum types, as the baseline. For the minimum and maximum types, we use frequency ranges shown in Table I. For medium setting, we used {core 1000Mhz, mem 3500Mhz} for GTX1060, and {core 1000Mhz, mem 4400Mhz} for GTX1080, respectively.

B. Evaluation

Fig. 5 shows the control accuracy of the proposed CD-GPC system in terms of the iteration time requirement violation and the power budget compliance. We consider only three DL tasks, i.e., a1, a4, and a11 (see Table II) for readability. Even

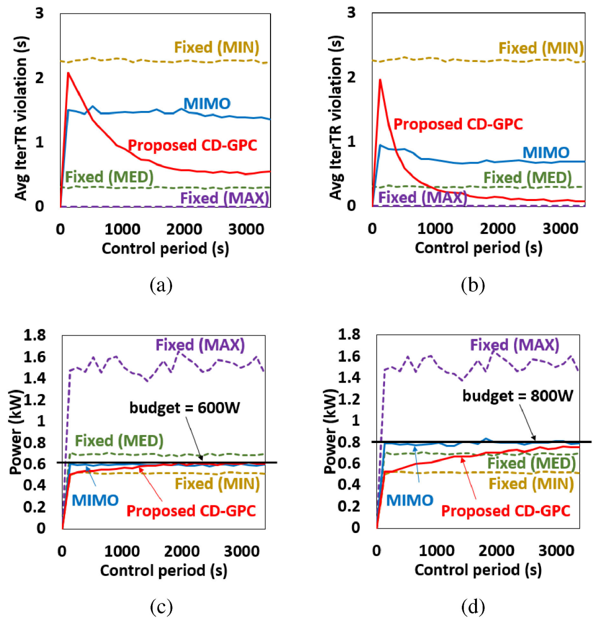


Fig. 7. Performance comparison of the CD-GPC method, fixed GPU frequency setting approach, and MIMO control [3]. (a) At $pb = 600$ W. (b) At $pb = 800$ W. (c) At $pb = 600$ W. (d) At $pb = 800$ W.

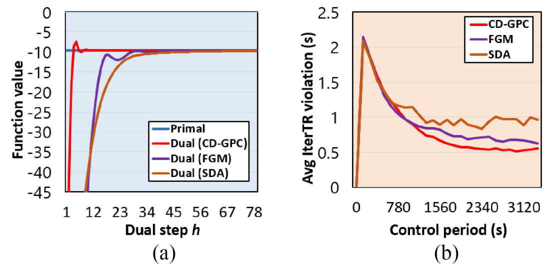


Fig. 8. Comparison of dual convergence rate. (1) CD-GPC (using tight Lipschitz constant). (2) Fast gradient method (FGM) [15]. (3) Standard decomposition algorithm (SDA) [26]. (a) Optimal Primal function value versus dual function value. (b) Average IterTR violation curve.

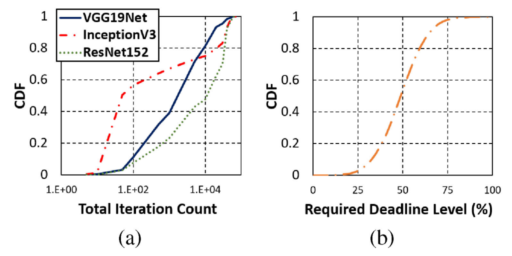


Fig. 9. (a) Cumulative distribution of the total iteration count of generated DL tasks based on Microsoft real job-trace [21] (b) Cumulative distribution of the required DL task deadline based on *Gaussian* distribution (avg = 50%, std = 7.5%).

under the low power budget, i.e., $pb = 600$ W, our CD-GPC system enables minimizing the violation of the iteration time requirement as shown in Fig. 5(a). Unsurprisingly, under the sufficient power budget, i.e., $pb = 800$ W, the CD-GPC system achieves nonviolation of the iteration time requirement after a control period $2100s+$ as shown in Fig. 5(b).

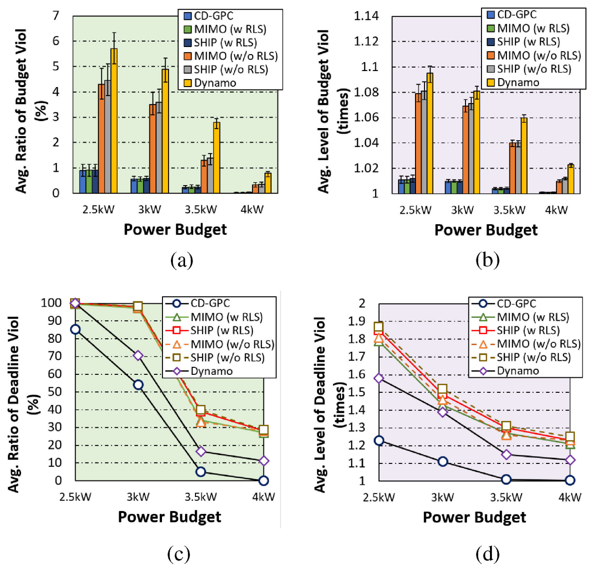


Fig. 10. Performance comparison of the proposed CD-GPC, MIMO [3], SHIP [1], and Dynamo [27], in terms of (a) average time slot ratio of budget violation (b) average level of budget violation (c) average time slot ratio of deadline violation (d) average level of deadline violation.

Note that in both cases, our controller can finely tune the GPU frequencies toward the target iteration time requirement and power budget before that the control period reaches 500 s. Without offline profiling information, our proposed RLS-based online model estimator accurately captures the model coefficients for different DL tasks upon heterogeneous GPU servers. Fig. 5(c) shows that the proposed CD-GPC system efficiently minimizes the performance degradation by maximally using the limited power budget. Fig. 5(d) shows that the proposed CD-GPC system can always ensure the power consumption does not exceed the given power budget.

Fig. 6 shows the CD-GPC control accuracy under a dynamic power budget. The power budget pb is set to 850, 600, and 670 W in control period intervals of $[0\,\mathrm{s}, 2000\,\mathrm{s}]$, $[2001\,\mathrm{s}, 4000\,\mathrm{s}]$, and $[4001\,\mathrm{s}, 6000\,\mathrm{s}]$, respectively. From Fig. 6(a) and (b), we observe that the CD-GPC can accurately scale the GPU frequencies adapting to the unpredictable changes of the power budget. Fig. 6(b) shows the power consumption dynamically drops at 2100 s and smoothly increases at 4100 s. This is because our system considers the power budget as a hard constraint. The associated control is more sensitive to the power budget than to the iteration time, indicating that the CD-GPC system is suitable for GPU clusters in which peak power management is critical.

Fig. 7 compares the performance among the proposed CD-GPC, MIMO power control, and fixed GPU frequency. Fig. 7(c) shows that the MIMO method approaches the power budget (600 W at 13 s) faster than the CD-GPC system does. However, its average violation for iteration time requirement of 1.5 s is much higher than that of the CD-GPS, i.e., 0.5 s. This is because the MIMO mainly focuses on the power budget, and does not explicitly consider the deadline of running DL tasks. The approach using fixed GPU frequency performs the worst. Even though the power consumption of fixed setting "MAX" is more than twice of the proposed CD-GPC, their derived iteration times are almost the same as shown in Fig. 7(b) and (d). Comprehensively, the results indicate that our EMPC formulation in the CD-GPC system enables efficient best-effort use of the given power budget.

Fig. 8 compares the dual convergence rate of the proposed CD-GPC with other dual variable updating schemes when the power budge is $pb = 600$ w. In particular, we compare the proposed CD-GPC system with the standard decomposition algorithm (SDA) [15] and the fast gradient method (FGM) [26]. The SDA uses an arbitrary small step size ϵ for dual updating without considering the Lipschitz continuity. We set $\epsilon = 0.1$. The SDA cannot ensure a certain dual convergence rate. In contrast, the FGM uses the Lipschitz constant

$$L' = \frac{1}{\sigma} * \max_{\forall h\in[2N\cdot H^c]} \sum_{l=1}^{H^c} |A^p_{lh}| * \max_{\forall l\in[H^c]} \sum_{h=1}^{2N\cdot H^c} |A^p_{lh}| \qquad (26)$$

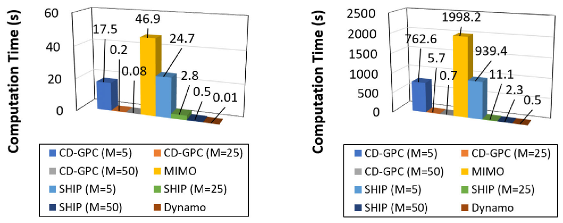
where $N = 2\sum_{i=1}^{M} N^i$, σ, and A^p_{lh} denote the total number of GPU servers, the strongly convex parameter, and the hth column value of lth row in $\mathbf{A}^p$, respectively. The dual convergence rate of the FGM is slower than the CD-GPC since L' does not reflect the curvatures of $\mathbf{H}(k)$ in different directions.

Fig. 8(a) depicts the gap between the optimal primal function value $J^*(k)$ and the iterative dual function value $D(\gamma)$. We can see that while the SDA and the FGM require the nonnegligible number of dual steps, i.e., $h > 35$, to reach the optimal point, i.e., min $J = $ max D, the proposed CD-GPC reaches the optimal point within only seven dual steps, i.e., $h = 7$. Moreover, the violation of the iteration time requirement of the CD-GPC decreases faster than the other dual variable updating schemes, i.e., FGM and SDA, as shown in Fig. 8(b).

VII. Large-Scale Simulation Experiments

To evaluate the performance of the proposed CD-GPC in large-scale GPU clusters, we conduct the simulation for GPU clusters consisting of 50 and 200 GPU servers, i.e., $N = 50$ and $N = 200$, respectively. The simulation settings on servers and DL tasks are shown in Tables I and II for both cases. Besides, we adopt three DNN models, i.e., VGG19Net, InceptionV3, and ResNet152, for DL task invocation and exploit the workload characteristics of the Microsoft real job-trace [21]. We generate the total iteration count for each DL task as shown in Fig. 9(a) based on the distribution of job run-time. The deadline of each DL task is generated based on the *Gaussian* distribution (avg $=$ 50%, std $= 7.5\%$) in the range of [0, 1], indicating how fast the DL task is completed. Fig. 9(b) shows the deadline distributions.

We compare the CD-GPC with two existing power capping methods, i.e., SHIP [1] and Dynamo [27], and MIMO. SHIP is a hierarchical version of MIMO and Dynamo adopts the distributed capping structure including multiple rack controllers. In SHIP, a two-level control problem, saying the power distribution unit (PDU) level and rack level, are solved in every control period. Dynamo avoids the power dissipation under an overprovisioned power budget, by flexibly making the power capping and/or uncapping decisions.



(a) (b)

Fig. 11. Control computation time for (a) # of GPU servers = 50 (b) # of GPU servers = 200.

We run the simulation instance ten times and assign 2880 control time slots to each instance, with one minute per slot. Fig. 10(a) shows the ratio of time slots that violate the power budget. Due to the online model estimation by the RLS method, the CD-GPC system achieves a lower violation ratio (≈0.09 at 2.5 kW) than other methods. Interestingly, MIMO and SHIP using the RLS method also derive the low ratio of the budget violation, similar to the CD-GPC system. Note that MIMO and SHIP without the RLS method derive the high ratio (≈0.45 at 2.5 kW) of the budget violation because they use the inaccurate power model coefficients based on the static offline regression. Dynamo derives the worst budget violation (≈5.7 at 2.5 kW) because it uses simple heuristics for power control instead of using elaborate theoretical models. Fig. 10(b) shows the average level of the budget violation (the ratio of the amount of power consumption to the budget at budget-violated time slots) of which trends are similar to Fig. 10(a). Fig. 10(c) shows the average ratio of the deadline violation. The CD-GPC system derives the lowest violation ratio (36% on average) due to the IterTR-based control. The MIMO and SHIP derive the high violation ratio (65% on average) because they adjust the frequency only based on the server utilization. The Dynamo shows better performance (49% on average) than the MIMO and SHIP. This is because the Dynamo periodically redistributes the power budget to the servers according to the priority of the DL tasks. Fig. 10(d) shows the average level of the deadline violation (the ratio of the completion time to the deadline for deadline-violated DL tasks) of which trends are similar to Fig. 10(c).

Fig. 11 shows the control computation time. The MIMO requires high computation time for both cases ($N = 50, 200$) because the inverse computation burden in (13) is exponentially increased with the increasing H. In contrast, the CD-GPC ($M = 50$) needs the trivial computation time (0.08 s for $N = 50$, 0.7 s for $N = 200$). This is because the computation burden of the CD-GPC system can be dispersed by the local controllers. Note that the CD-GPC ($M = 5$) requires the nonnegligible high computation time (762.6 s for $N = 200$). This is because the number of local controllers is not enough to sufficiently reduce the dimensionality of each subcontrol problem. Nonetheless, the computation time of CD-GPC can be further reduced by simply increasing the number of local controllers. We observe that the computation time can be decreased by 1070 times (762.6 s $\rightarrow$ 0.7 s) when M is increased by just ten times (5 $\rightarrow$ 50). Dynamo shows the improved computation time compared to our CD-GPC system. However, the rapid computation of Dynamo is due to its simple heuristics at the expense of elaborate control. As shown in Fig. 10, our CD-GPC system outperforms Dynamo in terms of power budget and deadline assurance, at the trivial cost of the slightly increased control computation time.

VIII. Conclusion

In this article, we proposed a CD-GPC system for DL task processing so as to minimize the training completion time without exceeding the limited power budget. To the best of our knowledge, this is the first study to explore the device-agnostic and scalable power control for heterogeneous GPU servers handling DL tasks. Through our presented RLS model estimation-based GPU FS method, we achieved accurate power capping without needing the offline-based profile operation. Our sophisticated EMPC formulation based on the tight Lipschitz continuous Lagrangian dual decomposition efficiently accelerated the convergence to the optimal control. The experimental results based on both the NVIDIA GPU based lab-scale real experiments and large-scale simulation with job-trace demonstrated that the CD-GPC is the promising candidate for the power control system of modern GPU clusters handling DL tasks.

References

[1] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller, “Ship: A scalable hierarchical power control architecture for large-scale data centers,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 1, pp. 158–176, Jan. 2012.

[2] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, ““No” power” struggles: Coordinated multi-level power management for the data center,” in *Proc. 13th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2008, pp. 48–59.

[3] X. Wang, M. Chen, and X. Fu, “MIMO power control for high-density servers in an enclosure,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 10, pp. 1412–1426, Oct. 2010.

[4] J. Yao, X. Liu, W. He, and A. Rahman, “Dynamic control of electricity cost with power demand smoothing and peak shaving for distributed internet data centers,” in *Proc. 32nd Int. Conf. Distrib. Comput. Syst.*, 2012, pp. 416–424.

[5] Q. Wang and X. Chu, “GPGPU performance estimation with core and memory frequency scaling,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 12, pp. 2865–2881, Dec. 2020.

[6] X. Mei, X. Chu, H. Liu, Y.-W. Leung, and Z. Li, “Energy efficient real-time task scheduling on CPU-GPU hybrid clusters,” in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2017, pp. 1–9.

[7] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas, “GPGPU power modeling for multi-domain voltage-frequency scaling,” in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2018, pp. 789–800.

[8] S. Chaudhary, R. Ramjee, M. Sivathanu, N. Kwatra, and S. Viswanatha, “Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning,” in *Proc. 15th Eur. Conf. Comput. Syst.*, 2020, Art. no. 1.

[9] R. A. Bridges, N. Imam, and T. M. Mintz, “Understanding GPU power: A survey of profiling, modeling, and simulation methods,” *ACM Comput. Surv.*, vol. 49, no. 3, pp. 1–27, Dec. 2016.

[10] Z. Tang, Y. Wang, Q. Wang, and X. Chu, “The impact of GPU DVFs on the energy and performance of deep learning: An empirical study,” in *Proc. 10th ACM Int. Conf. Future Energy Syst.*, 2019, pp. 315–325.

[11] Y. Wang *et al.*, “Benchmarking the performance and energy efficiency of AI accelerators for ai training,” in *Proc. 20th IEEE/ACM Int. Symp. Cluster Cloud Internet Comput.*, 2020, pp. 744–751.

[12] Y. Jianguo, L. Xue, G. Zonghua, W. Xiaorui, and L. Jian, “Online adaptive utilization control for real-time embedded multiprocessor systems,” *J. Syst. Architecture*, vol. 56, no. 9, pp. 463–473, Sep. 2010.

[13] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres, “Power and performance characterization and modeling of GPU-accelerated systems,” in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, 2014, pp. 113–122.

[14] M. Ellis, H. Durand, and P. D. Christofides, "A tutorial review of economic model predictive control methods," *J. Process Control*, vol. 24, no. 8, pp. 1156–1178, Aug. 2014.

[15] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1439–1451, Aug. 2006.

[16] P. Giselsson, "Improved dual decomposition for distributed model predictive control," *IFAC Proc.* vol. 47, no. 3, pp. 1203–1209, 2014.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[18] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 2818–2826.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 770–778.

[20] S. Karen and Z. Andrew, "Very deep convolutional networks for large-scale image recognition," in *Proc. 6th Int. Conf. Learn. Representations*, 2015.

[21] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant GPU clusters for DNN training workloads," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 947–960.

[22] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[23] F. Yan, O. Ruwase, Y. He, and E. Smirni, "SERF: Efficient scheduling for fast deep neural network serving via judicious parallelism," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2016, pp. 300–311.

[24] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[25] F. Seide and A. Agarwal, "CNTK: Microsoft's open-source deep-learning toolkit," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery and Data Mining*, pp. 2135–2135. 2016, [Online]. Available: https://github.com/Microsoft/CNTK.

[26] A. Beck, A. Nedic, A. Ozdaglar, and M. Teboulle, "An $o(1/k)$ gradient method for network resource allocation problems," *IEEE Control Netw. Syst.*, vol. 1, no. 1, pp. 64–73, Mar. 2014.

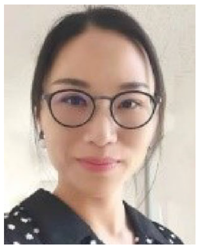
[27] Q. Wu *et al.*, "Dynamo: Facebook's data center-wide power management system," *ACM SIGARCH Comput. Architecture News*, vol. 44, no. 3, pp. 469–480, 2016.

Dong-Ki Kang received the B.S. and M.S. degrees in computer engineering from Jeonbuk National University, Jeonju, South Korea, in 2009 and 2011, respectively, and the Ph.D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2019.

From 2019 to 2020, he was a Postdoctoral Fellow with Prof. Youn's Group, KAIST. He is currently an Assistant Professor with the Department of Information Technology, Jeonbuk National University. His research interests include cloud computing, data center management, GPU management, deep learning, and distributed optimization.

Yun-Gi Ha received the B.S. degree in electronic and electrical engineering from Sungkyunkwan University, Suwon, Korea, in 2013 and the M.S. degree in electrical engineering from KAIST, Daejeon, Korea, in 2014. Currently, he is a Ph.D. Candidate in electrical engineering at KAIST.

His research interests include online learning and efficient resource management in heterogeneous computing environment.



Limei Peng received the B.S degree from the Department of Computer Science and Technology, South-Central University for Nationalities, China, in Sep. 2004, the M.S. and Ph.D. degrees from the Department of Computer Engineering, Jeonbuk National University, Republic of Korea, in Aug. 2006 and Feb. 2010, respectively. She is currently an Associate Professor at the School of Computer Science and Engineering, Kyungpook National University (KNU), Daegu, Republic of Korea. Before joining KNU, she was an Assistant Professor in the Department of Industrial Engineering, Ajou University, Suwon, Republic of Korea, from 2014 to 2018, and an Associate Professor in Soochow University, China, from 2011 to 2013. Her research interests include cloud computing, fog computing, datacenter networks, IoT/IoV, and 5G communications networks.

Chan-Hyun Youn (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in electronics engineering from Kyungpook National University, Daegu, South Korea, in 1981 and 1985, respectively, and the Ph.D. degree in electrical and communications engineering from Tohoku University, Sendai, Japan, in 1994.

Before joining the University, from 1986 to 1997, he was the Leader of High-Speed Networking Team, KT Telecommunications Network Research Laboratories, where he had been involved in the research and developments of centralized switching maintenance system, high-speed networking, and ATM network. Since 2009, he has been a Professor with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea. From 2013 to 2017, he was an Associate Vice-President of office of planning and budgets with KAIST. He is also the Director of the Grid Middleware Research Center and the XAI Acceleration Technology Research Center, KAIST, where he is developing core technologies that are in the areas of high-performance computing, edge-cloud computing, AI acceleration system, and others. He has authored a book *Cloud Broker and Cloudlet for Workflow Scheduling* (Springer, 2017).

Dr. Youn was the General Chair of the 6th EAI International Conference on Cloud Computing (Cloud Comp 2015), KAIST, in 2015 and the Guest Editor for the IEEE WIRELESS COMMUNICATIONS in 2016. He was a TPC Member for many international conferences.