IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 30, 2020, accepted October 13, 2020, date of publication October 26, 2020, date of current version November 9, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3033771

An Accelerated Edge Cloud System for Energy Data Stream Processing Based on Adaptive Incremental Deep Learning Scheme

SEONG-HWAN KIM, (Associate Member, IEEE), CHANGHA LEE, AND CHAN-HYUN YOUN, (Senior Member, IEEE)

Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 305-701, South Korea

Corresponding author: Chan-Hyun Youn (chyoun@kaist.ac.kr)

This work was supported in part by Korea Electric Power Corporation under Grant R18XA05, and in part by the Electronics and Communications Research Institute (ETRI) and the National Research Council of Science and Technology (NST) Grant funded by the Korea Government [Ministry of Science and ICTA (MSIT)] (Research on Foundation Technology of IDX Platform) under Grant 19ZS1200.

ABSTRACT As smart metering technology evolves, power suppliers can make low-cost, low-risk estimation of customer-side power consumption by analyzing energy demand data collected in real-time. With advances network infrastructure, smart sensors, and various monitoring technologies, a standardized energy metering infrastructure, called advanced metering infrastructure (AMI), has been introduced and deployed to urban households to allow them to develop efficient power generation plans. Compared to traditional stochastic approaches for time-series data analysis, deep-learning methods have shown superior accuracy on many prediction applications. Because smart meters and infrastructure monitors produce a series of measurements over time, a large amount of data is accumulated, creating a large data stream, which takes much time from data generation to deployment of deep-learning model training. In this article, we propose an accelerated computing system that considers time-variant properties for accurate prediction of energy demand by processing the AMI stream data. The proposed system is a real-time training/inference system that deploys AMI data over a distributed edge cloud. It comprises two core components: an adaptive incremental learning solver and deep-learning acceleration with FPGA-GPU resource scheduling. An adaptive incremental learning scheme adjusts the batch/epoch in training iteration to reduce the time delay of the latest trained model, while trying to prevent biased-training due to the sub-optimal optimizer of incremental learning. In addition, a resource scheduling scheme manages various accelerator resources for accelerated deep-learning processing while minimizing the computational cost. The experimental results demonstrated that our method achieved good performance for adaptive batch size and epoch for incremental learning while guaranteeing a low inference error, a high model score, and queue stability with cost efficient processing.

INDEX TERMS Accelerator scheduling, continual learning, deep learning acceleration, stream data processing, time-series prediction, incremental learning, hyperparameter tuning.

I. INTRODUCTION

As the global energy demand of countless electronic devices increases rapidly, many researchers are paying close attention to energy data analysis to reduce energy waste. Sensing devices throughout power systems, such as smart meters and infrastructure monitors, generate a sequence of measured values over time, and huge amounts of data accumulate, resulting in large data streams. In response to this massive amount of data being stored and processed, meaningful information and hidden patterns are extracted from distributed and heterogeneous sensors that produce large data streams. The information and patterns can be used to determine the amount of energy to be harvested and stored in a stable manner.

Advanced Metering Infrastructure (AMI) is an integrated system that enables two-way communication between utilities and customers. IndustryARC research shows that the number of AMIs will continue to grow as the industrial

The associate editor coordinating the review of this manuscript and approving it for publication was Dominik Strzalka.

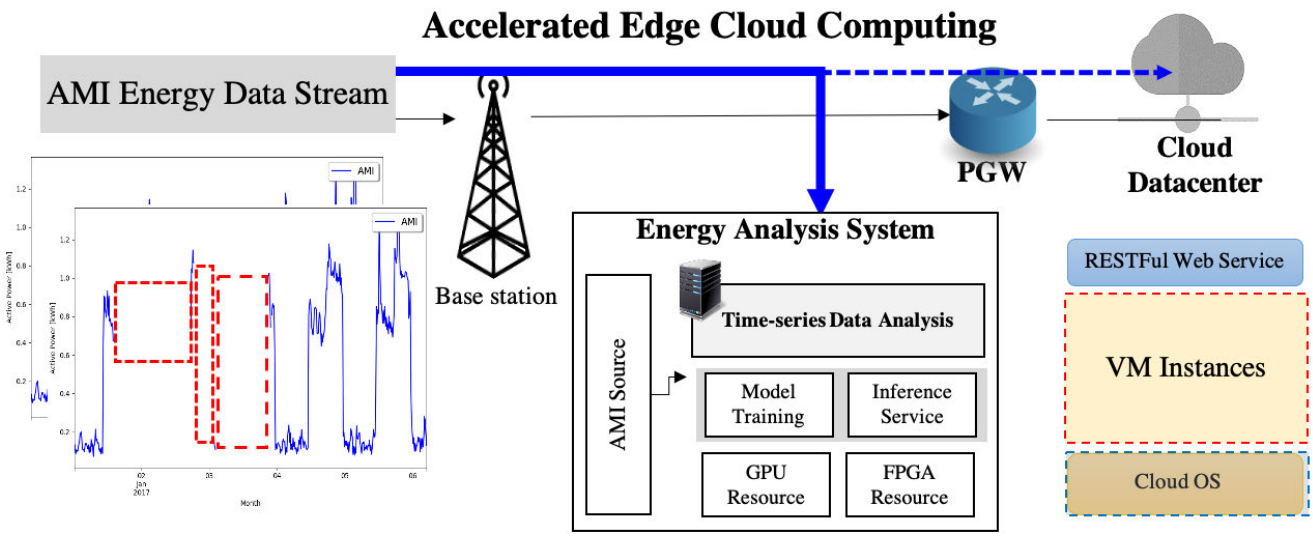

FIGURE 1. Overall configuration of energy stream processing system in edge-cloud environment.

IoT market is expected to reach $ 123.89 billion by 2021. By 2020, Korea Electric Power Corporation (KEPCO) will expand its AMI equipment to 22.5 million customers nationwide. Based on the distribution of 2.5 million AMI devices in 16 years, the data storage capacity is about 68 TB, and the cumulative data is expected to be about 700 TB when 22 million units are supplied [1]. AMIs provide power utility companies with real-time power consumption data and allow customers to choose information about their energy usage based on price during the day. Compared to traditional data, stream data are not bounded in size, time-series data, and a sequence of successive data points include the characteristics of the number of passes, memory usage, heterogeneity, and evolving properties in time order. In addition, with these growing applications of analyzing time series data using massive data stream to reveal future insights, a scheme to extract the information from distributed and heterogeneous source has become a challenging task. Therefore, there are needs to analyze the massive amount of streaming data from the variety AMI. The patterns of one sensor can be predicted by comparison with renewable energy data, while the other sensor shows a completely different aspect.

In particular, next-generation energy systems manage customer demand through complex event processing. Increased adoption of deep learning models has led to better practices in energy demand/supply forecasting and energy services. Energy services, such as power resource provisioning, electricity pricing, and energy storage charging/discharging, requires prediction for input stream data. Prediction of various factors for the input stream, such as energy demand, is a complex multi-variate time-series function with multiple classes of data. In particular, more accurate and faster forecasting of energy demand and supply has become essential to enable the provision of a stable energy supply and to reduce energy costs. In addition, model accuracy for energy demand prediction has become essential to provide a stable energy supply and reduce energy costs. Therefore, there are huge requirements for the low risk system with the acceleration of stream data processing framework while providing excellent accuracy [2]. Especially, Renewable Energy Sources such as solar, wind, and geothermal power generation can make effective energy management difficult due to intermittence/unpredictability in their production, and can lead to big accidents such as blackouts and fires if the failure of an immediate response in the detection of power plant failures [3]. Therefore, energy demand/supply prediction is being researched as a core technology of energy system. In addition, this is an important task to minimize the maintenance cost of smart home/building/city, and maximize the profit of the utility company. In the field of time-varying energy data stream analysis, artificial intelligence such as deep learning has been reported to outperform traditional time series prediction schemes (e.g., ARIMA) [4].

In addition, traditional training method that repeats the same model update operation on stream data sets that accumulate over time, so that existing distributed deep learning computing frameworks can repeat the entire computation of the entire data set. However, it is inefficient and wasteful of computational resources, and the retraining period gradually increased due to the increased training time for the accumulated data set, resulting in reduced predictive performance for short-term non-stationary AMI data. At the same time, frequent retraining and deployment of trained models for data freshness can lead to cost performance degradation due to the huge overhead of model initialization, worker initialization, memory access, and so forth. Therefore, low cost system with accelerated stream data processing while providing excellent accuracy is also required [5]

As smart meters and infrastructure monitors produce a series of measurements over time, a large amount of data is accumulated, creating a large data stream. A large amount of data collection/refining/processing is required for deep learning, which takes much time from data generation to the deployment of deep-learning model training [6]. The challenges for the systems involve limitations of storage, memory, and computing capacity for data processing. Therefore, the processing of big stream data cannot be handled by traditional analysis systems, and there is a need for efficient analysis systems [5]. Compared to traditional data, stream data are not bounded in size; rather they are time-series data, and a sequence of successive data points include the characteristics of the number of passes, memory usage, heterogeneity, and evolving properties in time order [6]. In addition, with these growing applications for the analysis of time-series data using unbound data streams to reveal insights to guide predictions, a technique to extract the information from distributed and heterogeneous sources has become a challenging task. Incremental learning (or online learning) is a machine-learning method that trains an analytic model through stream data consisting of time-ordered data instances. Compared with offline training, the model updates in incremental learning are much lighter because they only train a dataset with one or a few instances at a time. The method of stochastic analysis with cost-efficient computing can provide low-latency model training without a delay issue for service model deployment [7]. As result, there are an framework for incremental learning on common deep learning tasks with the consideration of sophisticated epoch and batch scheduling [3].

In this article, to accelerate the deployment procedure of a deep neural network after model training, we propose an accelerated edge cloud computing method for energy data stream processing based on an incremental deep learning scheme. The incremental learning scheme can perform real-time learning by performing learning using only the queue model of the latest learning data. In addition, a model recency metric with the profile of DL operation is proposed to minimize the latency of given input AMI data to be trained. Furthermore, we consider real-time processing to search for the optimal solution of multi-criteria utility functions. The system determines the number of data instances and the number of epochs for temporary mini-batch training to reduce the amount of processing time and computational cost required for learning while retraining the model to reflect new features in recently incoming data stream immediately.

We summarize our contributions as follows:

1) We resolve the parameter update scheduling problem to reflect short-term non-stationary AMI data at a low latency while minimizing the degradation of prediction performance that can result from the use of a partial training dataset.

2) We propose a utility function for adaptive incremental deep learning to improve model accuracy through rapid reflection of changes in concept drift and to increase the learning speed in batch learning. In addition, we propose a heuristic to quickly find the decision vector that satisfies multivariate optimization of the utility function and reduces the overhead in each training iteration.

3) We implemented heterogeneous accelerator (FPGU, GPU) resource scheduling through layer partitioning in the edge cloud.

II. RELATED WORK

In this section, we briefly introduce the incremental deep learning scheme, debate characteristics of data streams, and discuss the challenges of resource scheduling of heterogeneous accelerators in the edge cloud system.

A. TIME-VARYING CHARACTERISTIC OF AMI STREAM DATA

In predictive analytics and machine learning, the conceptual drift is a phenomenon in which the statistical characteristics of the modeling variable change over time. In data streaming, which is the theoretically-infinite set of data, it is a challenging issue to solve concept drift in which the statistical properties of data change [5]. The concept drift is a notion of a change in the probability distribution of a dataset, such as time-evolving characteristic [5] and class-wise characteristic [8], etc. Stream data distribution can be time-varying, result in a model trained on historical data being inconsistent with the new data, which reduces the accuracy of the prediction system. In case, incremental learning is a way to solve the problem by regularly updating the model for recent data [9], [10].

AMI data [13] is composed of power demand data from multiple customers over the yearly collected, which might have both time-evolving and class-wise concept drift. Therefore, we measured the characteristics of AMI data. To evaluate the characteristics of the AMI data, we obtained the energy demand data collected by the Korea Electric Power Corporation (KEPCO) measured in 2017 in Jeon-nam, Korea, provided for research purposes only [14]. It is household power consumption data collected every 15 minutes from January to December. Firstly, we use Cosine Similarity Score (CSS) [11] to measure the distance between two datasets for the monthly collected AMI from January to December in 2017, Korea. The cosine similarity score is similarity between two non-zero vectors measured by cosine in inner product space that is a distance in frequency domain within $-1 \geq CSS \geq 1$.

$$CSS(A, B) = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{A}\sqrt{B}} \tag{1}$$

where, A and B are vectors, and A_i and B_i are components of vector A and B respectively.

We measured average cosine similarity with AMI data randomly sampled monthly from January to December, with January as the basis of comparison given as v_{basis}. Let k-th randomly sampled monthly household data of User j as vector $v_{j,k}$. Then, a set of randomly sampled vectors is denoted as $V_j = \{v_{j,k} | k = 1, \ldots, K\}$, where K is number of monthly

samples.

$$AverageCSS(V_j) = \frac{\sum_{v \in V_j} CSS(v_{basis}, v)}{|V_j|} \quad (2)$$

From the January to December, we generated random samples as $V_j^{Jan}, V_j^{Fab}, \ldots, V_j^{Dec}$ and measured Average CSS for each vector set. We figured out that average cosine similarity of power demand pattern continuously decreases over time. The reduction in cosine similarity score means that the properties of the AMI data change over time and have further distance than previous datasets as shown in Table 1. From the result, AMI data has time-varying feature and pre-trained model on previous data can be out-of-date.

TABLE 1. Cosine similarity (Eq. (2), [11]) of AMI data at 2-month intervals from January to December 2017 (only for research purposes from KEPCO).

Target month	Jan	Mar	May	Jul	Sep	Nov
Cosine similarity	0.71	0.56	0.63	0.67	0.63	0.56

In addition, Table 2 shows test statistic and 5% of critical value of the Augmented-Dickey-Fuller test on the AMI dataset with respect to AMI Pattern Range. The Augmented-Dickey-Fuller (ADF) test [12] is a type of statistical analysis called a unit root test to test how strongly a time series is defined by a trend. With the measurement, we can determine that time-series data is stationary or not. If the test statistic value is lower than 5% of critical value, then, it means that the time-series data is stationary with 95% confidence. The results indicate that the smaller or larger AMI Pattern Range show stationary characteristics and the other is non-stationary. Since the AMI data has shown the time-dependence, and we need to train deep learning models due to the nature of non-stationary energy data streams [9], [10].

TABLE 2. Augmented-dickey-fuller test [12] results for annual AMI data (only for research purposes from KEPCO).

AMI Pattern Range	10	36	100	1000	10000
Test Statistic	-4.55	-1.59	-2.66	-2.78	-14.65
Critical Value (5%)	-3.29	-2.96	-2.89	-2.86	-2.86

B. AN ONLINE LEARNING FRAMEWORK

Incremental learning is a machine learning method that trains an analytic model through stream data consisting of time-ordered data instances. Because stream data is theoretically defined as an infinite set of data instances, the system should process continuous, high-volume, open-ended data streams as they arrived [15]. As we cannot have infinite computing resources, and it is not cost-efficient, we could consider one-shot learning for the short-term of incoming data at each step. Therefore, several data instances for training should be processed finitely and should be discarded without being stored in memory. For the cost-efficient parallel processing, training data could be consisting of bunches, but the number of maximum instances should be constant without increasing indefinitely over time. Besides, the time required to process each instance should be small and constant over time for efficient training [5]. Compared with offline training, model updates in incremental learning are much lighter because they only train dataset with basically one or small batch instances at a time. Incremental learning can be also applied for reinforcement learning (RL) for their processing acceleration [16]. Different from traditional batch learning algorithms, they proposed novel incremental learning scheme for RL with concrete convergence and robustness. As a result, the incremental learning with cost-efficient computing can provide low latency model training without straggling issue for service model deploy [7].

Velox [17] is an efficient data management system for implementing large-scale analytic pipeline for an online model management, maintenance for deployed model, and model serving which are model based data analysis procedure. For more specific example for machine learning framework, Clipper [18], as one of Velox system, is a general-purpose low-latency prediction serving system. Clipper propose a standard platform architecture to simplify model deployment for serving at various frameworks, while arbitrating between trained model serving and varying deep neural network frameworks. Therefore, new machine learning frameworks and models can be introduced without modifying end-user applications. The model selection layer simultaneously deploys many models and then dynamically selects and aggregates forecasts for each model to render more powerful, accurate, and contextualized forecasts while reducing the straggler cost. In conclusion, Clipper introduces caching, layout, and adaptive model selection technologies to reduce forecast latency and improve predictive throughput, accuracy, and robustness without modifying the underlying machine learning framework. However, Clipper has limitations, such as not managing the training or retraining of the base model within each framework. All models are outdated or inaccurate, and Clipper cannot improve the accuracy beyond what can be achieved through an ensemble. In addition, as Clipper does not optimize model execution within its machine learning framework, the latency or execution time has dependent on processing performance of backend framework.

For the incremental deep learning model with classification problem, the objective is to learn a function $F : \mathbb{R}^d \rightarrow \mathbb{R}^c$ based on a sequence of training example $D = (x_1, y_1), \ldots, (x_T, y_T)$, that arrive sequentially at each timestep, where $x_t \in \mathbb{R}^d$ is d-dimensional instance representing the features and $y_t \in \{0, 1\}^C$ is the class label assigned to x_t and C is finite number of classes. The prediction is denoted by $\hat{y}_t$, and the performance of the learnt functions are usually evaluated based on the cumulative prediction error: $\epsilon_T = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}(\hat{y}_t, y_t)$, where $\mathcal{L}$ is the loss function that defined as in 1 if the given value is same, and 0 otherwise. Let assume b as the size of temporary mini-batch given in time t to train the model. With the mini-batch SGD algorithm [19], the incremental deep learning can be derived

as follows:

$$\theta_{t+b} \leftarrow \theta_t - \frac{\eta}{b}\nabla_{\theta_t}\sum_i \mathcal{L}(F(x_{t+i};\theta_t), y_{t+i}) \quad \forall i = 1, \ldots b \tag{3}$$

where, θ is model parameter and η is training rate.

As the model cannot be trained over an entire data set, a sub-optimal predictor of incremental learning cannot gain full information. As a result, the performance of prediction is not so much accurate as batch learning. An adaptive assignment of hyperparameter to incremental deep learning scheme is considered in this article for the robust and fast energy data stream processing.

For incremental learning tasks, it is important to determine the retraining time to improve the data integration latency, which selects the best time instance to perform a given task, such as updating the model. At the same time, frequent retraining and deployment of latest trained models can lead to cost performance degradation due to the huge overhead of model initialization, worker initialization, memory access, etc. Finally, there is a meta-value called hyperparameters in the common framework of machine learning, which yields an optimal model by minimizing predefined loss functions in a given independent data set [7]. Among the hyperparameters, there are two main control variables for model training. It is used interchangeably with the number of instances training the stream dataset, the size of the batch, and the number of passes in a given training stream set is called an epoch in this chapter. Training with small batches is usually vulnerable to outliers (or noisy data). Also, large numbers of epochs generally help to find good predictors before the model is overfitted, while larger epochs require longer training time.

C. RESOURCE SCHEDULING SCHEME FOR ACCELERATED EDGE CLOUD

The Graphics Processing Unit (GPU) is a structure optimized for matrix operations and operates in a single instruction multiple data (SIMD) scheme, making it ideal for quickly processing DL training that repeatedly performs the same task on a large amount of data. Unlike general-purpose processors (GPPs) such as GPUs and CPUs with pre-synthesized general-purpose cores, the logic functions of FPGAs can be changed by user design. While FPGA is reconfigurable, in the case of GPP, the structure of the core logic is fixed, so that the logic can not be changed according to the needs of a particular application. In contrast, FPGAs can change logic functions to favor an application, modern FPGA-based acceleration can guarantee higher processing performance than GPP, especially on utilization, power efficiency and latency for Real-time AI [20]. Also, unlike other processors operated by operation set, FPGA can solve synchronization according to timing problem by specifying function at gate-level, so it can accelerate specific operation like Sparse Matrix Multiplication [21] or design like BNN [22], TNN [23], etc. Accelerated learning by implementing a compressed neural network. However, there is a context switch overhead that must be continuously performed while data input/output (DMA) model training is performed from the main host memory space to the co-processor processing space.

There have been many studies on scheduling schemes to improve energy efficiency and execution speed when executing specific tasks in consideration of task partition in the FPGA-GPU hybrid system [24], [25]. In particular, Rethinagiri *et al.* [25] have applied task splitting to face recognition applications based on LBHP algorithms and found a 40% increase in energy efficiency compared to traditional GPUs. Partitioning is done in a way that maximizes efficiency, taking into account the energy efficiency of each subtask. In Inta *et al.* [24], task division is applied to a variety of compute-intensive applications such as Monte Carlo Integration, LAPACK, and wavelet decomposition. Partitioning is applied to reduce execution time, taking into account the running characteristics of each subtask.

III. A MODEL DESCRIPTION OF STREAM DATA PROCESSING SYSTEM WITH THE ADAPTIVE INCREMENTAL LEARNING SCHEME

A. ARCHITECTURE AND FUNCTIONAL DESCRIPTION

In this section we present the architecture and procedure of the proposed incremental learning scheme with streamlines of continual updating in the deep learning model as shown in Fig. 2.

1) DEEP LEARNING PREDICTION MODULE

Time-series data sets can be converted to supervised training data sets as further data arrived in the future timestamp, which mean that we can know the ground truth values for the prediction target. The training loss value on the prediction model of the i-th value can be represented $\mathcal{L}(y_i, y_i^*)$ where y_i is the i-th predicted value and y_i^* is the i-th measured value. In incremental learning, pre-trained data instances are discarded without being stored on memory or storage. Unlike batch/mini batch training, which trains with multiple epochs for a given entire data set, incremental learning can only train with temporary mini batches.

2) DATA STREAM HANDLER

The process of converting a data sequence into an input tailored to the size of the convolutional layer's input dimension is required for the data sequence. After the pre-processing of the input data stream, the data is refined with their arrival sequence. Here, k-th values grouped in X and Y are referred as an data instance d_k. An arrived time of d_k is given to a_k.

At any time t, data instance except for discarded from previous $(i-1)$-th update, are waiting in the queue for i-th update. In case, the length of the queue at time t is given by:

$$Q(t) = \{d_k | t_{i-1} \leq a_k \leq t\} \tag{4}$$

Expectation of $|Q(t_i)|$ is $\tau_{i-1}\lambda$, where τ_{i-1} denotes time interval between consecutive update $(t_i - t_{i-1})$ and data arrives as Poisson or uniform distribution with arrival rate λ.

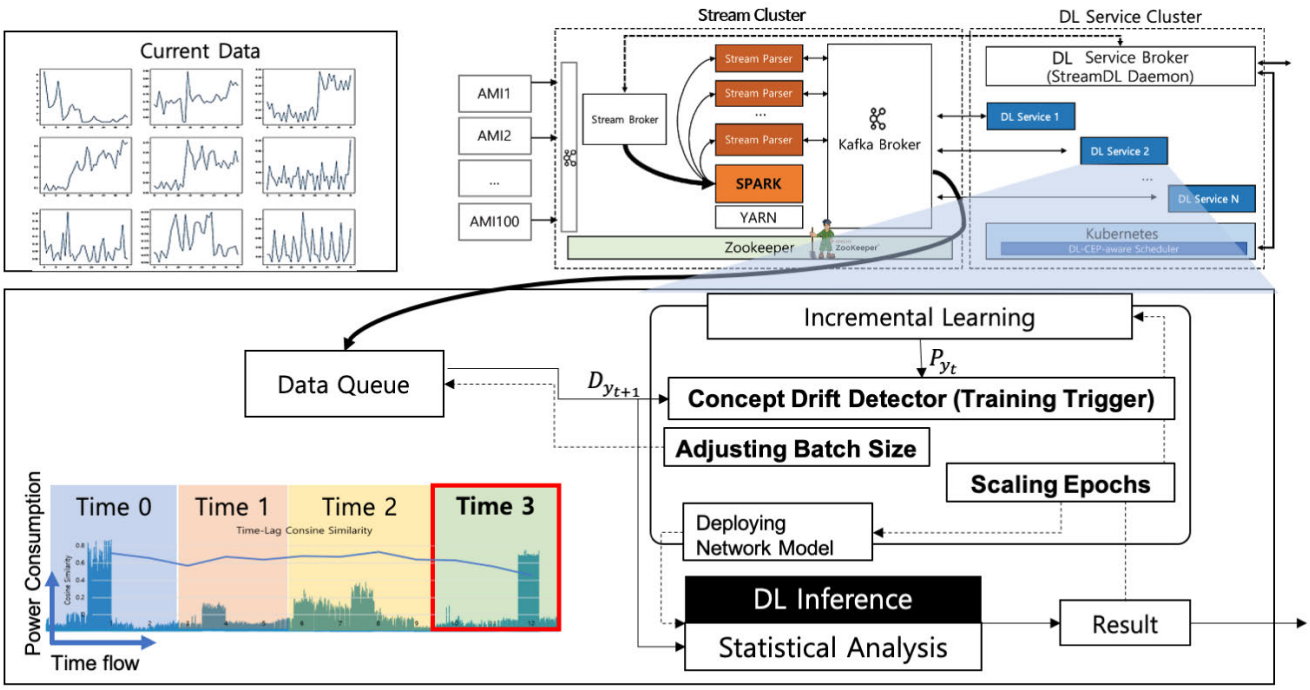

FIGURE 2. An illustration of the system model for AMI data stream processing with adaptive incremental learning.

3) ADAPTIVE INCREMENTAL DEEP LEARNING SOLVER

Incremental deep learning uses an online optimizer that trains a model through temporary mini-batch training. Therefore, it is not possible to train with several epochs over the entire dataset as usual. To prevent the possibility of falling into local minima, a robust learning method for error and noise is required. A compromise between batch learning and incremental learning is incremental learning in a batch-based manner, which can be applied to a data stream analysis system. For a given temporary mini-batch, we can determine the hyperparameter for training, while expecting high accuracy and training time performance [3]. In particular, the adaptive incremental deep-learning solver dynamically schedules the batch size and epoch for the given temporary mini-batch based on the degree of concept drift and the training curve of the regression model.

Burst input data stream rates can lead to situations in which the queue is full, and the system only processes data streams that have previously been entered for training. Therefore, the scheme should operate under queue stability conditions that do not cause queue overflows, expecting high levels of training convergence. Profiled results are a characteristic of a system that can be profiled or known before training, such as the computing power of the hardware.

In section IV, we present the hyperparameter assignment criteria, which are used whenever the system repeats model training with temporary mini-batches.

B. DEEP-LEARNING TRAINING MODEL WITH ADAPTIVE INCREMENTAL LEARNING

In this subsection, the deep-learning training model with the incremental learning scheme is presented. An overview of the scheme is shown in Fig. 3. Let a deep-learning model (in this article, simply call it model) be defined as set of parameters (referred to as weight and bias in a neural network) exquisitely adjusted to represent the relationship between the input and output data domains. As a data stream arrives, a series of model retraining begins with incremental learning to adjust to new features in the data. In addition, inference is performed on the arrived stream using the pre-trained model in parallel.

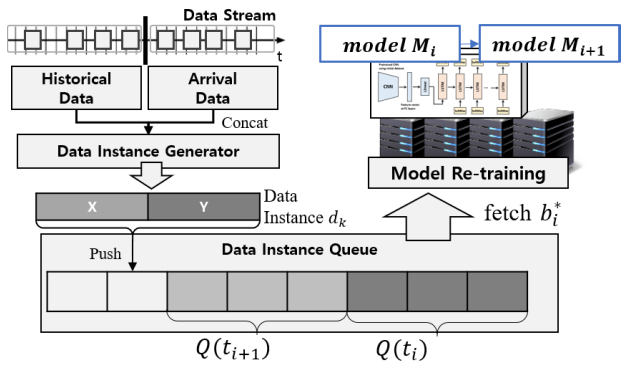

FIGURE 3. Description of data instance and instance queue used for Eq. (5).

1) ITERATION TIME MODEL OF MINI-BATCH SGD BASED DEEP LEARNING

In this subsection, the cost model of processing time for updating deep learning model in time t_i is considered; it basically follows the model in [3], [7], [26].

Re-training is proceeded intermittently when enough data is collected in queue. As a model update, a series of data integration, retraining results are conducted.

The time of i-th update is defined as t_i. A set of instances D_i arrived between t_{i-1} and t_i is to be used to update a model M_i to M_{i+1} which model parameter are adjusted with the set of instances. Note that the data is not discarded after it is included; values are overlapped over stream sets.

$$D_i = \{d_k | t_{i-1} \leq a_k \leq t_i\} \tag{5}$$

Let $|D_i|$ as number of instance arrivals within i-th model update, which is batch size simply denoted as b_i.

Updating model over new data incurs non-trivial training cost, which is directly measured by the processing time. Within a machine, iteration time of epoch in DL task is represented as sum of time measured in mini-batch upload, feed-forward, back-propagation and gradient transfer tasks. The feed-forward and back-propagation tasks are processor related and mini-batch upload and gradient transfer tasks are memory related.

As there are model related and computing related parameters existed, we can estimate performance model coefficients by using nonlinear multi-variable regression model. In paper [3], [7], they empirically shown strong linearity of relationship between processing time and data size $d \in \mathbb{N}$, which is in set of natural number. The approximated processing time to compute i-th update with size of input b_i is given by,

$$\begin{aligned} \tau^{iter}(D_i) &= (FW_c(D_i) + BP_c(D_i)) \\ &\quad + (UP_m(D_i) + TR_m(D_i)) + \theta_{ind} \\ &\approx \theta_c\,|D_i| + \theta_m\,|D_i| + \theta_{const} \\ &= (\theta_c + \theta_m)\,|D_i| + \theta_{const} \\ &= \theta_{proc}\,b_i + \theta_{const} \end{aligned} \tag{6}$$

where, FW_c, BP_c, UP_m, TR_m represent the completion time estimation function with respect to given mini-batch D_i for feed-forwarding, back-propagation, mini-batch upload, and gradient transfer with communication, respectively. And θ_c, θ_m refers to the performance estimation parameter for computation and memory access operation. Also, feed-forwarding and back-propagation are computation-oriented task, and gradient transfer and communication are memory-oriented task, which is illustrated in Fig. 4. Then, the elapsed time of i-th update $e \in \mathbb{N}$ epoch can be presented as proportional to the number of epoch. As a result, we can derive the elapsed time of model update as follows:

$$\tau^{iter}(b_i, e_i) = (\theta_{proc}\,b_i) \times\ e_i + \theta_{const} \tag{7}$$

where, θ_{const} is the overhead of model training and initialization procedure in devices. Models trained at each stage are deployed to service applications, responding to user requests using the latest deep learning models. Deployment of the updated model involves loading memory, disks, and networks to load model weight parameters, depending on the system environment. In addition, for the next deployment,

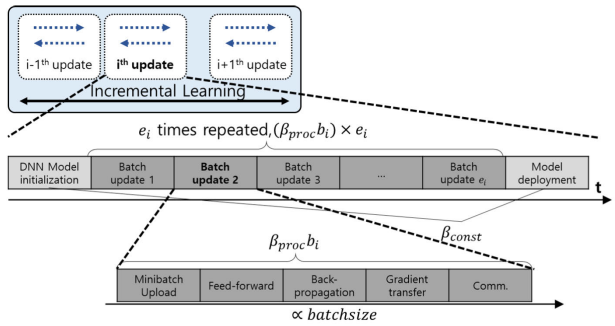

FIGURE 4. An example of iteration time in mini-batch SGD based Deep Learning used for Eq. (7) [7], [26].

there is initialization overhead for training new instances in data stream.

For the same amount of data instance, the number of update to training the data instance can affect on training time. As the number of update larger, model deployment and initialization occur more frequently, result in need more cost for training.

From the training time model in Eq. (7), we define cost reduction function CR derived with subtraction of cost through only once/ most frequent update policy. For the same amount of temporary training data in queue, the most frequent update requires $(\theta_{proc} e_i + \theta_{const}) b_i$ and only once update requires $\theta_{proc} b_i e_i + \theta_{const}$. The degree of cost savings through fewer model updates as follows:

$$CR_i(b_i) = -\theta_{const}(b_i - 1) \approx -\theta_{const} b_i \tag{8}$$

2) MODEL RECENCY

Model Recency, R, is defined as a ability of how quickly the deep learning model can reflect a recent trend of input data [7]. R_i is time for each data in streaming data set If the time difference between t_i and t_{i-1} is small enough, it will have the best recency performance. In extreme cases, updates can be performed every time an instance arrives. If there is not enough data to train the model, it should be able to use the data quickly for model updates to provide better quality services.

When queue is in stable condition, model recency R_i is waiting time of a instance for further update is only time interval between arrival time of instance and next update time as follows:

$$R_i = \sum_{k \in Q(t_i)} t_i - a_k^{Q(t_i)} \tag{9}$$

Assume that data generated in data source is periodic and arrival comes as uniform or Poisson distribution with average interarrival time $1/\lambda$. When there are b_i number of instance arrived, model recency (Eq. (9)) is derived as function of b_i as follows:

$$R_i(b_i; \lambda) = \sum_{i=0}^{b_i - 1} i/\lambda \approx b_i^2/\lambda \tag{10}$$

3) MODEL CONVERGENCE RATE

The impact of training set batch size on accuracy performance is controversial. Usually, the models trained with the smaller batch size are likely to fail to generalization, showing the more unstable fluctuation [27], [28]. Research experimentally shows the testing accuracy of the trained CNN on the MNIST data set and CIFAR-10 according to the batch size [29]. The result with small batch size shows the lowest test accuracy, while it shows highest test accuracy when the large batch size. The larger number of epochs, the number of data to pass the model, to adjust the parameters in the model usually improve prediction accuracy. Therefore, the number of epochs for training helps to a find good predictor in deep learning before the number of epochs where model becomes overfitting. Optimus [26] model shows the convergence curve and the equation leverages the training loss converges at a rate of O(1/E) regarding the number of epochs, E. The unstableness metric is proposed to implement robust DL framework on concept drift in [3], but shows non-convergence of model training according to high fluctuation in performance.

Therefore, we define the degree of the trained model quality converged into saturation as a model convergence rate function. In incremental learning, as concept drift occurs, there may be high training loss resulting in low model recency from the viewpoint of model convergence. Lower convergence for the new arriving stream degrades the accuracy. To achieve fast convergence for a given stream with new distribution, the batch size and epoch size are increased. With a large batch size, robustness to noise is guaranteed, while a large epoch provides high convergence for a given mini-batch. To detect concept drift, we utilize the training loss at each update. However, swift loss fluctuation is not conducive to model stability [27], [28]. In the paper by Ross *et al.* [5], they proposed an exponentially weighted moving average to detect concept drift. We use the average loss value to determine instability through the k time window method. As training loss of the update instance in $i-1$ is given as tl_{i-1}, the average training loss is derived as follows:

$$atl_i = 1/k \sum_j tl_{i-j} \quad \forall j = 1, \ldots k \tag{11}$$

We define model convergence rate as $\frac{atl_i}{b_i e_i}$, where b_i is the number of instances in a stream data set and e_i is the number of passes of given stream data.

$$MCR_i(b_i, e_i) = \frac{atl_i}{b_i e_i} \tag{12}$$

4) QUEUE STABILITY CONDITION

In the queuing theory, the system of queueing networks is considered as stable if its long run averages exist and are less than infinite. Especially, a single queue system is stable if arrival rate is less than system processing capacity.

The queue length after i-th update is denoted as summation of previous queue length and gap of incoming and processed stream data in queue.

$$|Q(t_i)| \rightarrow |Q(t_{i-1})| + \tau_{i-1}\lambda - b_i \tag{13}$$

When we assume that initial state of queue $Q(t_0)$ is empty, a condition for queue stability can be simply defined. From the definition of stability and Eq. (7), a stability condition of queue is derived as follows:

$$\tau_{i-1}\lambda \leq b_i = \frac{\tau_{i-1} - \theta_{const}}{\theta_{proc} e_i} \tag{14}$$

From the definition of τ_i, it should be greater than 0. Thereafter, by using Eq. (14), we can get the constraint for the number of epoch in update i as follows:

Constraint for queue stability:

$$e_i \leq \frac{\tau_{i-1} - \theta_{const}}{\tau_{i-1}\theta_{proc}\lambda} \tag{15}$$

In addition, due to $e_i \in \mathbb{N}$, the right term of Eq. (15) should be greater than or equal to 1. As a result, the minimum time interval between succesive updates is derived as:

$$\tau_{i-1} \geq \frac{\theta_{const}}{\theta_{proc}\lambda - 1} \tag{16}$$

where, $\frac{\theta_{const}}{\theta_{proc}\lambda - 1}$ is lower bound of time interval between subsequent updates. If $\theta_{proc}\lambda < 1$, then stable processing is not possible and data drop policy might be useful, however it is not considered in this article.

IV. AN ADAPTIVE INCREMENTAL DEEP LEARNING SCHEME

A. COST FUNCTION AND MULTI-CRITERIA COST OPTIMIZATION

This subsection presents the cost function of the incremental learning solver, which implies the multi-objective equation of model recency, training cost, and model accuracy to minimizing inference service loss.

Over time, the characteristics of data change, and retraining for a data stream must be handled in real time. Therefore, we consider the degradation of computing performance due to scheduling overhead. To address this problem, we propose short-term, stateless scheduling of model retraining. The decision vector contains the size of the input and the number of epochs for the i-th update. These variables indirectly contribute to the recency and short-term instability of the model, which determines the performance of the trained model.

At the same time, queue stability should be guaranteed to avoid system failure caused by buffer overflow. As stream data continuously arrives in the system, the decision variable t_i, which is the time to update is simply equivalent to the input size (or batch size) b_i.

Therefore, based on Eqs. (9), (8) and (12), we design a cost function for the quality of model retraining that indirectly affects inference loss and training cost as follows:

$$f(b_i, e_i) = R_i(b_i) + \gamma_1 CR_i(b_i) + \gamma_2 MCR(b_i, e_i) \tag{17}$$

where γ_1 and γ_2 are the weight factor of cost reduction and instability, respectively.

Thus, from Eqs. (15) and (17), the optimization problem is derived as follows:

$$\min_{(b_i,e_i)} f(b_i, e_i) := \frac{b_i^2}{\lambda} - \gamma_1\theta_{const} b_i + \gamma_2 \frac{atl_i}{b_i e_i}$$
$$\text{subject to } 0 < e_i \leq \frac{\tau_{i-1} - \theta_{const}}{\tau_{i-1}\theta_{proc}\lambda} 0 < b_i \tag{18}$$

In addition, to find the optimal solution of Eq. (18) and to train the deep-learning model with (b_i^*, e_i^*), in a continuous manner are proceeded on incremental learning system in advanced.

B. OPTIMAL TRAINING SCHEME

In this subsection, we introduce our proposed incremental deep-learning-based optimal model retraining scheme. Fig. 5 shows the structure of the proposed incremental learning solver with system variables.

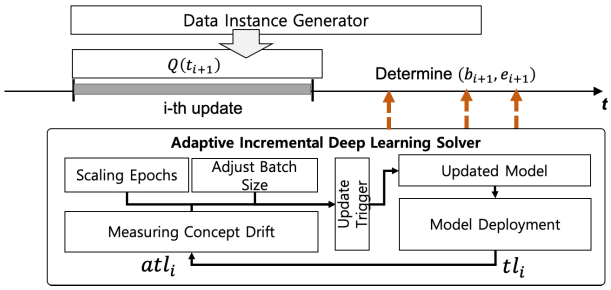

FIGURE 5. Structure of incremental learning solver used for Eq. (18).

Assume that model training is proceeded until $(i-1)$-th update. After the previous update, the solver obtained information of the average training loss atl_i with time-window averaging. With the conditions of recency, training cost, instability, and instance queue, the solver makes a decision for the next update time t_i, equivalently b_i, and epoch e_i.

Theorem 1: For the condition of $e_i \in \mathbb{N}$, there are only one b_i^* exists in $\mathbb{R}^+$ that minimize the cost function of Eq. (17) within continuous range that $b_i > 0$

Proof: To prove conditions, the first and the second-order partial derivative of f for b_i are given as follows:

$$\frac{\partial f(b_i, e_i)}{\partial b_i} = \frac{2b_i}{\lambda} - \gamma_1\theta_{const} - \gamma_2 \frac{atl_i}{b_i^2 e_i} \tag{19}$$

$$\frac{\partial^2 f(b_i, e_i)}{\partial b_i^2} = \frac{2}{\lambda} + \gamma_2 \frac{2\, atl_i}{b_i^3 e_i} \tag{20}$$

From the corollary of convexity for second-order conditions [30], function f is strictly convex if $\nabla^2 f(x) \succ 0$ for $x \in \operatorname{dom} f$. The cost function f is strictly convex where $b_i \in \mathbb{R}^+$, because f if continuous function for $e_i \in \mathbb{N}$ and second-order partial derivative is at least $\frac{2}{\lambda}$, even second term is also larger than zero. As Eq. (18) is strictly convex in case of $b_i > 0$, the minimum value of multi-criteria cost function can be derived as solution of $\frac{\partial f(b_i,e_i)}{\partial b_i} = 0$.

Algorithm 1 Adaptive Incremental Learning Algorithm for Stream Data Processing

Input: Update Step i, Current Model M_i, Training time regression param $\theta_{proc}, \theta_{const}$, Instance Queue $Q(t)$, Prev update timestamp t_{i-1}, Optimization Parameters (ϵ, η, γ)
Output: Retrained Model M_{i+1}
01: **Wait until** $(t - t_{i-1}) > \frac{\theta_{const}}{\theta_{proc}\lambda - 1}$
02: **Let** $e_i = e = \frac{(t-t_{i-1})-\theta_{const}}{(t-t_{i-1})\theta_{proc}\lambda}$
03: **while do**:
04: $grad = \frac{f(e_i)-f(e_i-\epsilon)}{\epsilon}$ // gradient
05: **If** $grad \geq 0$ **or** $\eta \cdot grad < \epsilon$:
05: break //stop condition
06: $e_i = e_i - \eta \cdot grad$
07: $\eta = \gamma \cdot \eta$ //decay step size
08: $b_i = d(e_i)$ //refer Eq. (21)
09: **Wait until** $Q(t) \geq b_i$
10: $M_{i+1} \leftarrow train(M_i; b_i, e_i)$ //mini-batch training. Eq. (3)
11: **return** M_{i+1}

In case of $b_i \neq 0$, the first-order partial derivative of f is equivalent to

$$\Rightarrow \frac{2}{\lambda} b_i^3 - \gamma_1\theta_{const} b_i^2 - \gamma_2 \frac{atl_i}{e_i}, \tag{21}$$

which is form of cubic equation. For the general cubic equation $ax^3 + bx^2 + cx + d$, critical point is given to,

$$x_{critical} = \frac{-b \pm \sqrt{b^2 - 3ac}}{3a} \tag{22}$$

When c $= 0$, there are two critical points on $x_{critical} = (0, \frac{-2b}{3a})$. For the original problem, $b_{i,critical} = (0, 2\gamma_1\theta_{const})$. By using geometric analysis, the equivalent function of $b_i > 0$ is $\frac{\lambda\gamma_1\theta_{const}}{3}$. Because the value of the equivalent function on $b_i \to 0^+$ is $-\frac{\gamma_2\, atl_i}{e_i}$, which is a negative value, there are always two imaginary solutions and one positive solution at $\frac{\partial f(b_i,e_i)}{\partial b_i}$ is zero. Additionally, there are already solutions of cubic equations in general. □

We define function $d(e_i)$ as natural number solution of cubic equation in Eq. (21) with respect to parameter e_i.

For $e_i \in \mathbb{N}$, a cost function is derived from Eq. (18) with $d(e_i)$ as follows:

$$\min_{e_i} f(e_i) := \frac{d(e_i)^2}{\lambda} - \gamma_1\theta_{const} d(e_i) + \gamma_2 \frac{atl_i}{d(e_i)e_i}$$
$$\text{subject to } 0 < e_i \leq \frac{\tau_{i-1} - \theta_{const}}{\tau_{i-1}\theta_{proc}\lambda} \tag{23}$$

According to Eq. (23), it is not easy to find an optimal solution. Instead, to reduce the computational complexity of optimization, we propose heuristics to find a sub-optimal solution of the objective cost function. For any non-convex function, we can find the local minimum through the gradient descent method [31]. Gradient descent is a first-order iterative optimization scheme for finding the minimum of a function.

In the heuristic approach, starting from $e = \frac{\tau_{i-1}-\theta_{const}}{\tau_{i-1}\theta_{proc}\lambda}$, while gradually reducing the step size, the function space is searched until the moving step becomes smaller than ϵ, to find the local optimum.

$$e_i^{j+1} = e_i^j - \eta \nabla f(e_i^j) \tag{24}$$

where η is the rate of the step size.

We can apply early-stop to the optimization procedure when $\nabla f(e_i^0)$ is positive, based on the upper bound condition. Indeed, through the an empirical test, we found that the number of searches for convergence is approximately 5 because the frequent occurrence of early-stop case even for the worst condition. Finally, we can always obtain the solution with the heuristic.

As a result, the numbers of instances and repetitions for each training iteration depend on the following contributing factors: the detected number of errors in concept drift, the computational overhead of the training model, and the arrival rate of the input data stream. High input data stream rates can lead to situations in which the queue is full and the system can only afford to process data streams that previously entered for training. Therefore, the scheme works under queue stability conditions in which queue overflow will not occur, expecting high learning convergence.

V. ACCELERATED DEEP LEARNING PROCESSING IN EDGE CLOUD SYSTEM

A. DEEP LEARNING TASK SCHEDULING SCHEME IN ACCELERATED EDGE CLOUD

The processing time requirement for the n-th task among N tasks is defined as $T_{req}(n)$. When the task comes in, T_{req} is user requirement on processing latency of given request, but it is reduced over time. T_{req} is defined as a value obtained by subtracting the difference between the time at which the task is entered and the current time at which the scheduling is performed from the initial processing time requirement. We also assume that FPGA and GPU are scheduled with non-preemptive method. After one task has been processed, the next task is getting processed similarly.

1) RESOURCE SET

When the maximum processing performance of each FPGA and GPU is defined as P_{max}^{FPGA}, P_{max}^{GPU} request/sec, the overall processing performance of the system consisting of N FPGAs and M GPUs is generally $(N \times P_{max}^{FPGA} + M \times P_{max}^{GPU})$ request/sec. If more tasks are entered than given throughput per 1 second, it can not be processed through the current system. More accelerator resources are required through resource provisioning, such as VM scaling or Cloud offloading. This situation is not considered, and it is assumed that a smaller number of packets are always received than the maximum processing performance. A purpose of scheduling is to minimize total energy consumption for processing N tasks while satisfying all processing time requirements of N tasks.

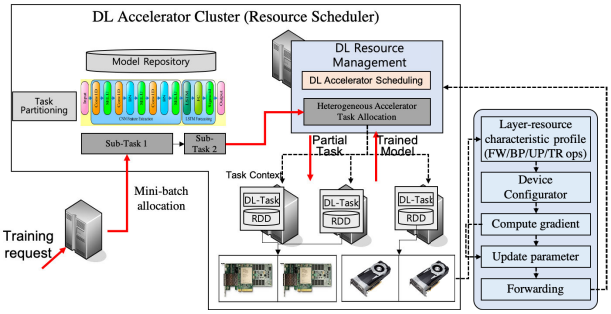

FIGURE 6. Illustration of resource scheduling scheme in accelerated edge cloud that we proposed.

When we define the resource set of entire computing respectively,

$$R = \{R_i \text{ where } 0 < i \leq N \text{ for FPGA}, \\ R_j \text{ where } N < j < N + M + 1 \text{ for GPU}\} \tag{25}$$

Since the intermediate data size and the preprocessing overhead of the deep learning model are different, the limited network bandwidth and service functions of the edge node will solve the scheduling problem for maximizing the number of deep learning tasks. We also want to ensure the quality of service (QoS) of each deep learning service in scheduling.

Deep learning network is designed in a multi-layered structure. Each layer creates the next feature through the intermediate feature data created in the previous layer, and classifies or recognizes the output through the feature output from the final layer.

In deep learning networks, layers adjacent to input data are considered as lower layers, and intermediate feature data generated in each layer is generally smaller than the size of input data.

Deep learning network partitioning is performed through layer decomposition and profiling to solve the communication performance problem between GPU and FPGA and the communication bottleneck problem in multi-level edge structure.

2) DEEP LEARNING EXECUTION TIME AND COST ESTIMATION

Let us define a typical Deep Learning Task as a set of sequentially linked tasks represented as pipeline as follows:

$$T = \{T_1, T_2, \ldots, T_K\} \tag{26}$$

Among them, the function $x_i^k(t)$ expressing the occupation due to the allocation of the task k at a specific time t to define available computing resources is as follows.

$$x_i^k(t) = \begin{cases} 1 & \text{if task } T_k \text{ is allocated to } R_i \text{ at time } t \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

Resource profiling is the method to figure out expected execution time for a task T_k when it is processed on resource type in FPGA/GPU and manages the execution time data in the form of table. The row of table represents different task

type of each layers in neural network $k$and the column represents accelerator (resource) type i. Execution time data τ_i^k is the average value acquired from enough times of repeated execution. Especially for the FPGA, additional execution time cost is required for model as reconfiguration overhead rc_k^i while logic changes of FPGA as $rc_k^i(t) = |x_i^k(t-1) - x_i^k(t)| * r_i^k$, where r_i^k is reconfiguration time for task k in FPGA i.

Based on this, the execution time for sub-task should be modeled in Eq. (7) with regression parameters $\theta_{proc}, \theta_{const}$ for each FPGA and GPU, respectively. When adaptive incremental deep learning solver make decision, they request the resource allocation and also acquire performance profiling represented as $\theta_{proc}, \theta_{const}$, respectively. After then, with the adjusted batch size and epoch, the solver send DL pipeline request to task scheduler.

3) TASK SCHEDULING

At this time, the execution time as a service response time for one application service is expressed as an input of the next allocation as a result of the previous time allocation. Since the mathematical model of execution time is expressed in the form of coupled input and output, this problem is difficult to solve with general linear programming and is an np-complete problem.

To solve cost minimization along with the issue of guaranteed resource scheduling, you need to allocate the appropriate resources to pay the minimum cost for the processing of individual tasks. In addition, the entire DL schedule created as an individual task schedule must meet the custom deadline. Kim [32] proposed a step-by-step scheduling scheme. With the colored Petri net model, colored tokens are defined in the pipeline that perform different behavior depending on the color, which controls the scheduling of the specified structure. This plan consists of two phases. The first step is the scheduling step, which determines the hierarchical ratio and assigns a child deadline to each job through reverse token delivery of scheduling tokens through the Petri net. The second is the execution phase, which passes execution tokens to allocate the appropriate resources according to the load percentage of each task, so as not to violate the deadline. As a result, the path responsible for most of the load in the pipeline, that is, the critical path, is discovered and passed by the scheduling token. The scheme uses quiet, simple and intuitive heuristics. This method uses static estimated job processing time, but is classified as dynamic scheduling as job scheduling is performed according to the remaining service level indicator.

Defining relative load as the average execution time over FPGA/GPU resources as relative load rate for all sub-tasks in the pipeline,

$$\text{relative load } rl^k(t) = \frac{1}{(M+N+1)} \sum_{i=1}^{N+M+1} \tau_i^k(t) + \sum_{i=1}^{N} rc_i^k(t) \tag{28}$$

Algorithm 2 Adaptive Incremental Deep Learning Task Scheduling Scheme in Accelerated Edge Cloud

Step 1. Heterogeneous Accelerator Profiling (task scheduler)

Evaluate execution time of DL task for FPGA/GPU with respect to the size of task workload (linear regression on batch size/epoch);

Repeat until every entitled DL task in model $M = \{m_1, m_2, \ldots, m_P\}$ repository evaluated;

Create a resource profiling table;

Step 2. Planning of DL Training (adaptive incremental deep learning solver)

Wait for enough stream data to queue up;

Adaptive incremental learning solver send the information of DL model, expected size of task workload;

Get resource profiling in terms of $(\theta_{proc}^{ip}, \theta_{const}^{ip})$ from resource scheduler, $\tau^{iter}(b_i, e_i) = (\theta_{proc}\ b_i) \times\ e_i + \theta_{const}$;

Adjust the batch size and epoch using Eq. (18);

Send DL pipeline $T = \{T_1, \ldots, T_k\}$ into resource scheduler, Eq. (26);

Step 3. Execution of DL Pipeline (task scheduler)

Assign the best accelerator $R^k(t) = \{R_i | \tau_i^k < sd^k(t), min(C_i \times \tau_i^k), x_i^k(t) = 1\}$;

Execute tasks based on mini-batch SGD, Eq. (3;

Send training error to the solver;

Defining load rate as relative load of task k compared to the relative load of its following tasks.

$$\text{load rate } lr^k(t) = \frac{rl^k(t)}{\sum_{j=1}^{K} rl^j(t)} \tag{29}$$

Defining a sub-deadline to identify a problem as a combination of sub problems to reduce the complexity of problem solving

$$sd^k(t) = lr^k(t) \times \left(T_{req}(n) - \sum_{k'=1}^{k} R^k(t) \right) \tag{30}$$

When the leasing cost per unit time for arbitrary resource type i is denoted as C_I and there are known estimated times for task execution times on each task, we can allocate the most efficient resource guaranteeing the sub-deadline for the j-th task from resource type as follows:

$$R^k(t) = \{R_i | \tau_i^k < sd^k(t), min(C_i \times \tau_i^k), x_i^k(t) = 1\} \tag{31}$$

If there is no available resource type to guarantee the sub-deadline, this may cause deadline violation for the entire pipeline. Then there might be additional policies for the system operation might be required. Because resource set R is defined for both FPGA and GPU in Eq. (25), specified of resource type is not preferred and only the resource that minimize processing time evaluated in Eq. (31) is allocated for the task.

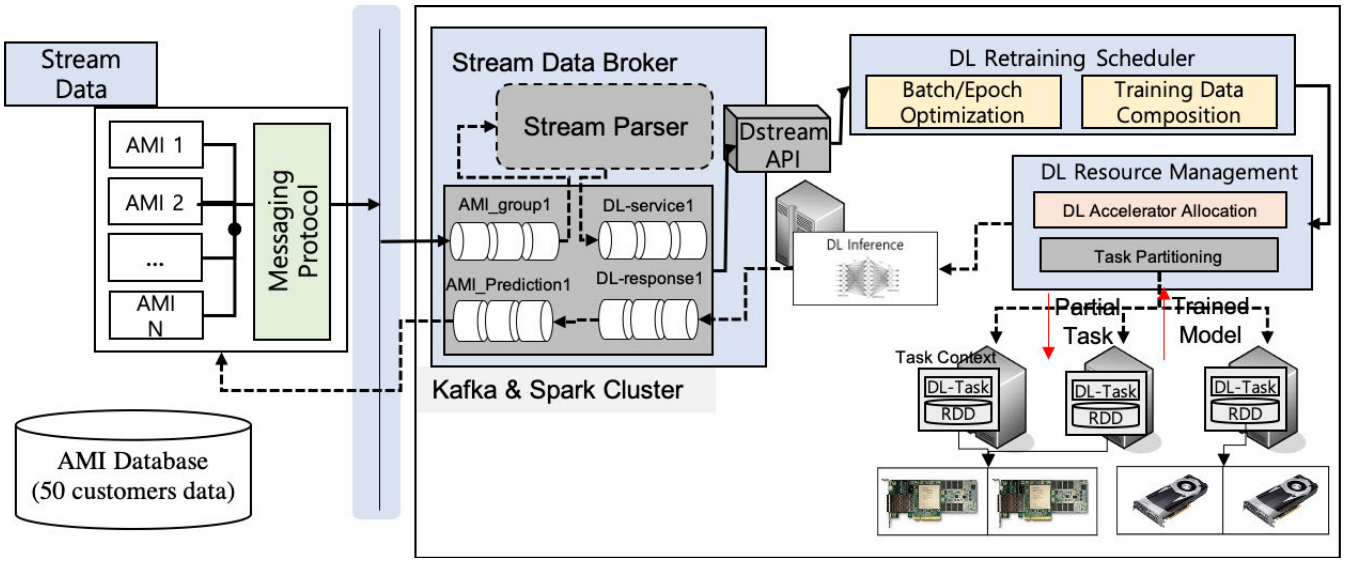

FIGURE 7. A stream-based incremental deep learning system with accelerated computing platform for performance evaluation.

In algorithm 2, there is procedure of incremental deep learning acceleration through task scheduling on heterogeneous accelerator. As adaptive incremental learning solver send the information of DL model, expected size of task workload, the profiled parameter $(\theta_{proc}, \theta_{const})$ is returned. After allocating amount of training task with the parameter, configured request of DL task is submitted to resource scheduler. A resource scheduling scheme manages various accelerator resources to accelerated deep learning processing while minimizing the computational cost. When adaptive incremental deep learning send DL task into resource scheduler, it is able to accelerate DL task processing using heterogeneous accelerators.

VI. PERFORMANCE EVALUATION

In this Section, to evaluate the performance of the proposed adaptive incremental deep learning scheme and resource scheduling scheme, which provides efficient operation of stream data training in the energy data processing system, we implemented the stream-based incremental deep learning system with accelerated computing platforms, as shown in Fig. 7. We conducted experiments on the real benchmark dataset for the prediction of future demand and generation. The cluster setup for training and testing, datasets, specification of a used deep learning model are given in Section 6.1, the detailed performance metrics for evaluation is provided in Section 6.2, and the results with the comprehensive comparisons are reported in Section 6.3.

A. EXPERIMENTAL ENVIRONMENT

In the experiment system, the smart metering device generates energy stream data. The data is converted to a standard format and sent to the data processing cluster through a distributed queuing system. The data processing cluster connects to the stream, acquires the data in small batches, preprocesses them, and queues them for deep learning training. By obtaining learning parameters of the next step using the adaptive incremental learning solver, cluster utilize heterogeneous acceleration resources to training the demand prediction model.

1) CLUSTER SETTING IN EXPERIMENT

Firstly, We built the two clusters with the edge cloud platform using the heterogeneous type of computing nodes, respectively. Due to the heterogeneity in the multiple cluster composition, various resource profiling was conducted on each cluster. The specification of computing nodes in the heterogeneous cluster of each having GPUs, FPGAs and CPUs was shown in Table 3. The cluster was consisted with two types of GPU computing nodes, namely, one with GTX 1080 and one with NVIDIA RTX 2060 super. The edge server is constructed with Arria 10 GX FPGA from Altera platform. The details of the specification were described in Table 3. All computing nodes in the cluster was installed with Ubuntu 16.04, and all experiment was conducted with deep learning platforms: version 1.12.0 of TensorFlow and 2.2.4 version of Keras. Intel® FPGA SDK for OpenCL software technology is one of High Level Synthesis (HLS) development environment that enables software developers to accelerate applications by targeting heterogeneous platforms with Intel CPUs and FPGAs. This environment combines abstracts FPGA details while delivering optimized results. We used TensorFlow to implement CNN-LSTM model on GPU, and OpenCL with C++ to implement on FPGA. Also, for the stream data handling, we deployed a Kafka message broker and spark distributed data processing framework as a front-end system. The input data streams in the AMI database were sequentially sent into the system in a uniform time interval

TABLE 3. Detailed specification of each computing node in the cluster used for experiment.

	Cluster1	Cluster2
CPU	I7-6700 @3.40GHz	Intel Xeon E5-2650 v2 @ 2.60GHz
GPU	GTX 1080	RTX 2070 super
NVIDIA Cores	2560	780
GPU Memory	8GB GDDR5	8GB GDDR6
Memory Bandwidth	320.3 GB/sec	448.0GB/sec
FLOPS	8.873T	9.062T
FPGA	Arria10 GX dev	Arria10 GX dev

TABLE 4. Benchmark dataset used for evaluation.

	Prediction on Demand Side [13], [33]
Type	House energy consumption
Data Interval	Measured every 15 minute in 2017
Missing Data	True
Explanation	Energy demand data from 51 house which is randomly selected

using Kafka messaging API. After pre-processing the data stream, it is used as inference and training for incremental deep learning in spark workers.

2) AMI DATASETS WITH ENERGY DATA ANALYSIS SCENARIO

In this article, we use AMI data for an experimental dataset. We obtained the energy demand data collected by the Korea Electric Power Corporation (KEPCO) measured in 2017 in Jeon-nam, Korea, provided for research purposes only [14]. It is household power consumption data collected every 15 minutes from January to December. The AMI dataset consists of datasets of more than 50 energy subscribers created every 15 minutes with 24/7. Each customer sends more than 30,000 data to the energy analytics platform on the edge cloud system. The system monitors the sensor data stream for predicting future power demand.

However, missing data was observed in AMI data due to incomplete infrastructures such as harsh working conditions, imperfect communication signals, and device failure, which causes prediction errors in the energy planning model [34]. Various algorithms have been proposed to solve the missing data problem using interpolation [35], principal component analysis (PCA) [36] and k-NN [37] methods. In addtion, deep neural network based imputation schemes have been proposed recently, named Generative Adversarial Imputation Network (GAIN) [38]. In this article, we use quadratic interpolation model [35] on AMI dataset for missing data imputation.

3) CNN-LSTM MODEL USED FOR EXPERIMENTS

In this experiment, we used the Convolutional Neural Network Long Short-term Memory (CNN-LSTM) model for the energy demand prediction scenario. The CNN-LSTM model was initially proposed as a Long-term Recurrent Convolutional Network (LRCN) to process visual recognition for

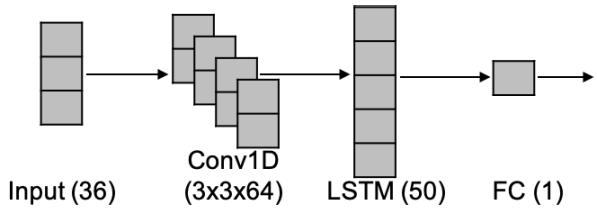

FIGURE 8. Structure of CNN-LSTM Model used in experiments.

images and videos [39]. The proposed LRCN was configured with the sequential combination of CNN and LSTM model. The CNN offered superior performance in structural or spatial feature extraction and robust to noise. On the other hand, the LSTM can model complex temporal dynamics with variable-length inputs. Indeed, the LRCN model showed a deep understanding of spatial and temporal information embedded in the visual dataset and was evaluated to have an excellent performance on visual recognition applications.

Furthermore, in paper [40], they proposed a particulate matter forecasting model in the smart city based on CNN-LSTM. For time-series sequence data as well as visual data, CNN-LSTM extracted structural features from the arrangement of lookback data and showed high prediction performance.

In paper [4], they proposed a CNN-LSTM model for robust and efficient forecasting of energy consumption. Similarly, the CNN-LSTM model showed excellent performance on power consumption prediction than other machine learning methods with low fluctuation. For the experimental dataset, we configured the model parameters of the input dimension of the Conv1D layer and the output dimension of the LSTM layer. We configured the input dimension as 36 and the output dimension as 1. A stride of the pooling layer can affect the size of the model, and stride in the used model is 2. The number of parameters in the model used for evaluation was 24,267 parameters, and all of these parameters are trainable. The number of parameters affected model storage operation called deployment overhead because the more significant amount of parameters, requires more memory and longer time for memory read and write.

To implement AMI demand prediction module on FPGA, we first implemented each layer (convolution layer, fully connected layer) to kernels with the baseline model of CNN implementation in FPGA [41]. Through the High Level Synthesis (HLS) tool provided by Intel Altera SDK for OpenCL, we compiled the raw kernel(.cl) to the synthesized bitstreams (.aocx) with the compiling tool. We implemented the host side code in C++ to load the binary program of the compiled kernels. Logic utilization on Combinational ALUTs, Memory ALUTs, Logic registers or Dedicated logic registers was 65,026/427,200 (15%) and Memory bits is 1,722,512/55,562,240 (31%). It's power consumption was measured as 2W. The LSTM layer is implemented on both CPU and GPU for the compatibility. We measured two kinds of execution times, the layer execution time and the kernel

execution time. The layer execution time is measured from the host code, and kernel execution time is measured by detecting the OpenCL kernel's start and end event. And for the GPU, we measured execution time for Conv, LSTM and FC layers. To implement CNN-LSTM on GPU, we implemented each layer using Tensorflow. The execution time is measured using the profiler provided by Tensorflow.

B. PERFORMANCE METRICS FOR EVALUATION

The main goal of the energy service system is to use streaming data to update learning models to more accurately predict future energy demand and supply for providing a stable energy supply and save energy costs. As described in section 4.1.2, the household energy consumption data in type of AMI is used as a experimental data set. Performance is evaluated in terms of the accuracy of the trained model and the training time for the newly arrived data. In detail, the accuracy is measured by the Root Mean Squared Error (RMSE) at a given time, and also the standard deviation of the RMSE observed so far is measured to see whether the trained model provides a stable prediction without the risk of changes in incoming data distribution. The higher the RMSE on unseen data, the more accurately the model used for serving predicts. On the other hand, training time is evaluated to measure the execution time of the proposed system. Shorter training time save computation cost on edge cloud, means the system spends less time and capacity allows to reduce computing resources.

1) PREDICTION ERROR (RMSE)

RMSE is a validation metric which indicates the the difference between an estimated value or a value predicted by a model and observed in a real environment. For the i-th trained model M_i, the prediction function $F_{M_i}(X)$ predict Y on i-th set of stream data instances d_i. Then, the $RMSE_i$ is denoted to Eq. (32).

$$\sqrt{\frac{\sum_{X,Y\in d_i} Y - F_{M_i}(X)}{|d_i|}} \tag{32}$$

In addition, to investigate the degree of prediction accuracy over the entire dataset, we define the average RMSE as the mean value of RMSE for the variable i. Here we evaluate average RMSE in the training loss and model score. Training loss is referred as loss of predictive feature for instances right next to training batch instances, and Model score is referred as predictive performance for future instances (e.g. validation dataset). We also compared the inference error, which is the accuracy of energy services for users.

2) AVERAGE TRAINING TIME

Large amount of computational resources are required while training the DL model. The objective is to computing the parameters of the learning model and achieve it for a short time period with efficient computing configuration. Therefore, to evaluate training time efficiency, we define a experimental metric of throughput as the total number of processing instances per training time for the same input dataset.

$$AverageTrainingTime = \frac{\tau_i}{b_i}, \tag{33}$$

where τ_i is the time interval of two consecutive updates i and $i-1$.

C. EXPERIMENTAL RESULTS

To assess the accuracy of the proposed scheme, household energy consumption data was used. The effect of the proposed batch size adjustment and scaling of the number of epochs on the prediction accuracy was assessed in terms of RMSE. The model with the lower RMSE provides precise prediction. The number of training instances and the number of epochs were determined based on detected degree of concept drift, profiled results such as hardware capacity, and the flow rate of input data streams. The experimental results showed the effectiveness of batch size adjustment and scaling of the number of epochs.

In experiment 1, we compared the run-time accuracy performance of the proposed scheme with that of the continuum scheme [7] in a cluster profiled as $\theta = (0.0001, 0.09)$ with $\lambda = 340$. Figure 9 (a) shows the inference loss performance of the proposed and continuum schemes with the parameter r = 1 and r = 2, respectively. Figure 9 (b) shows the inference loss performance of the proposed and continuum schemes with parameter r = 4 and r = 8, respectively. Because the execution time of each iteration varies according to the workload size (batch size, epoch), each time step is different. The experiments were conducted until both results showed convergence of model training. Because there were missing data in the dataset and there was multi-user class data (two user data), there was fluctuation of loss during training. Although the proposed scheme showed a large fluctuation in performance, it showed fast convergence of the CNN-LSTM model used for experiments, and it achieved good performance on average when we applied the retraining scheme with scheduling algorithm 1. Because Eq. (17) was designed for low recency and short-term instability of the model, which determines the performance of the trained model, the result showed a better convergence rate for the case of concept drift and showed stability for non-concept drift cases. For the proposed scheme and compared Continuum schemes with various parameters (r = 1, 2, 4, 8), inference loss was evaluated as 0.0630, 0.0744, 0.0755, 0.0810, and 0.0860 on average, respectively. Through the empirical evaluation, it was hard to find the best r value for continuum. Therefore, to figure out the requirements of the parameter variance, we conducted an additional experiment. The proposed scheme showed fast convergence of the model for concept drift. It also showed smaller loss on average. In addition, for the overlapped arrival of multi-user class data, (two user data), there were class-wise concept drift, known as skewed distribution, is occurred while training proceeded. In the same environment, inference loss was evaluated as 0.0729, 0.0669, 0.0659, 0.0885 and

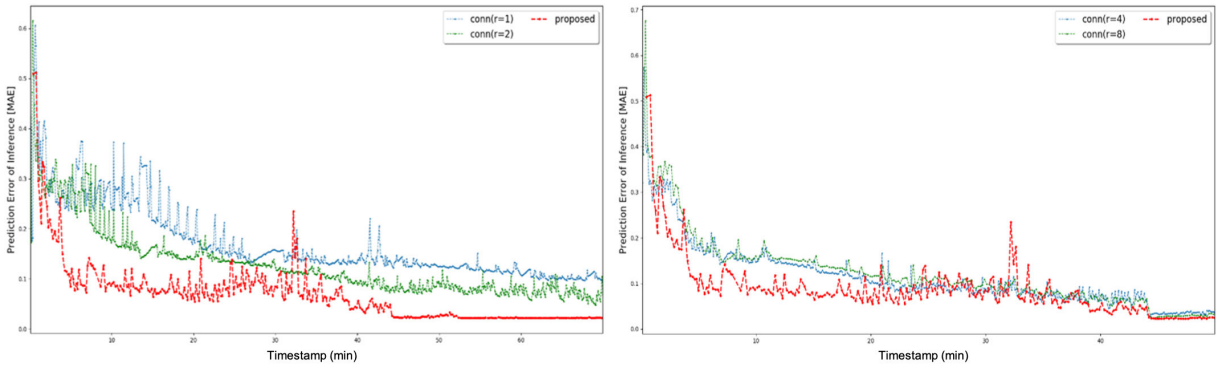

FIGURE 9. Exp 1 - Performance comparison of accuracy on the proposed and continuum scheme in cluster profiled as $\theta = (0.0001, 0.09)$ with $\lambda = 340$. Workload of training requests arrives in an exponential distribution of 340 tasks/sec. Each users send energy demand data (AMI class 1, 2) sequentially (Energy demand data are not sorted by datetime). Method: proposed scheme, and continuum scheme with various parameters (r=1, 2, 4, 8) (a) Prediction Error of Inference Performance of proposed and con(r=1, r=2) schemes (b) Prediction Error of Inference Performance of proposed and con(r=4, r=8) schemes (result of the proposed scheme is plotted with red color).

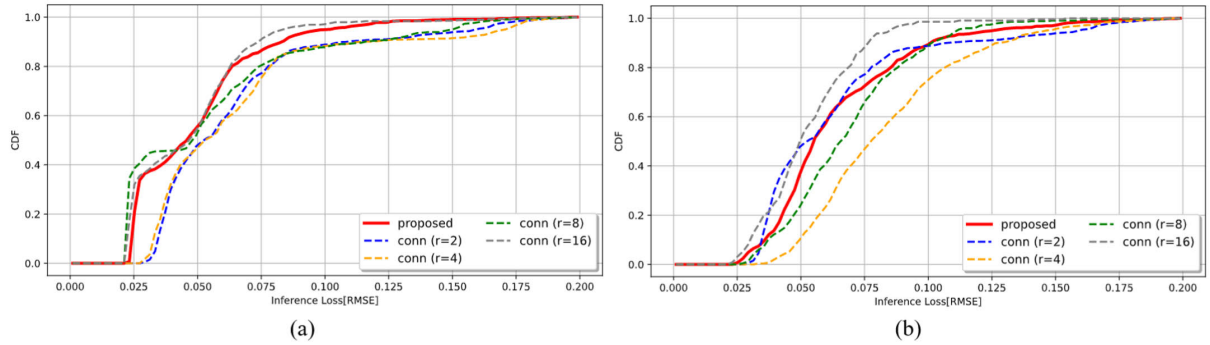

FIGURE 10. Exp 2 - Performance comparison of accuracy in form of CDF on the proposed and continuum scheme in cluster profiled as $\theta = (0.0001, 0.09)$ with $\lambda = 340$. Workload of training requests arrives in an exponential distribution of 340 tasks/sec. (a) Each users send energy demand data (AMI class 1, 2) sequentially (b) Two users continuously send energy demand data (AMI class 1, 2) at the same time (Energy demand data are sorted by datetime). Method: proposed scheme, and continuum scheme with various parameters (r=2, 4, 8, 16) (result of the proposed scheme is plotted with red color).

0.0702 in average respectively for comparison schemes. For concurrent user input for the multiple data, the proposed scheme incurs large training loss. From the large amount of concept drift, the proposed scheme tries to continuously fit into newly arrived data. It seems that missing of convergence result in low test loss performance. In this case, user-wise model management in the Velox framework [17] could be very helpful to enhance accuracy performance.

In experiment 2, we performed statistical analysis of proposed scheme in plotting of Cumulative distribution function (CDF) as shown in Figure 10. Figure 10 (a) shows the inference loss performance of the proposed and continuum schemes for the sequential arrival of multi-user class data. Figure 10 (b) shows the inference loss performance of the proposed and continuum schemes for the overlapped arrival of multi-user class data. The CDF curve of inference loss showed higher degree of convergence on CNN-LSTM model for the sequential arrival case. As mentioned in experiment 1, class-wise concept drift problem disturbed model convergence result in low accuracy. For the both cases, the low degree of inference loss were mostly observed in the CDF curve of the proposed scheme. For the sequential input case, the performance of proposed scheme showed almost top score. In addition, with its dynamic scheduling property, it can also be considered a general model with relatively low bias. Even for the overlapped arrival case, it showed good performance which was less accurate then conn (r=16) for entire domain and less accurate then conn (r=2) for $RMSE < 0.6$ domain. It means the proposed framework is more vulnerable to the class-wise concept drift from the side-effect of scheduling dynamicity.

In experiment 3, with the workload of training requests arriving in an exponential distribution of 340 tasks/sec, we measured the RMSE for training loss, inference loss, and model scores by applying various incremental learning solvers related to batch size allocation for all training iterations. As seen in Figure 11, we used different categories of user datasets in the case that each user sends energy demand data (AMI class 1, 2) sequentially (energy demand data are not sorted according to date and time). Continuum schemes with various parameters (r = 2, 4, 8, 16, 32, 64) were evaluated according to the test dataset after training was

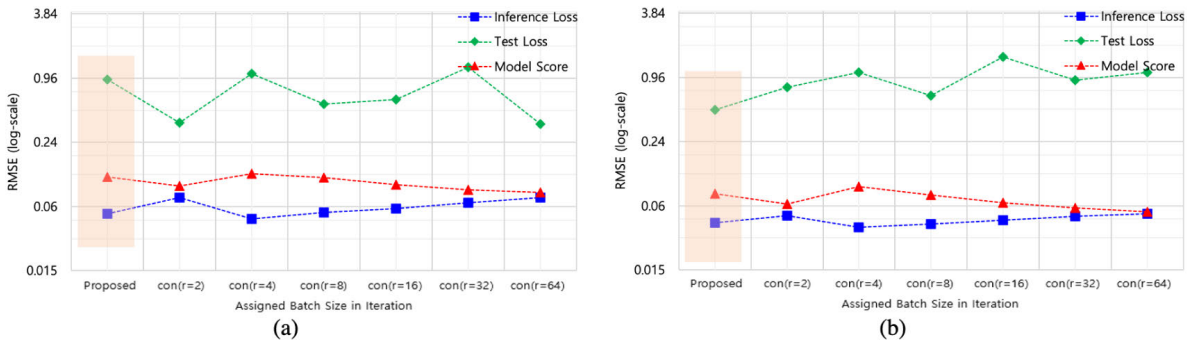

FIGURE 11. Exp 3 - Performance comparison of AMI prediction accuracy on inference loss (RMSE), training loss (MAE) and model score (RMSE) of adaptive incremental deep learning scheme in log scale and multi parameter configured continuum algorithm on difference characteristic class of incoming dataset (a) Each users send energy demand data (AMI class 1, 2) sequentially (b) Two users continuously send energy demand data (AMI class 1, 2) at the same time. (x-axis is the target schemes to be compared).

finished with the given dataset. The loss value for training, inference, and model scores were measured by applying various incremental learning solvers. The experiments showed that, although the RMSE of the proposed scheme was evaluated on various data sets, it showed comparatively good performance in all cases for inference loss, training loss and test loss. The proposed scheme showed excellent results in comparison to the other static configurations (variance in r). While it did not always produce the best result, adaptability has the advantage of allowing balanced performance in many aspects. Notably, the proposed model achieved good performance for the model score, which means that the proposed scheme is robust to class-wise concept drift.

In Experiment 4, to evaluate the computational performance of the proposed adaptive batch allocation scheme, we measured the average training time and queue length according to the progress of learning in a scenario in which two users send energy demand data (AMI class 1, 2) continuously. As shown in Figure 12 (a), the proposed scheme achieves lower average training time performance than comparison models (continuum) with various parameter configurations (r = 1, r = 2, r = 4, r = 8, r = 16, r = 32). The average training times spent in learning the same amount of input data were 0.0029243, 0.0116425, 0.0058195, and 0.0029366 second/instance, respectively, for proposed, con(r = 1), con(r = 2), conn(r = others) schemes. The speed gain term allows the batch size to be increased to minimize the time spent for initialization and operational overhead. If θ_{proc} is the dominant parameter, the influence of θ_{const} on the execution time is relatively reduced, allowing us to update the model more often. However, according to the actual device profile, theta is measured at a non-negligible value, so the cost reduction term is important for optimization of the execution time through batch allocation. Although various performance indicators must be considered in combination, we confirmed that our proposed method is effective for optimizing the execution time.

In addition, the queue stability conditions in Figure 12 (b) provide reliable queue management without queue overflow problems and coordinate the best epoch for training.

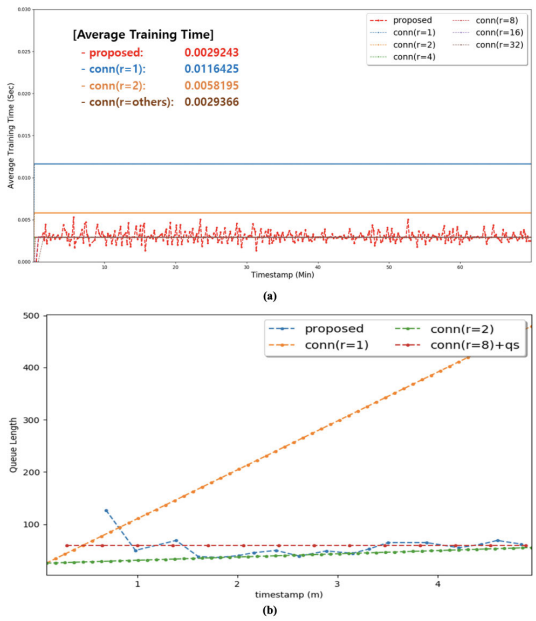

FIGURE 12. Exp 4 - Performance comparison of processing time on the proposed and continnum scheme in cluster profiled as $\theta = (0.0001, 0.09)$ with $\lambda = 340$ (x-axis is training timestamp in minutes, qs refers to queue stability in Eq. (15)) (a) Average training time of performance of proposed, continuum scheme with various parameter setting over entire dataset training (b) Measurement of queue length for the proposed and continuum scheme to evaluate effectiveness of queue stability condition.

This means that queue stability status should reserve as much time as possible to avoid exceeding specific training deadlines in every iteration. Queue stability conditions reserve as much time as possible without exceeding specific training deadlines for every iteration. As a result, there may be more ways to allocate batches and epochs in various combinations. Although the solution space is large for the proposed adaptive incremental deep learning scheme, relatively robust hyperparameters can be found for various concept drifts.

In Experiment 5, through the acceleration scheme of FPGU+GPU scheduling and implementation of FPGA logic development for the convolution layer, we assessed the processing time performance with respect to an inference task.

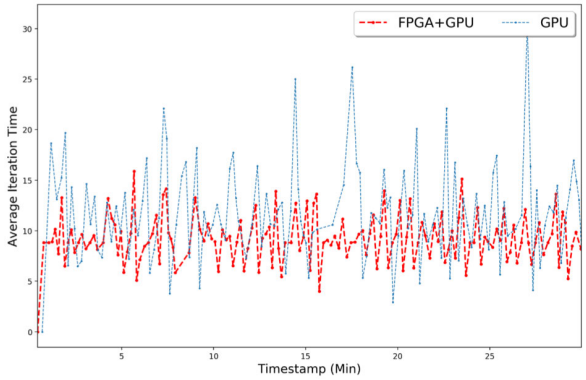

FIGURE 13. Exp 5 - Performance comparison of processing time on FPGA+GPU hybrid and GPU driven scheduling scheme with $\lambda = 340$(x-axis is training timestamp in minutes).

Figure 13 shows a comparison of the average iteration times with hybrid scheduling and GPU-oriented scheduling. From the profile result, 31 times acceleration of the processing time was achieved through the FPGA+GPU hybrid scheduling for the convolution layer. Because of the excellent FPGA performance on convolution layer processing, the results shows that the hybrid scheduling scheme has achieves about 1.2 times acceleration of the iteration time.

VII. CONCLUSION

In this article to accelerate the deployment procedure of a deep neural network after model training, we proposed an accelerated edge cloud system for energy data stream processing based on an adaptive incremental deep learning scheme. The proposed system is a real-time training/inference system that deploys AMI data through distributed edge computing. As the volume of data increases and models become more complex, a faster training method is required to process energy data streams. We propose an acceleration scheme with an adaptive incremental learning solver. For incremental deep learning, it is essential to determine the term of the retraining time to improve data incorporation latency, which determines the mini-batch with the best training time to update the model. We proposed a multi-criteria utility function for batch size and epoch. Also, we proposed a heuristic to find a multi-criteria solution of the cost function with the gradient descent method (sub-optimal). It was empirically shown that it converges to local-minima with only 5 steps on average. In addition, a resource-scheduling scheme manages various accelerator resources to accelerate deep-learning processing while minimizing the computational cost. When adaptive incremental deep learning sends a DL task to the resource scheduler, it is able to accelerate DL task processing using heterogeneous accelerators. Furthermore, with a deadline-constrained DL pipeline scheduling scheme, the execution time of sub-tasks is reserved in a sub-deadline. As a result, through the adaptive incremental learning and accelerated edge cloud, we can expect increment of the processing time of energy stream data processing. To evaluate the proposed system, we implemented an accelerated DL cluster stream data processing system based on the incremental learning scheme. Also, we use a real dataset as the stream benchmark data on user energy demands with the AMI data of more than 50 subscribers. The experimental results showed that our method achieves good performance with adaptive batch size and epoch with incremental learning while guaranteeing a low inference loss, a high model score, and queue stability with cost-efficient processing. Several experiments were conducted, which demonstrated that the proposed incremental deep-learning scheme quickly accepted concept drift in diverse data patterns with 2 times acceleration of execution time when we conducted a comparison of the elapsed time to reach convergence of training. It also provides real-time model updating, although it loses a little accuracy regarding sub-optimization in the demand prediction model. In addition, heterogeneous accelerator (FPGU, GPU) resource scheduling through layer partitioning in the edge cloud shows that the performance of the CNN-LSTM model for AMI data processing can be improved. Processing time is 31 times faster for FPGA+GPU hybrid scheduling in the convolution layer, and for the entire iteration time, 1.2 times processing time acceleration was achieved. Through the proposed framework, computation of time-critical task for real-time energy analysis with high level accuracy is achieved, result in low risk service of demand prediction and power generation plan. In addition, through the sophisticated incremental learning, low cost training is available.

REFERENCES

[1] Y. Lee, B. Youn, J. Kim, J. Jeong, and H. Lee, "Kemri power economy review," KEPCO, Naju-si, South Korea, Tech. Rep. 19, 2016.

[2] D.-K. Kang and C.-H. Youn, "Real-time control for power cost efficient deep learning processing with renewable generation," *IEEE Access*, vol. 7, pp. 114909–114922, 2019.

[3] S. Bae, "An accelerated streaming data processing scheme based on cnn-lstm hybrid model in energy service platform," M.S. thesis, Dept. Elect. Eng., KAIST, Daejeon, South Korea, 2019.

[4] T.-Y. Kim and S.-B. Cho, "Predicting residential energy consumption using CNN-LSTM neural networks," *Energy*, vol. 182, pp. 72–81, Sep. 2019.

[5] G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand, "Exponentially weighted moving average charts for detecting concept drift," *Pattern Recognit. Lett.*, vol. 33, no. 2, pp. 191–198, Jan. 2012.

[6] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, and F. Herrera, "A survey on data preprocessing for data stream mining: Current status and future directions," *Neurocomputing*, vol. 239, pp. 39–57, May 2017.

[7] H. Tian, M. Yu, and W. Wang, "Continuum: A platform for cost-aware, low-latency continual learning," in *Proc. SoCC*, 2018, pp. 26–40.

[8] S. Wang, L. L. Minku, D. Ghezzi, D. Caltabiano, P. Tino, and X. Yao, "Concept drift detection for online class imbalance learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Aug. 2013, pp. 1–10.

[9] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Mach. Learn.*, vol. 23, no. 1, pp. 69–101, Apr. 1996.

[10] B. Wang and J. Pineau, "Online bagging and boosting for imbalanced data streams," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 12, pp. 3353–3366, Dec. 2016.

[11] N. Dehak, R. Dehak, J. R. Glass, D. A. Reynolds, and P. Kenny, "Cosine similarity scoring without score normalization techniques," in *Proc. Odyssey*, 2010, p. 15.

[12] Y.-W. Cheung and K. S. Lai, "Lag order and critical values of the augmented Dickey–Fuller test," *J. Bus. Econ. Statist.*, vol. 13, no. 3, pp. 277–280, 1995.

[13] R. R. Mohassel, A. Fung, F. Mohammadi, and K. Raahemifar, "A survey on advanced metering infrastructure," *Int. J. Elect. Power Energy Syst.*, vol. 63, pp. 473–484, Dec. 2014.
[14] Y. Kim, N.-G. Myoung, and S.-Y. Lee, "Study on AMI system of KEPCO," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Nov. 2010, pp. 459–460.
[15] P. Domingos and G. Hulten, "A general framework for mining massive data streams," *J. Comput. Graph. Statist.*, vol. 12, no. 4, pp. 945–949, Dec. 2003.
[16] T. Bian and Z.-P. Jiang, "Reinforcement learning for linear continuous-time systems: An incremental learning approach," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 2, pp. 433–440, Mar. 2019.
[17] D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan, "The missing piece in complex analytics: Low latency, scalable model management and serving with velox," 2014, *arXiv:1409.3809*. [Online]. Available: http://arxiv.org/abs/1409.3809
[18] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *Proc. 14th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2017, pp. 613–627.
[19] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Rev.*, vol. 60, no. 2, pp. 223–311, Jan. 2018.
[20] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale DNN processor for real-time AI," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018.
[21] J. Yinger, E. Nurvitadhi, D. Capalija, A. Ling, D. Marr, S. Krishnan, D. Moss, and S. Subhaschandra, "Customizable FPGA OpenCL matrix multiply design template for deep neural networks," in *Proc. Int. Conf. Field Program. Technol. (ICFPT)*, Dec. 2017, pp. 259–262.
[22] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2017, pp. 15–24.
[23] A. Prost-Boucle, A. Bourge, F. Petrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on FPGA," in *Proc. 27th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2017, pp. 1–7.
[24] R. Inta, D. J. Bowman, and S. M. Scott, "The 'Chimera': An off-the-shelf CPU/GPGPU/FPGA hybrid computing platform," *Int. J. Reconfigurable Comput.*, vol. 2012, p. 2, Mar. 2012, Art. no. 241439.
[25] S. K. Rethinagiri, O. Palomar, J. A. Moreno, O. Unsal, and A. Cristal, "An energy efficient hybrid FPGA-GPU based embedded platform to accelerate face recognition application," in *Proc. IEEE Symp. Low-Power High-Speed Chips (COOL CHIPS XVIII)*, Apr. 2015, pp. 1–3.
[26] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. 13th EuroSys Conf.*, 2018, p. 3.
[27] L. Bottou and Y. L. Cun, "Large scale online learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2004, pp. 217–224.
[28] P. Yin, P. Luo, and T. Nakamura, "Small batch or large Batch?: Gaussian walk with rebound can teach," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 1275–1284.
[29] P. M. Radiuk, "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," *Inf. Technol. Manage. Sci.*, vol. 20, no. 1, pp. 20–24, Jan. 2017.
[30] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[31] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*. [Online]. Available: http://arxiv.org/abs/1609.04747
[32] D.-S. Kim, "Adaptive workflow scheduling scheme based on the colored petri-net model in cloud," M.S. thesis, Dept. Elect. Eng., KAIST, Daejeon, South Korea, 2014.
[33] D. Alahakoon and X. Yu, "Smart electricity meter data intelligence for future energy systems: A survey," *IEEE Trans. Ind. Informat.*, vol. 12, no. 1, pp. 425–436, Feb. 2016.
[34] W. Chen, K. Zhou, S. Yang, and C. Wu, "Data quality of electricity consumption data in a smart grid environment," *Renew. Sustain. Energy Rev.*, vol. 75, pp. 98–105, Aug. 2017.
[35] K. Kornelsen and P. Coulibaly, "Comparison of interpolation, statistical, and data-driven methods for imputation of missing values in a distributed soil moisture dataset," *J. Hydrologic Eng.*, vol. 19, no. 1, pp. 26–43, Jan. 2014.
[36] L. Qu, J. Hu, L. Li, and Y. Zhang, "PPCA-based missing data imputation for traffic flow volume: A systematical approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 10, no. 3, pp. 512–522, Sep. 2009.
[37] R. Malarvizhi and A. S. Thanamani, "K-nearest neighbor in missing data imputation," *Int. J. Eng. Res. Develop.*, vol. 5, no. 1, pp. 5–7, 2012.
[38] J. Yoon, J. Jordon, and M. van der Schaar, "GAIN: Missing data imputation using generative adversarial nets," 2018, *arXiv:1806.02920*. [Online]. Available: http://arxiv.org/abs/1806.02920
[39] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko, "Long-term recurrent convolutional networks for visual recognition and description," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 2625–2634.
[40] C.-J. Huang and P.-H. Kuo, "A deep CNN-LSTM model for particulate matter ($PM_{2.5}$) forecasting in smart cities," *Sensors*, vol. 18, no. 7, p. 2220, 2018.
[41] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2015, pp. 161–170.

SEONG-HWAN KIM (Associate Member, IEEE) received the B.S. degree in media and communications engineering from Hanyang University, Seoul, South Korea, in 2012, and the integrated master's and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2020. He is currently a Postdoctoral Researcher with the Network and Computing Laboratory, KAIST. His research interests include cloud brokering systems, deep learning, and energy service platform and others.

CHANGHA LEE received the B.S. degree in electronic engineering from Hanyang University, Seoul, South Korea, in 2018. He is currently pursuing the M.S. degree with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea. Since 2018, he has been a member of the Network and Computing Laboratory, KAIST. His current research interests include deep learning platform, sequential learning, and energy prediction service platform.

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. Since 1997, he has been a Professor with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea. He was an Associate Vice President of Office of Planning and Budgets with KAIST, from 2013 to 2017. He wrote a book on *Cloud Broker and Cloudlet for Workflow Scheduling* (Springer, 2017). His research interests include distributed high-performance computing systems, cloud computing systems, edge computing systems, deep learning framework, and others. He received the Best Paper Award from CloudComp 2014. He was the General Chair of the 6th EAI International Conference on Cloud Computing (Cloud Comp 2015), KAIST, in 2015. He was a Guest Editor of *IEEE Wireless Communications*, in 2016.