An Energy-Efficient Tuning Method for Cloud Servers Combining DVFS and Parameter Optimization

Weiwei Lin, *Member, IEEE*, Xiaoxuan Luo, ChunKi Li, Jiechao Liang, Guokai Wu, and Keqin Li, *Fellow, IEEE*

***Abstract*—Emerging cloud computing applications place a growing demand on resources, leading to increasingly large data centers with significant energy consumption and carbon emissions. Various research conduct optimization methods to improve the energy efficiency of the server in the cloud data center. However, most existing optimization methods are designed for specific applications, thus making it difficult to handle complex cloud environments. In this paper, we propose a general parameter optimization method called MPOD to improve the energy efficiency of cloud servers in real time. MPOD considers issues in the cloud environment, such as SLA guarantee, user privacy, and dynamic workloads. We introduce energy efficiency curves to DVFS, implementing a low-overhead, fast response, and general frequency optimization strategy. Moreover, we design a workload classification framework and three prediction models based on machine learning algorithms to achieve accurate and adaptive Linux kernel parameters optimization. According to the experiment, MPOD can improve the energy efficiency of the server by an average of 30.5%, 20.1%, 10.8% in BenchSEE, SERT and TPC-H, respectively.**

***Index Terms*—Cloud data center, DVFS, energy-efficiency modeling, energy-efficiency optimization.**

I. Introduction

CLOUD data center is a new type of data center built on cloud computing technologies, which accomplishes a high degree of integration of various resources through virtualizing IT equipment. It provides users with safe, reliable, efficient, and flexible services in the form of IaaS, PaaS, and SaaS. With the development of technologies such as artificial intelligence, the Internet of things, and blockchain, emerging applications not only have greater demand in terms of the number of resources but also place greater demands on the efficiency of resource usage. Cloud data centers are scaling further to meet these increasingly complex business needs, which brings a significant increase in cluster deployment and maintenance costs [1]. With the continuously decreasing PUE of the data center, infrastructure accounts for an increasingly smaller share of data center energy consumption. Improving data center energy efficiency requires a greater focus on IT equipment [2]. Servers account for 70-80% of the energy consumption of major IT equipment [3]. Improving the energy efficiency of servers is an important measure to achieve energy savings and emission reduction in cloud data centers.

An important method to improve server energy efficiency is to tune various parameter configurations. Due to its flexibility and efficiency, parameter optimization is widely used in a variety of scenarios such as Big Data applications [4], machine learning hyperparameter adjustment [5], and distributed database query acceleration [6]. Servers and applications have numerous parameters with wide value ranges and suitable scenarios. The optimal values for parameters are difficult to obtain by manual adjustment. Consequently, many researchers have designed parameter optimization methods for different applications and usage scenarios. However, existing parameter optimization methods are accompanied by the following problems when applied to cloud servers:

- Servers and applications have a large number of parameters with wide value range space and complex interactions, which make parameter optimization much more difficult.
- In online optimization scenarios, improper parameter configurations may bring certain fluctuations in server performance, affect the quality of cloud services and even lead to SLA violations.
- Different parameters have different associations with application performance and energy consumption. Most existing research only targets optimization for a single application with specified parameters. Without the ability to perform targeted tuning measures based on different workloads, the optimization methods are unable to achieve good effects in a complex cloud environment.

Manuscript received 3 February 2023; revised 25 June 2023; accepted 23 August 2023. Date of publication 28 August 2023; date of current version 6 December 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62072187, in part by the Guangdong Major Project of Basic and Applied Basic Research under Grant 2019B030302002, in part by Guangdong Marine Economic Development Special Fund Project under Grant GDNRC[2022]17, in part by Guangzhou Development Zone Science and Technology Project under Grant 2021GH10, and in part by the Major Key Project of PCL under Grant PCL2023AS7-1. Recommended for acceptance by J. Aghaei. *(Corresponding author: Xiaoxuan Luo.)*

Weiwei Lin is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510641, China, and also with the Peng Cheng Laboratory, Shenzhen 518066, China (e-mail: linww@scut.edu.cn).

Xiaoxuan Luo, ChunKi Li, Jiechao Liang, and Guokai Wu are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510641, China (e-mail: 202210188550@mail.scut.edu.cn; 202021044488@mail.scut.edu.cn; 202121045094@mail.scut.edu.cn; 202120144402@mail.scut.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TCC.2023.3308927

- Many optimization methods require access to application performance metrics to evaluate tuning effects. However, cloud service providers can not directly collect user data for performance evaluation due to data security concerns. How to accurately capture application performance is a critical issue to address in cloud server parameter optimization.
- Existing optimization methods mainly focus on optimization effects while neglecting speed. In the highly dynamic cloud environment, lagging optimization may also lead to server performance degradation, thus affecting user services and even violating SLA.

To solve the above problems, we propose a multi-model prediction-based parameter optimization method with DVFS (MPOD). With the basic guarantee of SLA and privacy security, this method can identify the type of workload on which the server is running in real time and perform targeted optimization for the specific application, thus enabling it to adapt to the complex cloud environment. Moreover, for applications in the virtualization layer, the actual workload still needs to be executed by physical machines. Capturing the performance metrics of the physical machines and optimizing for the physical machines, it is still possible to capture the characteristics of the applications and improve the energy efficiency of the workload in the virtualization layer. Therefore, the above approach can also be adapted to the virtualized server environment.The main contributions of this paper are as follows:

- We propose a DVFS algorithm based on energy efficiency curves, which achieves low overhead and fast response to cope with the complex scenario of highly dynamic changes in cloud servers.
- We design a two-tier workload classification framework based on KNN to identify the type of application carried by the server, enabling MPOD to perform targeted optimization measures according to the recognized application type.
- We build multiple models to accommodate the parameter optimization in the cloud environment. Performance models based on system-level performance metrics can evaluate application throughput while protecting user privacy. Max performance models and energy efficiency models can minimize performance disruptions in real-time parameter optimization.
- We implement our proposed method based on BenchSEE data and show the effectiveness of MPOD on BenchSEE, SERT and TPC-H. In addition, we also demonstrate the low overhead of MPOD during optimization.

The rest of this paper is organized as follows. Section II reviews related work in parameter optimization for cloud servers. Section III details the design of MPOD. Section IV shows the experiments of MPOD in a real server environment. Section V summarizes this paper.

II. Related Work

Over the past few years, parameter optimization methods for cloud applications have been a critical research topic. Scholars have proposed various optimization methods depending on the application scenario and optimization objective. Big data applications (such as Hadoop, Spark, and Storm.) are mainstream scenarios in cloud computing, which contain many configurations related to component interaction, process parallelism, and I/O throughput. Improper parameter settings often lead to reduced application throughput, so many researchers have proposed application parameter optimization strategies for Big Data scenarios such as collaborative scheduling [7], dynamic cluster environment [8], and performance prediction [9]. Parameter optimization is also widely used in high-performance computing [10], distributed file system [11] and database [12].

In addition to application parameters, the operating system also contains numerous parameters that affect hardware performance, power consumption, and application interaction. With the appropriate configuring of such parameters, a server can run user workload optimally to improve energy efficiency. In addition, the operating system parameters do not require manipulation of user applications, making it more suitable for the cloud environment.

Sánchez et al. [13] explore the performance impact of NUMA policies and prefetchers on various benchmarks. In order to resolve the high optimization overhead caused by large search space, the authors use an unsupervised clustering algorithm to cluster applications with similar behavior based on 19 performance counters. Furthermore, they use machine learning algorithms such as ANN and SVM to determine the optimal NUMA policy and prefetcher setting for each type of workload. Luan et al. [14] discover that the number of CPU cores and core layout significantly affect parallel program performance, so they propose Otter tune CPU core configurations. Otter first determines the primary performance trend of a parallel program using a few samples, then applies interpolation to fit a function between the number of CPU cores, the CPU core distribution, and the program throughput. Finally, they use the golden section method for parameter optimization to maximize parallel program throughput. Qi et al. [15] conduct BIOS parameter optimization for the TaiShan2280 server with Kunpeng 920 processor. Due to the large search space and complex relationship, the authors develop a Markov model to guide the control decision of BIOS parameters and combine deep Q-networks to find the optimal BIOS configurations for various workloads. However, these methods require a reboot of the application or server to take effect, which results in long response times and business interruptions. Consequently, such optimization methods are challenging to apply in the cloud environment that has to ensure the stability of the user services.

Dynamic voltage frequency scaling (DVFS) is a common technique for real-time server optimization, which reduces wasted computing resources and improves server energy efficiency by adjusting the processor chip voltage and frequency to match the running workloads. The EEDTSA proposed by Mishra et al. [16] establishes the energy consumption correspondence matrix between frequency-voltage pairs and workloads by considering the number of instructions, execution order, and deadline requirements of each workload. EEDTSA selects the optimal voltage and frequency of CPU cores based on the energy consumption correspondence matrix to reduce the overall energy usage. The MDQ-CR algorithm proposed by Asghari et al. [17] extends DVFS from a single-task optimization to long-term multi-task optimization. MDQ-CR uses coral reefs optimization (CRO) to determine the optimal DVFS state for the initial workload assignment. Then, the deep Q-network is applied for DVFS state assignments of subsequent workloads. The deep Q-network can converge to the optimal global solution as it considers the long-term energy consumption varies with the help of a Markov game model. Nevertheless, the overhead of MDQ-CR is an issue that cannot be ignored. We experimentally demonstrate that using complex DVFS algorithms in a dynamic environment may lead to delays in tuning, thus reducing the throughput of user applications. Therefore, many researchers have also used low-overhead DVFS techniques for scenario-specific optimization. Lee et al. [18] extend the ondemand frequency governor for polling-based IO services. The proposed PollO governor can improve energy efficiency by 26.93% for IO-intensive services. However, we discover that simple DVFS techniques have uneven optimization effects and may only work for some applications in the cloud environment.

Linux kernel parameters are also well suited for optimization in cloud servers due to their real-time effect. To maximize network throughput, Gembala et al. [19] apply a genetic algorithm to optimize 27 Linux kernel network parameters which affect various aspects such as TCP transfer queue size, connection control, and window tuning. Junqing et al. [20] use Linux memory kernel parameters to optimize Redis performance. The authors classify Redis applications into three scenarios and use the eBPF technique for gathering memory performance metrics to identify the different scenarios. The authors consider a combination of multiple decision tree models constructed from random forests to filter out the essential kernel parameters for each scenario to optimize separately.

Parameter optimization has been applied to many application scenarios. However, as mentioned above, many studies do not consider factors such as SLA, user privacy, complex and dynamic applications in the cloud environment. As a result, it is not easy to apply existing parameter optimization methods to a real cloud environment.

III. MPOD Design

Processor frequency is a crucial factor affecting server energy efficiency, and operating system kernel parameters are ideal for energy efficiency optimization in a cloud environment. Therefore, we propose an energy efficiency optimization method for cloud servers based on these two parameters called MPOD (multi-model prediction-based parameter optimization method with DVFS). In this section, we will introduce the design and workflow of MPOD.

As Section II mentions, parameter optimization requires fitting complex functional relationships between multiple parameters and finding the best result in a large search space. Therefore, many studies have applied techniques such as evolutionary algorithms and deep neural networks to parameter optimization. Although such algorithms can accurately evaluate the impact of parameters in a high-dimensional space and have good optimization effects, they are not suitable for some parameters because

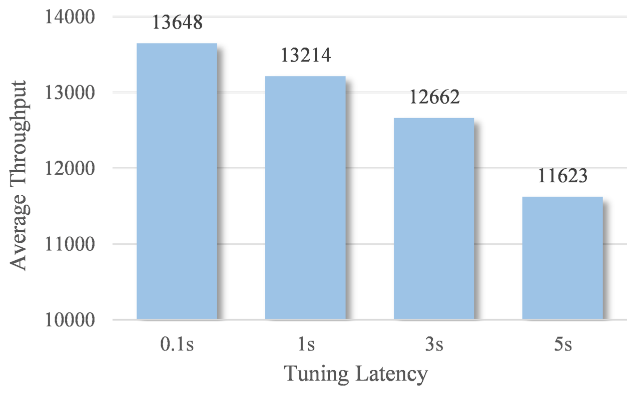

Fig. 1. Comparison of Compress average throughput with different optimization latency.

of their long tuning time and high resource usage. Frequency is a typical example. To evaluate the impact of lagging tuning on application performance due to excessive overhead of the optimization algorithm, we run a benchmark named Compress using the same frequency optimization strategy with different latency. Fig. 1 shows the average throughput of the server at different optimized latency, where the Compress workload runs for 200 s at 100%, 75%, 50% and 25% load levels respectively. As we can see, higher latency brings more performance loss, with a maximum of 14.8% throughput degradation in this experiment. This means that different parameters require different optimization methods according to their characteristics. As a result, we develop different granularity optimization strategies for frequency and Linux kernel parameters:

- *Fine granularity frequency optimization*: frequency variations greatly affect server performance and need to be tuned in a fine granularity manner to ensure basic server performance during optimization.
- *Coarse granularity Linux kernel parameter optimization*: Linux kernel parameters do not need to be changed frequently to adapt the short-term load fluctuations. In addition, the relationship between Linux kernel parameters on energy efficiency is more complex. Thus this parameter optimization requires less real-time and is more concerned with workload identification as well as accurate evaluation.

The overall architecture of MPOD is shown in Fig. 2. In the following contents of Section III, we will describe the design of each element in MPOD in more details.

A. Frequency Optimization

For current mainstream processor architectures, each core can be individually set to a frequency that ranges from high to low with several discrete values (1). Higher frequency has higher performance and power consumption. Matching the frequency to the load level can improve server energy efficiency.

$$F = \{f_1, f_2, \ldots, f_n |\ f_i > f_j \quad if \quad i < j\} \tag{1}$$

The main reasons for the high overhead of current DVFS are accurate identification of the relationship between frequency,

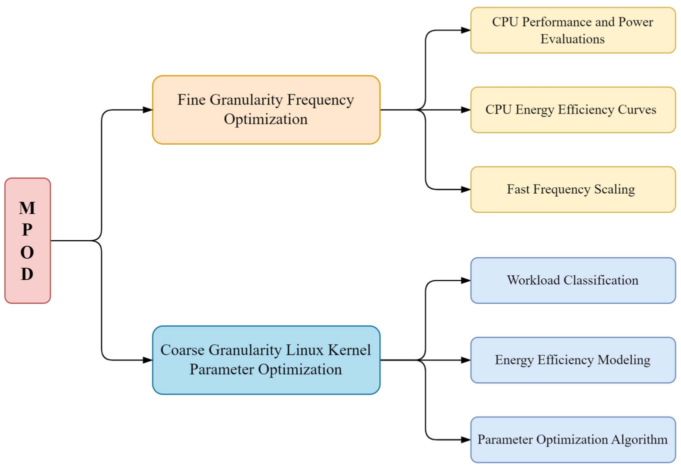
M P O D
Fine Granularity Frequency Optimization
CPU Performance and Power Evaluations
CPU Energy Efficiency Curves
Fast Frequency Scaling
Coarse Granularity Linux Kernel Parameter Optimization
Workload Classification
Energy Efficiency Modeling
Parameter Optimization Algorithm

Fig. 2. Basic architecture of MPOD.

load level, and energy efficiency, coupled with complex optimization based on an objective function. We simplify the performance and power consumption evaluation process using several simple constraints and combine them with the CPU energy efficiency curve to determine the best frequency configuration. Moreover, we design a fast scaling mechanism to accelerate frequency optimization. By applying the above approach, we achieve a simple and effective frequency optimization strategy for the cloud server.

1) Relative Evaluation: In order to determine the optimal CPU frequency for energy efficiency in a cloud environment, the DVFS algorithm needs to evaluate not only the CPU performance requirements of user applications to prevent SLA violations but also the variation in CPU power consumption due to different frequencies. In frequency optimization, we divide the server energy efficiency evaluation into performance evaluation and power evaluation, then combine the two parts to derive the energy efficiency evaluation of the server.

In terms of performance evaluation, it is common practice to use metrics such as performance counters to predict the exact throughput of an application. However, accurate prediction inevitably brings high overhead. To reduce overhead, we evaluate relative load levels directly using CPU utilization (without iowait utilization). This approach does not accurately capture application throughput, but identifies whether the current frequency is meeting the CPU performance requirement of the application and thus evaluates the likely application throughput change after the frequency adjustment. The detailed evaluations are as follow:

If the CPU frequency is set from f_i to f_j, the CPU utilization will change from u_i to u_j. The application throughput T changes in one of three ways. As (2) shows, if the CPU utilization before and after the adjustment is less than 1, both frequency settings meet the maximum performance required by the application, so the throughput will not change. As shown in (3), if the CPU utilization before tuning is 1, the application can fully exhaust the computing resources provided by the server. Unless f_i just meets the maximum performance requirement, any frequency increase will improve the application's throughput. (4) is similar to (3), if the CPU utilization becomes 1 after a frequency reduction, the application throughput is more likely to decrease.

$$T(f_j,u_j) = T(f_i,u_i), if\ u_i < 1\ and\ u_j < 1 \tag{2}$$

$$T(f_j,u_j) \geq T(f_i,u_i), if\ f_j > f_i\ and\ u_i = 1 \tag{3}$$

$$T(f_j,u_j) \leq T(f_i,u_i), if\ f_j < f_i\ and\ u_j = 1 \tag{4}$$

Similar to the performance evaluation, we still use relative evaluation to reduce the overhead of power evaluation by converting the exact power calculation into a relative power comparison. The CPU power P_{cpu} not only depends on the frequency setting but is also affected by the load level of the application. As shown in (5) and (6), running at a lower frequency for the same application throughput will reduce CPU power consumption. At the same frequency setting, the heavier load carried by the server, the more energy will be consumed.

$$P_{cpu}(f_i,T_0) > P_{cpu}(f_j,T_0), if\ f_i > f_j \tag{5}$$

$$P_{cpu}(f_0,T_i) > P_{cpu}(f_0,T_j), if\ T_i > T_j \tag{6}$$

Based on the above simplification, we obtain the primary performance and power optimization strategies, respectively. We assume that the CPU performance required, i.e., the maximum application throughput T_{max}, is constant for a short-term workload (e.g. about 5 seconds). Thus the load demand provided by users before and after tuning is consistent and the change in server energy efficiency level is comparable.

When the CPU utilization is less than 1, the real-time application throughput T can reach the maximum throughput T_{max}. If the CPU utilization is still less than 1 after reducing the frequency, it will reduce the CPU power consumption while maintaining the same application throughput, thus improving the energy efficiency of the server. However, if the CPU utilization is 1 after reducing the frequency, the CPU power consumption is reduced, but the application throughput is also likely to be reduced. Since we do not calculate actual performance and power values, it is difficult to determine the energy efficiency of the server in this case through relative performance and power evaluation strategy. In order to combine the simplified strategies for overall server energy efficiency optimization, we introduce the CPU energy efficiency curves to determine the optimal frequency configuration.

2) CPU Energy Efficiency Curves: Since MPOD uses CPU utilization to assess application load level, the CPU energy efficiency curves discussed in this section refer to the trend between different frequencies, CPU utilization, and energy efficiency.

Introducing CPU energy efficiency curves can improve the accuracy of CPU efficiency evaluation. First, different CPUs perform different energy efficiency trends, and not all servers are at their best energy efficiency when CPU utilization reaches 100% [21]. Moreover, the performance and power consumption do not change in equal proportion at different frequency levels, even for the same CPU. Fig. 3 shows the maximum throughput and server power consumption variation of the Compress benchmark at each frequency level for the Intel(R) Xeon(R) Gold 6248 CPU @ 2.50 GHz. When the CPU frequency increases from 2.4 GHz to 2.5 GHz, the application throughput increases by 962 while the power consumption increases by 65 W. When the

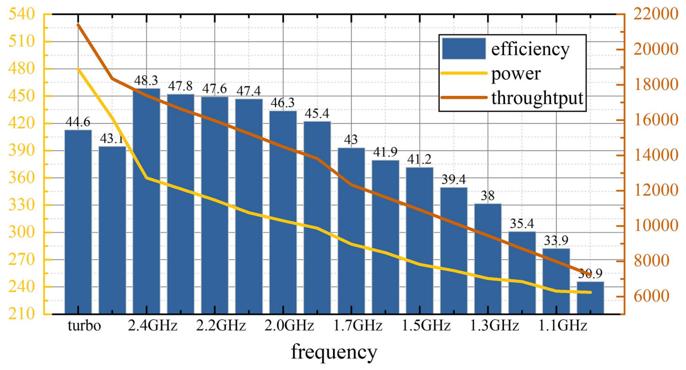

Fig. 3. Comparison of performance, power consumption and energy efficiency of each frequency level at full load.

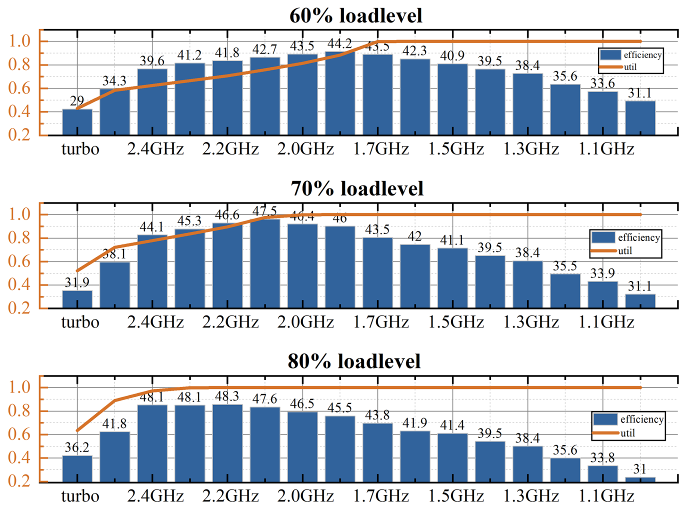

Fig. 4. CPU energy efficiency and utilization curves evaluated using Bench-SEE Compress with load levels of 60%, 70% and 80%.

CPU frequency increases from 2.3 GHz to 2.4 GHz, the application throughput increases by 760 while the power consumption increases by only 13 W. Such difference in energy efficiency between the frequency levels is more pronounced when the Intel turbo boost technology [22] is enabled.

We can gather the energy efficiency of each frequency level under different utilization by pressurizing the server using benchmarks with different load levels. This is the most common way to get the CPU energy efficiency curves. Then, we can determine the optimal utilization interval for each frequency according to the CPU energy efficiency curves. Fig. 4 will be used as an example to introduce how to obtain the optimal utilization interval. Fig. 4 shows the CPU energy efficiency and utilization curves when running three different load levels of the Compress benchmark. For 2.1 GHz frequency, it can achieve optimal energy efficiency at 70% load level. However, at an 80% load level, increasing the frequency can achieve higher energy efficiency. While decreasing the frequency can achieve higher energy efficiency at a 60% load level. For the 60% and 80% load levels, the CPU utilization is 75.8% and 100%, respectively; therefore [0.758, 1] is a possible optimal utilization interval for 2.1 GHz.

Using multiple sets of experiments similar to Fig. 4, we evaluate multiple sets of possible optimal utilization intervals for each frequency level. Taking the intersection of all possible optimal utilization intervals is the final optimal energy efficiency interval for that frequency class. For example, if there are three possible optimal utilization intervals for 2.1 GHz [0.7, 1.0], [0.75, 0.95], [0.72, 0.88], then the final optimal utilization interval for 2.1 GHz is [0.75, 0.88]. Although it is true that the frequency should be decreased at a utilization rate of 0.7, this value is not a critical value and the frequency should also be reduced when the utilization rate is 0.72. The reason for this situation is that the load level we set during the experiment is not continuous, and it is possible that 0.75 is not the real lower bound of the optimal utilization of 2.1 GHz. We can only keep approaching the real optimal utilization interval through the experiment. In general, the smaller the interval between the load levels set in the experiment, the more accurate the evaluated optimal energy efficiency utilization interval is.

For a specified application, the server energy efficiency is optimized by keeping the CPU utilization within the optimal interval. The optimal frequency can be set through the relative performance and power evaluations in Section III-A1 together with the optimal CPU utilization interval determined based on the energy efficiency curve. The optimization process does not need to obtain actual application throughput and server power consumption data.

3) Fast Frequency Scaling Mechanism: The tuning strategy of MPOD is to compare the current energy efficiency state with the previous energy efficiency state and gradually adjust it to the theoretically optimal frequency. Therefore, multiple energy efficiency evaluations are required from the initial to optimal frequency. During the optimization process, the energy efficiency of the server is not ideal. Level-by-level tuning may lead to a long optimization time, especially when the load level changes dramatically (for example, from an idle state to a full load state). Even if we can significantly improve the server energy efficiency under the optimal frequency configuration, long-term optimization will still decline the overall server energy efficiency. To solve this problem, we introduce a fast frequency scaling mechanism to reduce the frequency optimization time. This mechanism dynamically adjusts the optimization step according to the tuning behavior, so the server can be quickly configured to the frequency with the best energy efficiency.

Algorithm 1 is the pseudo-code of DVFS in MPOD. According to the optimal utilization interval, the status of the core can be divided into three categories. The current frequency configuration can achieve optimal energy efficiency if the core utilization is within the interval. When the utilization is exceptionally high, and the frequency is increased twice continuously, it is most probable that the frequency will need to be increased significantly to achieve optimal energy efficiency. Otherwise, searching around the current frequency step by step is enough. When the CPU utilization is significantly below the lower bound, it indicates that the frequency configuration is improper and should be dramatically decreased. When the CPU utilization is

Algorithm 1: Energy Efficiency Based DVFS With Fast Frequency Scaling Mechanism.

```
Input: Optimal utilization interval with upper bound $U_{up}$
       and lower bound $U_{low}$
Output: The optimal frequency of each core
 1: while not interrupt program do
 2:   for each core in CPU do
 3:     get real time utilization $u$ and frequency $f$
 4:     if $U_{low}(f) < u < U_{up}(f)$ then
 5:       maintain current frequency configuration
 6:     end if
 7:     if $u > U_{up}(f)$ then
 8:       if the core has increased frequency at least twice
          continuously then
 9:         increase 4 frequency levels
10:       else
11:         increase 1 frequency level
12:       end if
13:     end if
14:     if $u < U_{low}(f)$ then
15:       decrease frequency by $U_{low}(f)/u$ levels
16:     end if
17:   end for
18: end while
```

slightly below the lower bound, the frequency should be reduced only slightly. As a result, the frequency reduction will be scaled down based on the real-time utilization and the lower bound of the optimal utilization interval.

The reason we classify the increasing frequency behavior into two types is that, when CPU utilization reaches the upper bound, we can not determine how much performance boost is needed to meet the workload. There is no such problem when the frequency decreases. For example, it is clear that the server need to reduce the frequency more when the utilization is 10% than 30%. We use two successive frequency increases as an indication that performance needs to be significantly improved. The consideration for not using three or more values is that, we believe an early entry into the fast frequency scaling phase is preferable, thus avoiding SLA violations due to low performance. Moreover, the algorithm specifies the fast scaling phase increases 4 frequency levels at once. This is a reasonable value for our experiment platform with 16 different frequency levels. This value can be changed depending on the platform characteristics and we recommend trying to be as close as possible to the root of the number of frequency levels, e.g. a platform with 32 frequency levels could set it to 6. This balances the fast search overhead with the fine-tuning search overhead, thus minimizing the overall search overhead.

B. Linux Kernel Parameter Optimization

Fig. 5 shows the multi-model prediction-based parameter optimization method for tuning Linux kernel parameters. First, this method will collect the server data in real time to evaluate the load status. If the load fluctuation is significant, it is not easy to

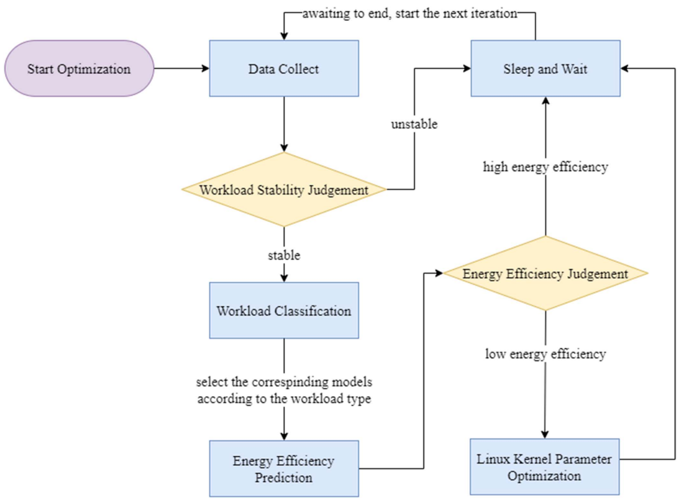
awaiting to end, start the next iteration
Start Optimization
Data Collect
Sleep and Wait
unstable
high energy efficiency
Workload Stability Judgement
stable
Energy Efficiency Judgement
Workload Classification
low energy efficiency
select the correspinding models according to the workload type
Energy Efficiency Prediction
Linux Kernel Parameter Optimization

Fig. 5. Execution process of parameter optimization method based on multi-model prediction.

accurately evaluate the status and apply an optimal combination of Linux parameters. When the load is relatively stable, the classification framework will judge the workload type carried by the server. The performance model, max performance model, and energy efficiency model are used to predict the energy efficiency data of the server. If the server has low energy efficiency, MPOD will launch the Linux kernel parameter optimization based on workload classification and energy efficiency evaluation results. After completing the parameter optimization, it will sleep and wait for a while, then repeat the above steps to achieve real-time optimization.

1) Workload Classification: To ensure the accuracy of energy efficiency prediction and the effectiveness of optimization, it is necessary to identify the type of application running on the server. Performance metrics such as server resource utilization and performance counters can be used to portray cloud application behavior and to classify cloud applications based on the distribution of these metrics [23].

We design a two-layer classification framework based on the KNN algorithm (shown in Fig. 6). The bottom layer is the KNN model for the specific application, and each KNN model stores the performance metric distribution data for the corresponding workload type. The upper layer KNN model is obtained by combining the performance metric distribution data stored in the same class of bottom KNN models. When classifying an unknown data point, each KNN model will output a distance sum between the unknown data point and the nearest K known data points. The smaller the distance sum is, the more similar the unknown data point is to the data distribution recorded in the current KNN model. Based on the distance sum, the two-layer KNN-based classification framework will first discriminate the most similar intensive type in the upper layer. Then the corresponding lower layer KNN models will discriminate which specific application type it belongs to.

The framework uses performance metrics recorded in the KNN model to evaluate workload similarity. Consequently, the selection of performance metrics dramatically affects the

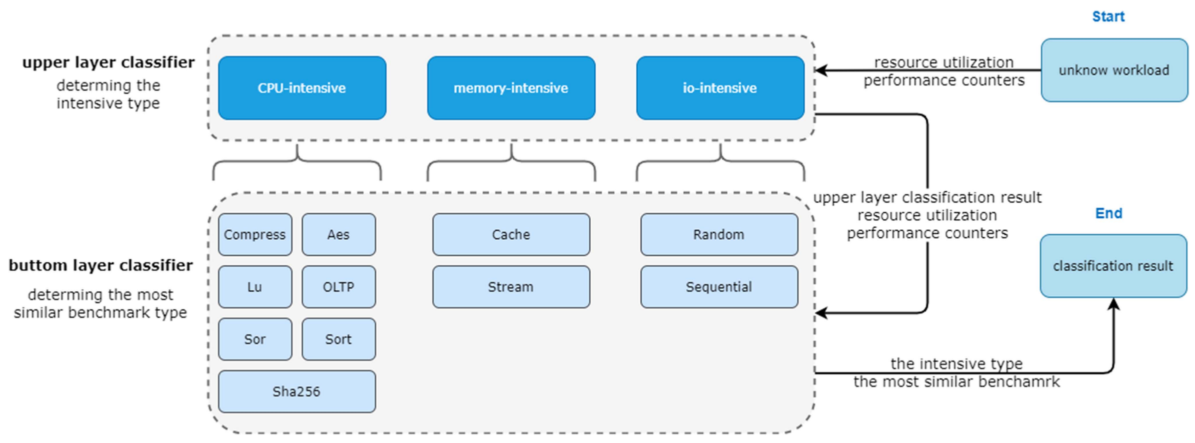

Fig. 6. Two-layer workload classification framework based on KNN.

TABLE I
PERFORMANCE METRICS USED FOR EACH LAYER OF KNN IN THE WORKLOAD CLASSIFICATION FRAMEWORK

Classifier Layer	Performance Metrics
upper layer	cpu_util, instructions, llc_loads, llc_stores, disk_util, avgrq-sz
cpu-intensive layer	instructions, cycles, llc_loads, llc_stores llc_load_misses, llc_store_misses, uops_retired.stall_cycles
memory-intensive layer	memory_usage, llc_loads, llc_stores, llc_load-misses, llc_store-misses, llc_load-misses-rate, llc_store-misses-rate, uops_retired.stall_cycles,
io-intensive layer	r/s, w/s, rMB/s, wMB/s, disk_util, avgrq-sz

accuracy of classification. The operating system can capture various performance metrics, but only a few reflect the characteristics of a given workload, while most are irrelevant. For instance, network read and write metrics are primarily worthless for CPU-intensive applications. If the KNN model contains many irrelevant features, the differences in the distance sums calculated by the significant feature metrics will be diluted. As a result, the distance sum of each KNN model will be similar, which may lead to worse classification results.

To address this problem, we use the lightgbm [24] to calculate the gain entropy of each feature in predicting energy efficiency and select the features with high gain entropy for each workload as the significant features (Table I). In creating the KNN models, the data points for each workload will only record the values of the significant features. Therefore, we can use a smaller number of features to distinguish the performance behavior of different applications, reducing the framework overhead while improving classification accuracy. Meanwhile, to prevent the huge amount of data from causing excessive overhead in KNN, the framework does not keep saving additional data online for the load behavior patterns already identified. If a new workload is encountered while running online that differs significantly from the known load behavior pattern, for example, the distance to the most similar workload is identified to exceed a certain threshold, data needs to be collected to train a new KNN model and add the model to the workload classification framework.

2) Energy Efficiency Modeling: The parameter optimization process needs to evaluate the energy efficiency of different parameter combinations, which contains parameter configurations that lead to server performance degradation. Setting these parameters on the server may reduce the application throughput or even violate SLA for cloud subscribers. Energy efficiency modeling is a common technique to solve the above problem. Building models to predict the server's performance, power consumption, and energy efficiency, we can directly output the energy efficiency data under different parameter combinations, thus avoiding improper parameter configurations from affecting the user service during optimization [25].

Due to randomization, system optimization, and other reasons, an application may produce different behavior patterns under the same parameter configuration, leading to the failure of the machine learning based black box mode in predicting the performance and energy efficiency of the server [26]. Moreover, some performance models take application parameters as input which is not available in the cloud environment. In contrast to static parameter configurations, metrics such as resource utilization and performance counters can reflect server performance in real time while protecting user privacy. Therefore we use these system-level performance metrics as input to the performance model. We assume that the application performance demand is constant over a short period. For the energy efficiency model, we can use the server performance derived from the performance model as input to eliminate the accuracy degradation caused by changes in application behavior. In addition, considering server parameter configurations as inputs allows the energy efficiency model to evaluate the energy efficiency of different combinations of parameters without actually setting the parameters on the server.

The energy efficiency model is insufficient to guarantee that the performance of the server meets the application requirement after optimization. Although the optimization goal of MPOD is to maximize energy efficiency, it is also necessary to ensure the server performance to prevent SLA violations.So we also built a max performance model to assess the maximum performance the server can achieve for specific parameter configurations.

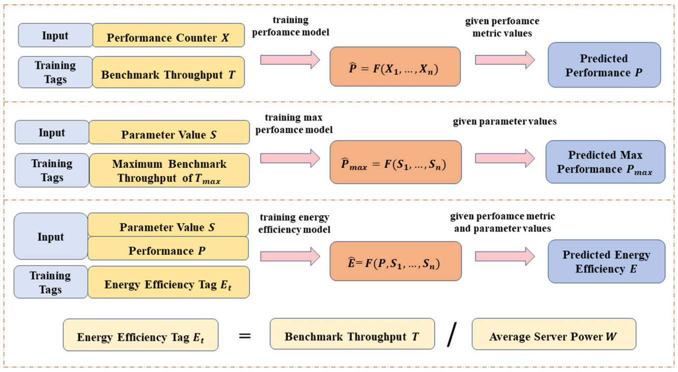
Input
Performance Counter X
Training Tags
Benchmark Throughput T
training perfoamce model
$\widehat{P} = F(X_1, \ldots, X_n)$
given perfoamce metric values
Predicted Performance P
Input
Parameter Value S
Training Tags
Maximum Benchmark Throughput of T_{max}
training max perfoamce model
$\widehat{P}_{max} = F(S_1, \ldots, S_n)$
given parameter values
Predicted Max Performance P_{max}
Input
Parameter Value S
Performance P
Training Tags
Energy Efficiency Tag E_t
training energy efficiency model
$\widehat{E} = F(P, S_1, \ldots, S_n)$
given perfoamce metric and parameter values
Predicted Energy Efficiency E
Energy Efficiency Tag E_t
=
Benchmark Throughput T
/
Average Server Power W

Fig. 7. Design of performance model, max performance model and energy efficiency model.

The load fluctuations mentioned above do not affect maximum performance with server parameter configurations. Even if the application behavior pattern changes for various reasons, the maximum performance that a server can achieve in a given configuration remains constant.

In summary, MPOD builds three prediction models to guide the optimization of Linux kernel parameters (Fig. 7). The performance model uses system-level performance metrics to assess the performance requirement of cloud applications over a short period. The max performance model uses Linux parameter values to assess the maximum performance that the server can deliver. The energy efficiency model uses predicted performance and Linux parameter values to assess the energy efficiency of the server.

3) Parameter Optimization Algorithm: The Linux kernel has many parameters with a wide range of values and interactions, resulting in the optimization function characterized by multiple peaks, non-convexity, and high dimension. The automated parameter optimization algorithm has difficulty learning the impact of different parameter combinations on server energy efficiency. In order to cope with the optimization difficulty caused by overly complex optimization functions, we decrease the complexity of optimization by reducing the parameter search space. We classify different Linux kernel parameters according to their operating principle (Table II) and select the corresponding partial parameters to optimize for different workload types. This approach allows the algorithm to learn the patterns of server energy efficiency for a different combination of parameters.

The Algorithm 2 is the pseudo-code of Linux kernel parameter optimization in MPOD. After inputting the workload type, initial performance, and energy efficiency, the algorithm will perform an iterative optimization for a period of time. Each iteration generates a set of parameter values based on the workload type and the optimization history. Then it evaluates the server's maximum performance and energy efficiency for specific parameter settings. If the maximum performance is significantly lower than the performance requirement (we allow a 5% performance degradation), the algorithm will reduce the weight of this set of parameters. The optimization record stores the results of each iteration corresponding to the workload type and the initial performance. After completing a period of iteration, if the optimal parameter configuration in the optimization record is higher than the initial energy efficiency, the optimal parameter values are set to the server. Otherwise, the original parameter configuration is maintained.

TABLE II
SIGNIFICANT LINUX KERNEL PARAMETERS CORRESPONDING TO EACH INTENSIVE WORKLOAD

Intensive Type	Parameter
CPU-intensive	kernel.sched_min_granularity_ns kernel.sched_wakeup_granularity_ns kernel.sched_migration_cost_ns kernel.numa_balancing vm.swappiness
memory-intensive	vm.dirty_ratio vm.dirty_background_ratio vm.dirty_writeback_centisecs vm.dirty_expire_centisecs vm.zone_reclaim_mode kernel.randomize_va_space vm.swappiness
IO-intensive	vm.dirty_writeback_centisecs vm.dirty_expire_centisecs block.scheduler block.read_ahead_kb block.max_sectors_kb block.queue_depth block.nr_requests

Algorithm 2: Linux Kernel Parameter Optimization Algorithm in MPOD.

```
Input: Workload type $L$, initial performance $P_{init}$, initial
       energy efficiency $E_{init}$
Output: Optimal parameter configuration $C$
1: while iterate for a certain time do
2:    generate a dictionary of parameter combination values
      $D$ according to $L$ and history record $R$
3:    use models to predict max performance $P_{max}$ and
      energy efficiency $E$ in $D$
4:    if $P_{max} < 0.95P_{init}$ then
5:       $E = E/2$
6:    end if
7:    append evaluation result $E$ and $D$ to $R$
8: end while
9: search the maximum energy efficiency $E_{max}$ in $R$
10: if $E_{max} > E_{init}$ then
11:    set the corresponding parameter configuration $C_{max}$
       on server
12: else
13:    maintain the original parameter configuration $C_{init}$
14: end if
```

IV. EXPERIMENT

The server experimented with is HUAWEI 2288H V5, equipped with two Intel(R) Xeon(R) Gold 6248 CPU @ 2.50 GHz processors and eight Samsung DDR4 2933MT/s 32 G RAMs. We first perform data collection based on the energy efficiency benchmark BenchSEE [27]. Then we determine the optimal energy efficiency utilization interval of each frequency and train the KNN classification models, performance models, max performance models, and energy efficiency models. Finally, we use BenchSEE again to verify the optimization effect of MPOD and use the energy efficiency benchmark SERT [28] and decision support benchmark TPC-H [29] to verify the generalization ability of MPOD under different workloads. Moreover, we also record the optimization overhead of MPOD.

MPOD targets physical servers rather than clusters, so it does not address job distribution, queuing, and other scheduling-related aspects. As a result, instead of considering SLA metrics such as workload latency when calculating server performance, we directly use average benchmark throughput (BenchSEE and SERT) and the time required to complete the same workload (TPC-H) for evaluation. As illustrated in Fig. 1, these metrics also reflect the response latency of the server to a certain extent, which are more comprehensive and effective evaluation metrics.

A. MPOD Implementation

In terms of data collection, we collect resource utilization, performance counters, server power consumption, and benchmark throughput while running seven CPU-intensive, two memory-intensive, and two IO-intensive workloads in BenchSEE. Each workload is run at ten different load levels ranging from 100% to 10%, and each load level runs for 60 seconds. We set different frequency and Linux kernel parameter values before the workload runs to evaluate the energy efficiency of the server under different parameter configurations. To reduce data collection overhead, we use Latin hypercube sampling to determine the values of each parameter used in BenchSEE, covering most of the feature space with as little data as possible.

In terms of model training, we determine the optimal energy efficiency utilization interval in Section III-A2 and train the KNN models in Section III-B1 based on the collected BenchSEE data. For the performance models, max performance models, and energy efficiency models in Section III-B2, we choose nine machine learning algorithms (including xgboost, linear, svr, and etc.) that are widely used in the areas of energy efficiency modeling to fit. The performance of each algorithm varies for different workload types and prediction objectives (see Section IV-B for details). So we combine the best three algorithms for each workload to establish the final performance models, max performance models, and energy efficiency models. The above process is done offline. In this way, we do not need to apply all the algorithms to predict in online usage, but only need to select the best three algorithms to use based on the results of workload classification.

In terms of optimization algorithm, we implement the logic in Algorithm 2 based on the hyperparameter optimization framework Optuna [30]. The parameter values for each iteration are decided by the CmaEs sampler in Optuna.

B. Model Accuracy

To test the accuracy of the models built in MPOD, we divide the dataset into training and test sets in the ratio of 7:3. The accuracy of different machine learning algorithms varies depending on the workload type and the prediction objective. As shown in Tables III, IV, and V, no single algorithm perform best in all scenarios. Random forest and linear work well in performance models, xgboost works well in max performance models, and all the four ensemble learning algorithms work well in energy efficiency models. Only a few specific scenarios are suitable for modeling with lasso, ridge, elasticnet, and SVR. We select the best three algorithms of each workload to generate the final performance, max performance, and energy efficiency models in MPOD.

MPOD identifies the BenchSEE workload with an average of 99.5% accuracy. It shows that variation in selected performance metrics between benchmarks is significant. In addition, we found that all the classification error samples have a load level of 10%, i.e., scenarios with extremely low application throughput. Beside load fluctuations, there are no significant numerical differences in the performance metrics for each workload at this time. As a result, it is difficult to completely distinguish load behavior patterns, which leads to wrong classification. However, such wrong classification has almost no impact on MPOD tuning. At this point, there is redundancy in server performance, and the optimization goal is reducing wasted resources. Therefore MPOD will take similar measures to reduce the idle power consumption of the server regardless of the identified workload.

On a comprehensive view, both the workload classification framework and the three prediction models can achieve at least 98% prediction accuracy, which is sufficient to provide MPOD with an accurate evaluation of real-time server status.

C. MPOD BenchSEE Optimization Effects

In this section, we will use BenchSEE to test the optimization effect of MPOD. Each workload is run for ten load levels from 100% to 10% to simulate the dynamic changes in the cloud environment, and each load level lasts for 60 seconds. Since MPOD requires iterative optimization to learn the optimal parameter configuration, we run the above workloads three times for MPOD before evaluating the optimization effect.

We compare MPOD with the other four methods:

- *Default*: server has default DVFS strategy and Linux kernel parameter values. The default configuration will be set as general values to satisfy the wide range of applications, representing the universal case rather than the worst case. Therefore, we select Default as the baseline in comparison.
- *Ondemand/Conservative*: both are DVFS policies available in the acpi_cpufreq module [31]. Since integrated into the Linux kernel, they are widely used in server optimization practice and research.
- *Efficient Policy*: a power management policy in BIOS configuration that aims to maximize the energy efficiency of the server. It includes energy efficiency optimization technologies provided by the server manufacturer. In our experiment platform, the efficiency policy consists of frequency modulation, hibernation, energy loss reduction technologies (MPC-PID), etc [32].

The effect of optimization is measured by the energy efficiency of the server shown in (7), where E_m, T_m and P_m are the energy efficiency, benchmark throughput and average power of the server using the optimization method m, m can be

TABLE III
MAE Values of Different Algorithms for Building Performance Models

Algorithm	Compress	Aes	Lu	OLTP	Sha256	Sor	Sort	Cache	Stream	Random	Sequential
xgboost	**0.001**	0.048	0.058	**0.003**	**0.005**	**0.01**	**0.002**	0.047	0.039	**0.004**	0.004
catboost	0.007	**0.042**	0.065	0.006	0.013	0.021	0.011	**0.046**	**0.036**	0.013	0.011
lightgbm	**0.001**	0.051	0.058	0.005	0.02	**0.01**	0.005	0.064	0.035	**0.004**	**0.003**
random forest	**0.001**	**0.042**	**0.04**	**0.004**	**0.004**	**0.008**	**0.003**	**0.03**	**0.018**	**0.004**	**0.003**
linear	**0.001**	**0.031**	**0.028**	**0.001**	**0.002**	**0.005**	**0.001**	**0.025**	11.506	**0.004**	**0.002**
lasso	0.203	0.204	0.189	0.226	0.15	0.268	0.263	0.193	0.277	0.274	0.221
ridge	0.002	0.030	**0.036**	0.01	0.011	0.013	0.009	0.172	**0.028**	0.01	0.026
elasticnet	0.203	0.204	0.189	0.226	0.15	0.268	0.263	0.193	0.277	0.274	0.221
svr	0.201	0.205	0.042	0.226	0.148	0.268	0.263	0.162	0.277	0.274	0.219
MPOD	**0.001**	**0.035**	**0.038**	**0.003**	**0.005**	**0.008**	**0.002**	**0.044**	**0.026**	**0.004**	**0.002**

TABLE IV
MAE Values of Different Algorithms for Building Max Performance Models

Algorithm	Compress	Aes	Lu	OLTP	Sha256	Sor	Sort	Cache	Stream	Random	Sequential
xgboost	**0.001**	**0.02**	**0.031**	**0.008**	**0.025**	**0.002**	**0.004**	**0.009**	**0.001**	0.006	**0.016**
catboost	**0.004**	0.051	0.052	0.032	0.048	**0.006**	**0.011**	0.032	**0.001**	**0.005**	**0.028**
lightgbm	0.007	0.183	0.19	0.151	0.152	0.014	0.056	0.132	0.002	0.006	0.189
random forest	**0.001**	**0.032**	0.055	**0.025**	0.04	**0.004**	**0.007**	**0.027**	**0.001**	0.006	**0.027**
linear	0.005	**0.011**	**0.007**	**0.029**	**0.02**	0.013	0.04	**0.011**	**0.001**	0.006	0.068
lasso	0.167	0.188	0.188	0.152	0.15	0.014	0.054	0.134	0.002	**0.005**	0.182
ridge	0.006	0.034	**0.042**	0.03	**0.031**	0.012	0.036	0.031	**0.001**	0.006	0.07
elasticnet	0.167	0.188	0.188	0.152	0.15	0.014	0.054	0.134	0.002	**0.005**	0.182
svr	0.166	0.188	0.191	0.15	0.149	0.028	0.08	0.14	0.002	**0.005**	0.182
MPOD	**0.002**	**0.023**	**0.038**	**0.018**	**0.022**	**0.003**	**0.007**	**0.018**	**0.001**	**0.005**	**0.02**

TABLE V
MAE Values of Different Algorithms for Building Energy Efficiency Models

Algorithm	Compress	Aes	Lu	OLTP	Sha256	Sor	Sort	Cache	Stream	Random	Sequential
xgboost	**0.004**	**0.023**	**0.021**	**0.009**	**0.015**	**0.012**	**0.012**	**0.002**	**0.01**	**0.005**	**0.006**
catboost	**0.005**	**0.029**	0.025	0.019	**0.02**	0.029	0.023	0.03	0.025	0.009	**0.008**
lightgbm	**0.005**	**0.025**	**0.023**	**0.016**	**0.021**	**0.012**	**0.016**	**0.007**	**0.014**	**0.008**	**0.006**
random forest	**0.003**	**0.029**	**0.024**	**0.016**	**0.021**	**0.013**	**0.015**	**0.004**	**0.01**	**0.006**	0.012
linear	0.043	0.058	0.055	0.051	0.044	0.046	0.046	0.035	0.039	0.027	0.022
lasso	0.229	0.235	0.229	0.244	0.222	0.242	0.246	0.269	0.241	0.25	0.204
ridge	0.044	0.059	0.059	0.05	0.053	0.048	0.046	0.034	0.04	0.028	0.022
elasticnet	0.229	0.235	0.229	0.244	0.222	0.242	0.246	0.269	0.241	0.25	0.204
svr	0.228	0.235	0.229	0.243	0.222	0.237	0.244	0.22	0.238	0.25	0.202
MPOD	**0.004**	**0.024**	**0.021**	**0.012**	**0.017**	**0.01**	**0.013**	**0.003**	**0.01**	**0.005**	**0.005**

default, ondemand, conservative, efficient policy and MPOD. The energy consumption of the server should be the average power multiplied by the time, but the same type of benchmark runs for the same amount of time, so we simply use the average power here. Large differences in the order of magnitude of throughput for different workload types (e.g. Compress maxes out at around 20000 while Stream can reach 10 million). In order to more clearly compare the energy efficiency of servers under different optimization methods, the energy efficiency of each workload is normalized by the energy efficiency value of the server under the default optimization method.

$$E_m = \frac{T_m}{P_m E_{default}} = \frac{T_m P_{default}}{P_m T_{default}} \tag{7}$$

Table VI shows the experiment result. Workloads between Compress and Sort are CPU-intensive, Stream and Cache are memory-intensive, Random and Sequential are IO-intensive.

TABLE VI
ENERGY EFFICIENCY COMPARISON OF SERVER RUNNING BENCHSEE WORKLOADS WITH DIFFERENT OPTIMIZATION METHODS

Method	Compress	Aes	Lu	OLTP	Sha256	Sor	Sort	Stream	Cache	Random	Sequential
Default	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Ondemand	1.18	1.25	0.94	0.95	1.24	0.88	0.87	0.93	1.0	0.99	0.96
Conservative	1.10	1.10	0.93	0.96	1.11	0.88	0.86	0.92	1.0	1.07	1.04
Efficient Policy	0.99	0.97	1.05	1.0	0.99	0.94	1.03	0.95	0.98	1.18	1.19
MPOD	**1.36**	**1.34**	**1.18**	**1.27**	**1.38**	**1.30**	**1.34**	**1.13**	**1.06**	**1.70**	**2.85**

As the result shows, simple optimization methods can not handle all workloads well, Ondemand, Conservative and Efficient Policy perform even less efficiently than the Default for some situations. In contrast, MPOD can achieve the best energy efficiency at all benchmarks. Moreover, different workloads have different optimization effects. The IO-intensive has the most significant optimization effect with an average of 127.5% while the memory-intensive has the least with an average of 9.5%. Compared to the best optimization method in each workload, MPOD still shows an average energy efficiency improvement of 30.5%.

D. MPOD SERT and TPC-H Optimization Effects

The variety of applications in a cloud environment makes it difficult to model all and develop corresponding optimization strategies. Our solution is that we use an existing workload classification model to identify unknown workloads. If they have similar load behavior (close performance metrics), we believe that the tuning strategy for the most similar known workload can also be applied to that unknown workload, thus reuse existing prediction models and optimization strategies.

In this section, to test the generalization capability of the MPOD, we use the models trained on BenchSEE data in Section IV-B to identify SERT and TPC-H workloads, then perform forecasting as well as optimization. Due to different implementations, SERT has a different behavior even with the same benchmark name as BenchSEE. We select seven CPU-intensive workloads (Compress-Sort), two memory-intensive workloads (Flood3, Capacity3), and two IO-intensive workloads (Random, Sequential) in SERT. Similar to Section IV-C, each workload is run for ten load levels from 100% to 10%, and each load level lasts 60 seconds. The energy efficiency of server is also evaluated by (7).

TPC-H is a benchmark closer to real application, which defines 22 query tasks and performs computational work based on the request data. Since a single TPC-H task pressurizes only a few cores, to make the experimental effect more obvious, we start one TPC-H task every 10 s, for a total of 30 starts. Different from BenchSEE and SERT, the load amount of each TPC-H task is fixed, i.e., the throughput is fixed, we directly use the energy consumption required to complete these 30 TPC-H tasks to evaluate the energy efficiency of the server. As shown in (8), where E_m, P_m and t_m are the energy efficiency, average power and total running time of the server using the optimization method m to complete the TPC-H tasks, m can be default, ondemand, conservative, efficient policy and MPOD. Similar to the BenchSEE and SERT experiments, we also use Default's energy efficiency for scaling.

$$E_m = \frac{1}{P_m t_m E_{default}} = \frac{P_{default} t_{default}}{P_m t_m} \tag{8}$$

Tables VII and VIII shows the optimization effects of Default, Ondemand, Conservative, Efficient Policy, and MPOD in SERT and TPC-H, respectively.

In SERT, Ondemand and Conservative perform better in CPU-intensive workloads, while Efficient Policy performs better in IO-intensive workloads. Memory-intensive workloads still have the most minor optimization effects. Similar to BenchSEE, MPOD can also achieve optimal energy efficiency with an average of 20.1% improvement compared to the best method of each workload, which shows the excellent generalization ability of MPOD.

For the more complex TPC-H, ondemand and conservative are apparently unable to handle it, and both appear to be negatively optimized at only 96% and 99% of the default energy efficiency, respectively. Although the efficient policy has significantly improved server performance, the power consumption has also increased to a certain extent, so the optimization effect of energy efficiency is not obvious, only a 2% improvement. In contrast, MPOD uses the opposite optimization strategy. Since our ultimate optimization objective is energy efficiency, the algorithm considers both server performance and power consumption in the process of the optimization search. From the results, there is a significant performance degradation of MPOD compared to other methods (around 18% compared to default), but the power consumption degrades more significantly (saves approximately 23% compared to default). Compared to the efficient policy, MPOD improves server energy efficiency by 10.8% which is considerably lower than the results of BenchSEE and SERT.

This amount of performance degradation does not generally result in SLA violations. As the load level decreases, the performance degradation caused by MPOD will gradually reduce. According to BenchSEE and SERT experiment result, when the load of benchmarks is 80% of the maximum performance and below, the throughput of MPOD is almost the same as other optimization policies.

E. MPOD Overhead

As mentioned before, high optimization overhead may cause server performance degradation. In this section, we record the overhead of the entire MPOD workflow. Since the overhead of frequency optimization is extremely low (occupy single CPU core less than 0.1%), we mainly evaluate the resource overhead of the Linux kernel parameter optimization process, which

TABLE VII
ENERGY EFFICIENCY COMPARISON OF SERVER RUNNING SERT WORKLOADS WITH DIFFERENT OPTIMIZATION METHODS

Method	Compress	CryptoAES	Lu	SSJ	Sha256	Sor	Sort	Flood3	Capacity3	Random	Sequential
Default	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Ondemand	1.26	1.33	1.22	0.93	1.15	1.24	1.26	1.0	1.0	0.92	0.99
Conservative	1.14	1.08	1.08	0.94	1.06	1.08	1.08	1.0	1.0	0.99	1.02
Efficient Policy	0.95	1.0	1.02	1.03	0.99	1.0	0.99	1.0	0.99	1.09	1.09
MPOD	**1.37**	**1.44**	**1.43**	**1.24**	**1.36**	**1.38**	**1.40**	**1.18**	**1.04**	**1.13**	**2.08**

TABLE VIII
RUNNING TIME, AVERAGE POWER AND ENERGY EFFICIENCY OF SERVER RUNNING TPC-H WORKLOADS WITH DIFFERENT OPTIMIZATION METHODS

Method	Time(s)	Power(W)	Energy Efficiency
Default	1269	408.6	1
Ondemand	1350	399.2	0.96
Conservative	1334	394.2	0.99
Efficient Policy	**1188**	426.0	1.02
MPOD	1497	**313.6**	**1.13**

TABLE IX
AVERAGE OVERHEAD FOR EACH STAGE OF MPOD

Stage	Time(s)	CPU(%)	Memory(MB)
Data Collection	10.02	3.1%	268.6
Prediction	0.74	59.9%	
Optimization	20.5	49.6%	
Sleep	8.7	0.5%	
Total	39.96	26.8%	

consists of four stages data collection, prediction, optimization, and sleep. We respectively specify the time of data collection, optimization, and one MPOD iteration as 10 s, 20 s, and 40 s.

Table IX shows each stage's time, CPU usage, and memory usage in one MPOD iteration. The result is generated based on the average of all BenchSEE workload optimizations. Due to the slight variation in memory usage, we only collect the average memory occupied during the MPOD execution. Although the prediction stage occupies the highest computing resource, it will complete quickly thus having little impact on server performance. Most of the overhead comes from the optimization stage. On average, MPOD occupies 26.8% of a single core and 268.6 MB of memory. Since our experimental platform has a total of 80 cores, it uses only 0.34% of the computing resource and 0.1% of the memory capacity for the entire server. Therefore, the low overhead of MPOD is sufficient for energy-efficient optimization of the server without affecting user services.

V. CONCLUSION

This paper presents an optimization method called MPOD, combining frequency and Linux kernel parameters based on the characteristics of cloud applications with high complexity and dynamic changes. We design a low overhead and fast response DVFS strategy based on CPU energy efficiency curves, a KNN-based workload classification framework, and a parameter optimization algorithm with multiple models to improve the server energy efficiency without accessing user data and violating SLA. In the experiment, we implement MPOD based on BenchSEE and demonstrate the effectiveness of the models along with optimization. We also use SERT and TPC-H to verify the generalization ability of MPOD in handling unknown workloads. Last, we demonstrate the low overhead of MPOD.

In the experiment, although MPOD provides some energy efficiency gains for various workloads, the energy efficiency optimization of MPOD decreases as the complexity of the workload increases. Further optimization of MPOD is needed to cope with more complex scenarios. Moreover, Heterogeneous CPUs (e.g. Intel Alder Lake architecture) are attracting more and more attention. Although the frequency optimization algorithm of MPOD is implemented for homogeneous CPUs, it can also be easily migrated to servers with heterogeneous CPUs if the energy efficiency curves are evaluated separately for different types of cores. Exploring the effect of applying MPOD to more servers of different architectures is also one of the topics that can be studied in the future.

REFERENCES

[1] A. Katal, S. Dahiya, and T. Choudhury, "Energy efficiency in cloud computing data centers: A survey on software technologies," *Cluster Comput.*, vol. 26, no. 3, pp. 1845–1875, 2022.
[2] A. S. Andrae and T. Edler, "On global electricity usage of communication technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.
[3] H. Cheung, S. Wang, C. Zhuang, and J. Gu, "A simplified power consumption model of information technology (IT) equipment in data centers for energy system real-time dynamic simulation," *Appl. Energy*, vol. 222, pp. 329–342, 2018.
[4] N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema, "Towards machine learning-based auto-tuning of MapReduce," in *Proc. IEEE 21st Int. Symp. Modelling, Anal. Simul. Comput. Telecommun. Syst.*, 2013, pp. 11–20.
[5] V. Perrone et al., "Amazon sagemaker automatic model tuning: Scalable black-box optimization," 2020, *arXiv: 2012.08489*.
[6] X. Fang, Y. Zou, Y. Fang, Z. Tang, H. Li, and W. Wang, "A query-level distributed database tuning system with machine learning," in *Proc. IEEE Int. Conf. Joint Cloud Comput.*, 2022, pp. 29–36.
[7] M. Malik et al., "ECoST: Energy-efficient co-locating and self-tuning MapReduce applications," in *Proc. 48th Int. Conf. Parallel Process.*, 2019, pp. 1–11.
[8] A. Fekry, L. Carata, T. Pasquier, and A. Rice, "Accelerating the configuration tuning of Big Data analytics with similarity-aware multitask Bayesian optimization," in *Proc. IEEE Int. Conf. Big Data*, 2020, pp. 266–275.
[9] G. Cheng, S. Ying, and B. Wang, "Tuning configuration of apache spark on public clouds by combining multi-objective optimization and performance prediction model," *J. Syst. Softw.*, vol. 180, 2021, Art. no. 111028.
[10] H. Menon, A. Bhatele, and T. Gamblin, "Auto-tuning parameter choices in HPC applications using Bayesian optimization," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2020, pp. 831–840.
[11] H. Zhu, D. Scheinert, L. Thamsen, K. Gontarska, and O. Kao, "Magpie: Automatically tuning static parameters for distributed file systems using deep reinforcement learning," in *Proc. IEEE Int. Conf. Cloud Eng.*, 2022, pp. 150–159.

[12] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 1009–1024.
[13] I. Sánchez Barrera, D. Black-Schaffer, M. Casas, M. Moretó, A. Stupnikova, and M. Popov, "Modeling and optimizing NUMA effects and prefetching with machine learning," in *Proc. 34th ACM Int. Conf. Supercomputing*, 2020, pp. 1–13.
[14] G. Luan, P. Pang, Q. Chen, S. Xue, Z. Song, and M. Guo, "Online thread auto-tuning for performance improvement and resource saving," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 3746–3759, Dec. 2022.
[15] X. Qi, J. Yang, Y. Zhang, and B. Xiao, "Bios-based server intelligent optimization," *Sensors*, vol. 22, no. 18, 2022, Art. no. 6730.
[16] S. K. Mishra, P. P. Parida, S. Sahoo, B. Sahoo, and S. K. Jena, "Improving energy usage in cloud computing using DVFS," in *Proc. Prog. Adv. Comput. Intell. Eng.*, 2018, pp. 623–632.
[17] A. Asghari and M. K. Sohrabi, "Combined use of coral reefs optimization and multi-agent deep Q-network for energy-aware resource provisioning in cloud data centers using DVFS technique," *Cluster Comput.*, vol. 25, no. 1, pp. 119–140, 2022.
[18] S. Lee, Y. Song, and Y. I. Eom, "PollO: Polling-aware on-demand governor for improving power efficiency," in *Proc. IEEE 14th Int. Conf. Ubiquitous Inf. Manage. Commun.*, 2020, pp. 1–4.
[19] B. Gembala, A. Yazidi, H. Haugerud, and S. Nichele, "Autonomous configuration of network parameters in operating systems using evolutionary algorithms," in *Proc. Conf. Res. Adaptive Convergent Syst.*, 2018, pp. 118–125.
[20] J. Yang, L. Chen, and J. Bai, "Redis automatic performance tuning based on EBPF," in *Proc. 14th Int. Conf. Measuring Technol. Mechatronics Automat.*, 2022, pp. 671–676.
[21] 2023. [Online]. Available: https://www.spec.org/power_ssj2008/results/power_ssj2008.html
[22] 2023. [Online]. Available: https://www.intel.com/content/www/us/en/gaming/resources/turbo-boost.html
[23] J. W. Smith and I. Sommerville, "Workload classification & software energy measurement for efficient scheduling on private cloud platforms," 2011, *arXiv:1105.2584*.
[24] G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3149–3157.
[25] A. T. Makaratzis, K. M. Giannoutakis, and D. Tzovaras, "Energy modeling in cloud simulation frameworks," *Future Gener. Comput. Syst.*, vol. 79, pp. 715–725, 2018.
[26] S. Fu, S. Gupta, R. Mittal, and S. Ratnasamy, "On the use of {ML} for blackbox system performance prediction," in *Proc. 18th USENIX Symp. Netw. Syst. Des. Implementation*, 2021, pp. 763–784.
[27] 2021. [Online]. Available: https://www.pzjdimg.com/public/file/benchsee/en/2022--05-16/BenchSEE%20Energy%20Efficiency%20Test%20Report%20Interpretation.pdf
[28] 2022. [Online]. Available: https://spec.org/sert2/SERT-designdocument.pdf
[29] 2022. [Online]. Available: https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf
[30] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 2623–2631.
[31] L. Brown, "ACPI in Linux," in *Proc. Linux Symp.*, 2005, pp. 51–68.
[32] 2023. [Online]. Available: https://e.huawei.com/kz/products/servers/accessories/fusiondirector

Weiwei Lin (Member, IEEE) received the BS and MS degrees from Nanchang University, in 2001 and 2004, respectively, and the PhD degree in computer application from the South China University of Technology, in 2007. He has been a visiting scholar with Clemson University from 2016 to 2017. Currently, he is a professor with the School of Computer Science and Engineering, South China University of Technology. His research interests include distributed systems, cloud computing, Big Data computing and AI application technologies. He has published more than 150 papers in refereed journals and conference proceedings. He has been the reviewers for many international journals, including *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Cybernetics*, *Future Generation Computer Systems*, *Information Sciences*, etc. He is a senior member of CCF.

Xiaoxuan Luo is working toward the phD degree with the School of Computer Science and Engineering, South China University of Technology. His research interests mainly focus on cloud computing and energy-efficiency optimization.

ChunKi Li is currently working toward the MS degree with the School of Computer Science and Engineering, South China University of Technology, China. His research interests include swarm intelligence algorithm, data center virtual machine scheduling.

Jiechao Liang is currently working toward the master's degree with the School of Computer Science and Engineering, South China University of Technology. His research interests mainly focus on cloud computing.

Guokai Wu received the bachelor's degree from the South China University of Technology, in 2021. He is currently working toward the master's degree with the School of Computer Science and Engineering, South China University of Technology. His research interests include cloud computing and sustainable computing.

Keqin Li (Fellow, IEEE) is a SUNY distinguished professor of computer science with the State University of New York. He is also a national distinguished professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, Big Data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored more than 850 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds more than 70 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top 5 most influential scientists in parallel and distributed computing in terms of both single-year impact and career-long impact based on a composite indicator of Scopus citation database. He has chaired many international conferences. He is currently an associate editor of the *ACM Computing Surveys* and the *CCF Transactions on High Performance Computing*. He has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is an AAIA fellow. He is also a member of Academia Europaea (Academician of the Academy of Europe).