Sparta: A Heat-Budget-based Scheduling Framework on IoT Edge Systems

Michael Zhang[1], Chandra Krintz[1], and Rich Wolski[1]

Department of Computer Science
University of California, Santa Barbara, CA 93106, USA
{lebo, ckrintz, rich}@cs.ucsb.edu

Abstract. Co-location of processing infrastructure and IoT devices at the edge is used to reduce response latency and long-haul network use for IoT applications. As a result, edge clouds for many applications (e.g. agriculture, ecology, and smart city deployments) must operate in remote, unattended, and environmentally harsh settings, introducing new challenges. One key challenge is heat exposure, which can degrade the performance, reliability, and longevity of electronics. For edge clouds, these problems are exacerbated because they increasingly perform complex workloads, such as machine learning, to affect data-driven actuation and control of devices and systems in the environment.

The goal of our work is to protect edge clouds from overheating. To enable this, we develop a heat-budget-based scheduling system, called Sparta, which leverages dynamic voltage and frequency scaling (DVFS) to adaptively control CPU temperature. Sparta takes machine learning applications, datasets, and a temperature threshold as input. It sets the initial frequency of the CPU based on historical data and then dynamically updates it, according to the applications' execution profile and ambient temperature, to safeguard edge devices. We find that for a suite of machine learning applications and deployment temperatures, Sparta is able to maintain CPU temperature below the threshold 94% of the time while facilitating improvements in execution time by 1.04x - 1.32x over competitive approaches.

Keywords: Edge Computing, Heat Budget, Scheduling System, IoT

1 Introduction

The Internet of Things (IoT) is a rapidly emerging set of technologies in which ordinary objects are equipped with digital intelligence – the ability to sense, analyze, and control their environment automatically. By linking the physical and digital worlds, IoT has the potential to enhance situational awareness and effective decision-making by humans, to detect, diagnose, and remediate problems without human intervention, to assist with personal and homeland security, to optimize manufacturing and business processes, and to automate operations throughout the economy.

To realize this impact, IoT must be embedded in the world around us – within buildings, cars, roads, homes, industrial machinery, and waterways, and distributed across farms, wild open spaces, cities, and oceans. Moreover, they increasingly leverage recent advances in data analytics, machine learning (ML), and automation *in-situ* – at the edge of the network – "near" (in terms of network latency) the locus of sensing and/or actuation. This move to the edge is the result of an increase in the velocity and volume of data and high response latencies imposed by the long-haul, intermittently available networks that connect the edge and cloud. Further, unlike in the context of e-commerce and other cloud application domains, IoT applications often benefit from spatial locality in terms of performance, robustness, and security. That is, co-location of processing infrastructure and IoT devices significantly reduces the latency between data acquisition and device actuation, enables the extension of device capability via local offloading, and alleviates the cost, power consumption, and congestion of network use versus the centralized, cloud-direct model [1].

Edge processing, however, introduces new challenges for IoT deployments. Unlike the devices themselves, edge computing elements are often designed for environments in which the ambient environmental conditions are controlled and kept within a narrow operational range. The operational settings in which these edge systems (in our work we deploy miniaturized "edge clouds" using clusters of commodity small-board computers to support IoT analytics) are deployed can be harsh, hard or costly to access, and exposed to harmful environmental elements (heat, moisture, dust, animals, other objects, humans, weather, etc.). For example, we currently support an IoT deployment for image processing and deep learning for the automatic, real-time identification of animals using camera traps deployed across UCSB Sedgwick Reserve, an ecology and wildlife educational and research reserve in California [2]. The reserve is 6,000 acres that comprise critical wildlife habitats, two watersheds at the foot of Figueroa Mountain in Santa Ynez, California, and a 300-acre farm easement. Our edge clouds fuse and analyze images from within out-buildings on the property. Sedgwick yearly outdoor temperatures range between 30° and 116° Fahrenheit (-1.1° to 46.7° Celsius); within enclosures (e.g. shelters for electrical pumping equipment where grid electricity is available) our cloud systems are subjected to much higher ambient operating temperatures.

Excessive heat can degrade the performance and reliability of devices and negatively impact their longevity (requiring more human intervention and frequent replacement). Commodity computers are particularly sensitive to high temperatures and extended exposure can cause these machines to break down, degrade in functionality, and fail prematurely – even they are protected using operational safeguards such as throttling and automatic shutdown [3]. For this reason, most manufacturers include an on-board thermal CPU temperature sensor and the ability to set a "shut-down" temperature if the CPU exceeds the manufacturer's maximum supported temperature. Figure1 shows a time series of CPU temperature in degrees Fahrenheit gathered from one of our edge clouds deployed at Sedgwick between February 2018 and June 2020. The cut-off temperature was set to 200°F (93.3°C) and the temperature drop early in the trace records the system's automatic shutdown.

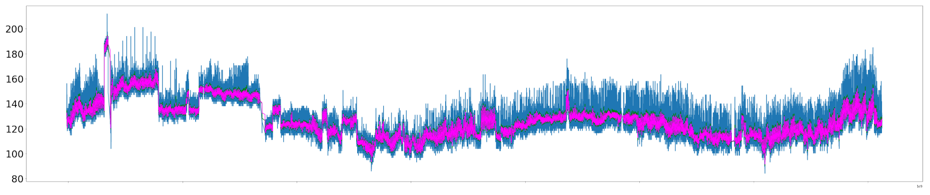

Fig. 1: The time series of CPU temperature in the edge cloud deployed at Sedgwick Natural Reserve from Feb. 28th, 2018 to Jun. 3rd, 2020. The x-axis is the epoch time and the y-axis is the CPU temperature in Fahrenheit.

In this paper, we investigate the use of dynamic voltage and frequency scaling (DVFS) [4,5] to control system temperature when the ambient temperature might cause it to exceed the acceptable operational range. DVFS is a technique that has been widely studied in the context of "power capping" – the implementation of a maximum power draw by the system. Our system – called *Sparta* – automatically exploits the relationship between system power consumption and generated heat. It does so by adjusting processor frequency dynamically so that CPU temperatures do not exceed a specified threshold as ambient temperature changes. Subject to the threshold, the system attempts to minimize the application "slow down" (relative to maximum CPU frequency) that frequency adjustments might introduce. We use Sparta to study the relationships between CPU frequency, temperature, power dissipation, and execution behavior. Moreover, we consider IoT workloads that employ a wide range of machine learning algorithms, including image recognition, natural language processing, decision forest, and time series prediction.

We consider three modes for the Sparta frequency scheduler: **Annealing**, **AIMD**, and **Hybrid**. Annealing employs an epsilon-greedy strategy to extrapolate an appropriate CPU frequency in real time. AIMD uses the linear growth of CPU frequency when temperature is under threshold and exponential reduction when it detects temperature anomalies to determine its CPU frequency. With Hybrid, we combine the best features of the two modes to overcome their drawbacks. Our results show that Sparta in Hybrid mode speeds up the execution of our applications by **1.16x** and **1.14x** on average in three thermal environments compared to Annealing and AIMD. Moreover, Sparta in Hybrid mode maintains CPU temperature below threshold **94.4%** of the time (as measured via temperature sampling), on average across all benchmarks.

In summary, with this paper, we make the following contributions:

– We investigate the relationship between CPU frequency and sampling temperature to precisely model and manage processor power dissipation during execution;

– We design and implement a heat-budget-based scheduling framework that protects edge systems from overheating and potential damage;
– We empirically evaluate the efficacy of using Sparta to control CPU temperature and accelerate machine learning applications on six real-world benchmarks in three thermal deployment environments.

In the following sections, we present the design and implementation of Sparta (Section 2). We then describe our experimental methodology and empirical evaluation of the system using multiple machine learning applications in different thermal environments (Section 3). In Section 4, we discuss related work. Finally, we present our conclusions and future work plans.

2 Sparta

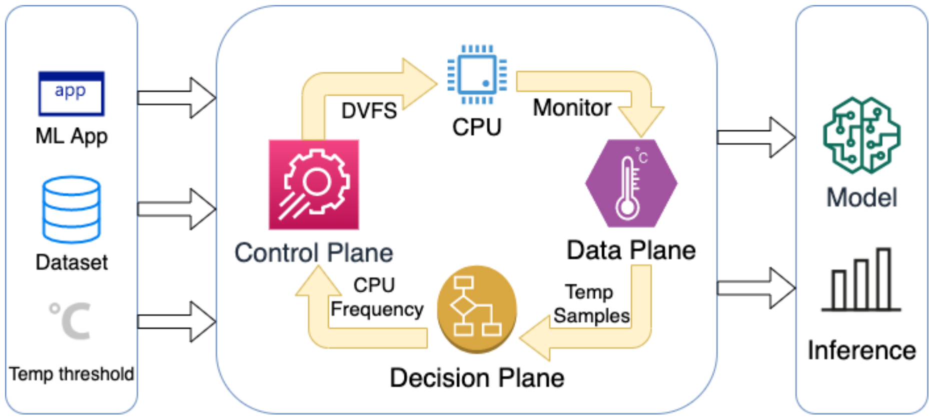

Fig. 2: The Architecture of Sparta

2.1 Architecture

To address the processor overheating challenge and accelerate the execution of applications under a CPU temperature threshold, we develop Sparta, a heat-budget-based scheduling framework for edge devices and machine learning applications. The architecture of Sparta is shown in Figure 2. The scheduler consists of three components: a control plane, a data plane, and a decision plane. Sparta takes a machine learning application, datasets, and a CPU temperature threshold as input. During the execution, the scheduler utilizes a feedback control mechanism that controls the CPU temperature by dynamically adjusting CPU frequency via system-level dynamic voltage and frequency scaling (DVFS).

Sparta returns the trained model and inference results at the end of the execution.

The data plane monitors, samples, and records the CPU real-time temperature via the lm-sensors interface [6] and selects the maximum temperature within a sliding time window. Both the sampling rate and window size are configurable. (1/second and 5 seconds by default) To signify the authentic temperature of multi-core processors, data plane records the temperature samples of the entire CPU package instead of any specific ones. Being accessible by decision plane, all structured temperature data helps determine the proper CPU frequency in real time to keep the CPU temperature under threshold.

The control plane manages the CPU power and temperature. In the design phase, we consider two methods: Sleep injection and DVFS. The first method injects sleep time in the iteration loop that lowers the CPU usage, whereas the second method adjusts the CPU frequency by tuning the CPU voltage. We experiment with these two methods on a multi-threaded matrix multiplication benchmark and monitor the CPU temperature. Figure 3 shows the CPU temperature time series using these two methods. We observe the latter method generates a controllable and stable temperature curve, and thus choose DVFS as the control plane interface. Upon the execution of scheduler, control plane receives the determined CPU frequency and sets the max clock speed of all cores in the CPU package on-the-fly. This way the control plane effectively manages the power consumption and heat generation of the processor.

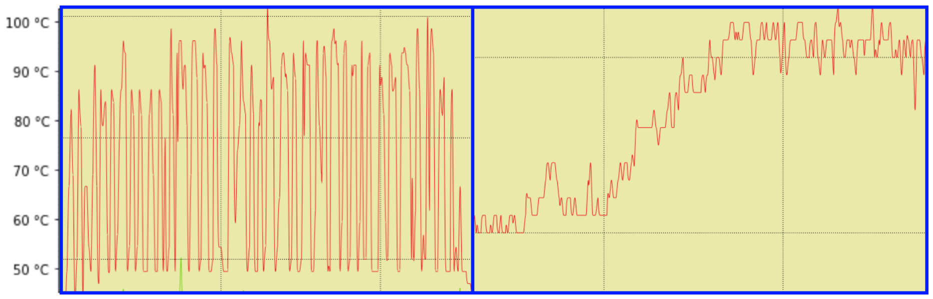

Fig. 3: The CPU temperature time series by sleep injection (left) vs DVFS (right). The x-axis is the time frame and the y-axis is the CPU temperature ranging from 48 °C to 100 °C.

The decision plane determines the CPU frequency based on historical and real-time temperature data throughout the execution. To provide the historical dataset, on which decision plane decides the initial CPU frequency, we collect CPU temperature and frequency data from a multi-threaded matrix multiplication (MATMUL) benchmark that simulates the underlying operations in machine learning applications.

We gather the data in the ambient temperature ranging from 2.6 °C to 43.8 °C to cover different thermal environments. In the experiment, we found the sequence of CPU frequency and maximum temperature in a time window demonstrate a better linear relationship than the sequence of all temperature, because of its inherent oscillating feature. To verify the correlation between MATMUL and machine learning applications, we collect the same data from an image recognition application written in Tensorflow [7]. As depicted in Figure 4, we found the correlated linear relationship between the CPU frequency and logarithmic delta temperature defined as $log(T_{max} - T_i)$, where T_{max} is the maximum temperature sample in the time window and T_i is the starting CPU temperature in idle state.

Depending on this correlation, decision plane extrapolates the appropriate CPU frequency by linear regression from the MATMUL dataset and assigns initial CPU frequency before the execution starts. During the process, decision plane starts to extrapolate CPU frequency from real-time data that accurately reflects the ambient temperature and the execution pattern of ML applications. The extrapolation frequency is 12/minute by default and configurable by users.

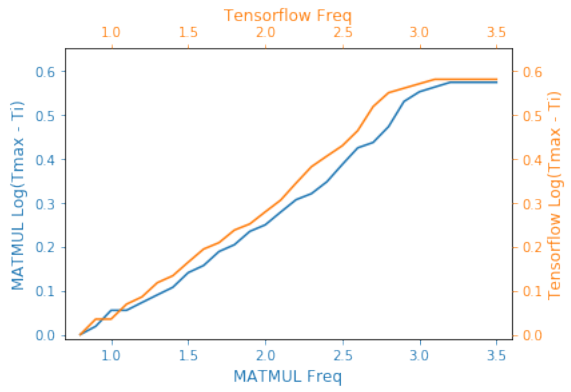

Fig. 4: The linear relationship between CPU frequency and logarithmic delta temperature of two benchmarks. The blue curve represents MATMUL and the orange curve represents the image recognition application. The plateaus at the right side of curves are caused by CPU hardware temperature throttling.

2.2 Operating Modes

In the testing phase of Sparta, we identified two major problems in the decision plane. First, the extrapolation from linear regression oftentimes gets stuck at a local minimum. Thus, the determined CPU frequency is frequently lower than the ideal one, which leaves computational resources idle during execution. Second, the response time to correct the CPU from overheating is longer than expected when CPU temperature surpasses the threshold. To solve these two problems, we construct three operating modes for Sparta: Annealing, AIMD, and Hybrid.

Annealing is a probabilistic algorithm that leverages the epsilon-greedy strategy that balances exploration and exploitation by choosing randomly. In this mode, Sparta scheduler picks a value(P) in the range [0, 1] uniformly at random and compares it with ϵ/K, where ϵ is a probability of taking random actions (0.5 by default) and K is the number of extrapolation decision plane has made. The scheduler assigns a random CPU frequency when P is greater, whereas it keeps the extrapolated frequency when P is less than ϵ/K. With a decreasing probability of ϵ/K as the application proceeds, scheduler stabilizes and chooses to exploit what it has learned so far. When the ambient temperature or the execution pattern shifts dramatically, the scheduler resets the ϵ/K that allows more random exploration. This mode effectively addresses the problem of CPU frequency stuck at a local minimum and expedites the execution of machine learning applications under temperature threshold.

AIMD is a feedback control mechanism that responds to CPU temperature anomaly faster. The scheduler configures the CPU frequency according to the historical data extrapolation at the start of execution. During the execution, it decreases the CPU frequency by a multiplicative factor (0.5 by default) when CPU temperature surpasses the threshold. Subsequently, it increases the frequency by a fixed amount (0.07 GHz by default) every iteration until the CPU temperature stabilizes right below the threshold. The decision plane turns into hibernation at this point to prevent redundant tuning on CPU frequency that leads to inefficient execution. Meanwhile, the data plane keeps monitoring the CPU temperature and wakes up decision plane if any anomalies caused by ambient temperature or execution pattern are detected. AIMD reduces the response time to temperature deviation and keeps most samples under the threshold.

Hybrid combines Annealing and AIMD modes to address each other's disadvantage: if the probabilistic actions in Annealing drive CPU temperature above threshold, AIMD brings the anomaly back to normal fast; when AIMD settles at a local minimum of CPU frequency and leaves resources idle, Annealing boosts the execution by assigning a random CPU frequency. This way, Hybrid mode provides a complement to accelerate the machine learning execution while keeping the CPU temperature under threshold.

3 Evaluation

Based on the fact that most resource-demanding programs on edge cloud in our Sedgwick Natural Reserve settings are machine learning applications, in this section, we empirically evaluate Sparta's performance in a series of experiments on six benchmarks, ranging from image recognition, natural language processing to random forest and time series prediction, which are implemented based on Tensorflow and executed through Sparta's actuator interface. We first overview the machine learning benchmarks that we consider and our experimental methodology. We then present our results.

3.1 Machine Learning Benchmarks

To comprehensively evaluate the efficacy and efficiency of Sparta, we implemented 6 machine learning benchmarks, which consist of four categories: image recognition, natural language processing, ensemble learning and time series analysis. We aim to test Sparta on a variety of machine learning applications that represent different execution patterns.

WTB_Train is an image recognition application that we use as a benchmark to train a convolutional neural network (CNN) [8] based on ResNet50 [9]. The training dataset contains animal images from a wildlife monitoring system called "Where's The Bear" (WTB) [10]. "Where's The Bear" is an end-to-end distributed data acquisition and analytical system that automatically analyzes camera trap images collected by cameras sited at the Sedgwick Natural Reserve [2] in Santa Barbara County, California. In total, there are five classes that we consider: Bear, Coyote, Deer, Bird, and Empty, by which we label images for training tasks. We also up-sampled minority classes using the Keras Image Data Generator [11], since the class size is unbalanced due to the frequency of animal occurrences. Doing so ensures that the classification model is not biased. We resized every image in the dataset to 1920×1080, and for each class, the dataset contains 60 images used to train the CNN model. Once the training is complete, the application stores this model in hdf5 format in object storage.

The WTB_Train application has a cold start at the beginning of the execution since it loads a pre-trained neural network model and training datasets. Once it completes loading, the entire training process has relatively consistent CPU usage and temperature.

WTB_Inf inferences the type of wildlife in camera trap pictures based on the model trained by WTB_Train. It loads the trained hdf5 model at the beginning and, for each picture, it assigns probabilities to five classes we consider in the training dataset by Softmax function. In each experiment, we assign 20 pictures for WTB_Inf to inference. In terms of the execution pattern, WTB_Inf runs in short bursts as opposed to WTB_Train. Therefore, the CPU usage and temperature fluctuate dramatically throughout the execution of this benchmark.

MNIST is a dataset containing grayscale pictures of handwritten digits, in which it has 60,000 examples as the training set and 10,000 examples as the testing set. Based on the dataset, we train a 2-layer convolutional neural network [12] and test its accuracy in the third application. In contrast to WTB benchmarks, the size of pictures is smaller (28×28) and the model is simplified in MNIST.

BiLSTM is a sentiment analysis application based on a dataset of the Internet Movie Database (IMDB) movie reviews. It consists of 25,000 sequences each for training and testing. The model is constructed as a bidirectional LSTM with a classification layer using the sigmoid activation function. We train the model by the the training dataset and validate its performance in classifying sentiment by the testing dataset. Since it has a large dataset and a complex model, the execution pattern is long-running and consistent in CPU usage and temperature.

Decision_Forest is an implementation of deep neural decision forests [13] that classifies high-earning individuals from the pool. The benchmark leverages the United States Census Income Dataset [14] that has 48,843 instances with 14 features, including age, education, occupation, etc. The dataset is split up that the training part has 32,561 instances and the testing part has 16,282 instances. The application has three phases: it firstly processes the dataset by encoding input features. Then, it trains a deep neural decision tree model. Based on that, the application trains a neural decision forest model consists of a set of neural decision trees. Therefore, the usage and temperature of CPU increasingly grow throughout the process.

Time_Series is a time series prediction application built on the climate data recorded by the Max Planck Institute for Biogeochemistry [15]. The dataset has 14 features such as temperature, pressure, humidity, etc. and the sampling frequency is 10 minutes. The time frame of the dataset ranges from Jan. 10th, 2009 to Dec. 31st, 2016. The application uses 300,693 rows to train a single-layer LSTM model, by which we can predict outdoor temperature in next 72 timestamps (12 hours) given the samples in the past 720 timestamps (120 hours). By this benchmark, we intend to evaluate Sparta on an application with a lightweight model and a large dataset.

3.2 Experimental Setup

Each edge cloud node used in the experiments is an Intel NUC [16] (6i7KYK) with two Intel Core i7-6770HQ 4-core processors (6M Cache, 2.60 GHz) and 32GB of DDR4-2133+ RAM connected via two channels. We use dynamic voltage and frequency scaling (DVFS) to control the frequency of CPU from 0.8GHz to 3.5GHz.

To simulate the natural temperature in Sedgwick natural reserve, we create three thermal environments in an isolated cooler that represent cold, neutral, and hot ambient temperature. In the cold scenario, the ambient temperature is 2.6°C and the CPU of NUC runs under 40°C in idle status. In the neutral scenario, the CPU of NUC starts at 51°C under the ambient temperature of

23.9°C. The hot scenario increases the ambient and CPU temperature to 43.8°C and 68°C respectively.

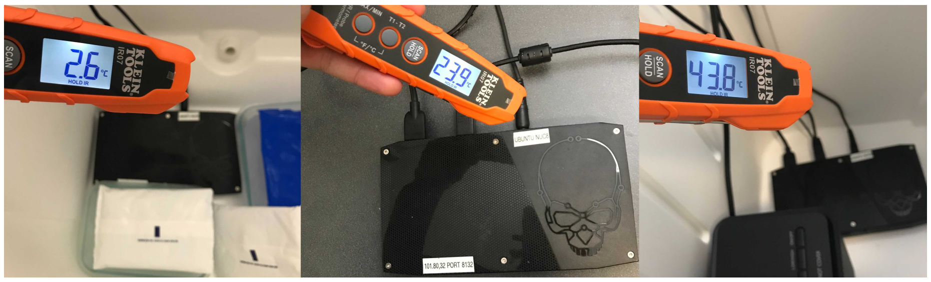

Fig. 5: Three thermal environments in the experiment

There are two goals of the Sparta scheduler: the first is to limit the CPU temperature under the threshold; the second is to accelerate tasks without overheating the edge cloud. To evaluate these two objectives, we execute 6 machine learning benchmarks under 3 modes of Sparta scheduler. In each experiment, Sparta takes inputs of task program, corresponding workload dataset and a threshold temperature. To keep the comparison consistent across 3 thermal environments, we use 75°C as the threshold temperature for all experiments.

In 3 modes of Sparta, we execute each machine learning benchmark repeatedly 100 times under 3 thermal environments (totally $3 \times 3 \times 6 \times 100 = 5400$ executions) and report relevant metrics, both mean and standard deviation, to compare the efficacy and efficiency among Annealing, AIMD, and Hybrid modes.

3.3 Application Efficacy

We first measure the stabilization time for six benchmarks. We define stabilization time as the elapsed time from the start to the point all CPU temperatures in the sampling time window are within $[T_s - T_d, T_s]$, where T_s is the threshold and T_d is a slack variable (3 °C by default). During each of the 10 consecutive executions (1 epoch) of benchmarks, we record the duration when the Sparta scheduler stabilizes the CPU temperature according to the threshold. As shown in the first part of Table 1, we report the mean and stdev of stabilization time for each benchmark in 3 modes. Hybrid mode uses less time to stabilize CPU temperature than Annealing and AIMD in all six benchmarks. It performs even better in WTB_Inf benchmark that has a short burst execution pattern and volatile CPU temperature.

As the second part of Table 1 presents the result in the thermal dimension, Hybrid mode also uses less time to stabilize CPU temperature across all 3 thermal environments, comparing to Annealing and AIMD. Averagely, Hybrid mode uses 43.61 seconds in the stabilization phase, in contrast to 61.16 seconds in

Table 1: The mean and stdev of **stabilization time** in seconds for 6 machine learning benchmarks in 3 Sparta modes. Compared to Annealing and AIMD, Hybrid mode uses less time to stabilize CPU temperature across all benchmarks and all thermal scenarios.

	WTB Train	**WTB Inf**	**MNIST**	**BiLSTM**	**Decision Forest**	**Time Series**
Annealing	53.79 (30.1)	50.7 (21.1)	62.02 (32.2)	69.02 (33.8)	59.13 (31.9)	72.31 (28.8)
AIMD	61.73 (24.9)	63.26 (25.0)	58.91 (14.0)	59.9 (11.2)	78.17 (30.7)	73.41 (28.9)
Hybrid	38.59 (26.11)	23.24 (17.33)	38.66 (22.17)	52.83 (20.6)	55.46 (25.5)	52.91 (29.8)

	Neutral	**Cold**	**Hot**	**Average**
Annealing	68.15	67.27	48.07	61.16
AIMD	63.10	77.40	57.20	65.90
Hybrid	43.03	52.43	35.39	43.61

Annealing and 65.9 seconds in AIMD. Hybrid mode's performance is even better in the hot scenario, which is the key use case for edge devices to prevent overheating in Sedgwick natural reserve.

We next empirically evaluate the execution time of six benchmarks by the Sparta scheduler. In the first part of Table 2, we report the mean and stdev of execution time for each benchmark under 3 modes. On average, the Hybrid mode completes the task of each benchmark faster than Annealing and AIMD. Given the stdev and degree of freedom, we also run student t-test among 3 modes for each benchmark and confirm that the execution time by Hybrid is smaller than Annealing and AIMD with a statistical significance level of 5%. Table 2 also indicates the speedup of Hybrid over Annealing and AIMD, ranging from 1.04x to 1.32x.

The second part of Table 2 demonstrates the average execution time in 3 thermal scenarios. On average, Hybrid mode completes the task in 126.61 seconds, in comparison with 146.09 seconds by Annealing and 147.46 seconds by AIMD. Hybrid mode provides 1.16x and 1.14x speedup respectively over Annealing and AIMD. These results show that Sparta in Hybrid mode efficiently executes more workloads than Annealing and AIMD mode under the same temperature threshold.

To investigate the error from the sampling temperature and threshold, we next evaluate the Root Mean Square Error (RMSE) of all temperature samples in the executions. We define $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(T_i - \hat{T})^2}$, where T_i is a sample of CPU temperature, $\hat{T}$ is the temperature threshold and n is the number of temperature samples. In Table 3, we display the mean and stdev of RMSE of all CPU temperature samples. The RMSE of Hybrid mode is the least across all six benchmarks among the other two modes. Hybrid mode also has the lowest RMSE in all three thermal scenarios. On average, Hybrid has 6.34 as RMSE for all temperature samples from the threshold.

Table 2: The mean and stdev of **execution time** in seconds for 6 machine learning benchmarks in 3 modes of Sparta. Compared to Annealing and AIMD, Hybrid mode uses less time to complete tasks across all benchmarks and all thermal scenarios.

	WTB Train	WTB Inf	MNIST	BiLSTM	Decision Forest	Time Series
Annealing	374.67 (9.8)	60.94 (3.9)	39.85 (2.9)	222.34 (3.5)	48.21 (3.6)	130.65 (7.6)
Speedup	1.17x	1.10x	1.21x	1.04x	1.15x	1.32x
AIMD	393.55 (5.8)	64.32 (3.9)	36.51 (4.3)	234.92 (5.2)	45.38 (2.2)	110.06 (6.2)
Speedup	1.22x	1.16x	1.13x	1.10x	1.09x	1.11x
Hybrid	318.55 (4.2)	55.31 (3.2)	32.53 (2.3)	212.91 (2.6)	41.56 (4.4)	98.78 (7.2)

	Neutral	Cold	Hot	Average
Annealing	145.72	116.14	176.42	146.09
Speedup	1.17x	1.06x	1.26x	1.16x
AIMD	134.67	122.28	185.42	147.46
Speedup	1.07x	1.15x	1.18x	1.14x
Hybrid	124.86	107.68	147.28	126.61

Lastly, we report the percentage of samples below threshold temperature in six benchmarks. The first part of Table 4 manifests the mean and stdev of PTBT (Percentage of Temperature Below Threshold) for six benchmarks. Because Annealing mode uses a probabilistic algorithm, it results in the lowest PTBT metric among three modes. Since AIMD mode multiplicatively decreases the CPU frequency whenever a temperature over the threshold is detected, it has the highest PTBT metrics in all six benchmarks. Combined with Annealing and AIMD, the PTBT of Hybrid mode is between the other two modes. This relationship holds for all three thermal scenarios, as depicted in the second part of Table 4. Hybrid mode maintains 94.4% of all temperature samples below the threshold. Thus, we consider the above results strong proofs of Sparta's efficacy in preventing overheating of edge devices and executing a variety of tasks as efficiently as possible.

4 Related Work

As related work, we consider recent advances in edge cloud's energy consumption and power management. [17] proposes computational sprinting which is a class of mechanisms that supplies additional power on processors for short duration to improve performance. It also introduces phase change materials onto processors to absorb additional heat primarily concerning the performance. Thrifty-Edge [18] presents a resource-efficient edge computing paradigm that consists of an offloading mechanism based on delay-aware task graph partition and a virtual machine selection method. To augment existing resources, [19] manifests a dynamic fog computing framework that schedules computing tasks to Citizen Fog (CF) with the highest computational ability. Different from the above systems,

Table 3: The mean and stdev of **RMSE** of all temperature samples for 6 benchmarks in 3 modes of Sparta. Compared to Annealing and AIMD, Hybrid mode has less RSME to threshold temperature across all benchmarks and all thermal scenarios.

	WTB Train	WTB Inf	MNIST	BiLSTM	Decision Forest	Time Series
Annealing	5.04 (1.0)	7.88 (1.7)	9.22 (0.8)	5.07 (1.3)	9.91 (1.5)	9.63 (2.4)
AIMD	4.39 (0.6)	6.24 (0.9)	8.35 (1.0)	5.81 (2.1)	9.95 (1.6)	8.67 (3.1)
Hybrid	4.32 (0.6)	5.79 (1.2)	6.11 (1.8)	4.90 (1.2)	9.48 (2.4)	7.25 (3.0)

	Neutral	Cold	Hot	Average
Annealing	7.12	9.59	6.67	7.79
AIMD	5.69	9.39	5.79	6.96
Hybrid	4.92	9.10	4.99	6.34

Table 4: The mean and stdev of **PTBT** (Percentage of Temperature Below Threshold) for 6 benchmarks in 3 modes of Sparta. Due to their inherent algorithm, Annealing has the lowest PTBT value and AIMD has the highest, whereas the Hybrid mode has the PTBT value in-between across all benchmarks and all thermal scenarios.

	WTB Train	WTB Inf	MNIST	BiLSTM	Decision Forest	Time Series
Annealing	71.8% (0.05)	83.0% (0.14)	83.4% (0.09)	72.6% (0.10)	84.3% (0.07)	91.0% (0.13)
AIMD	97.2% (0.07)	99.7% (0.01)	98.0% (0.08)	99.6% (0.09)	98.7% (0.04)	99.5% (0.25)
Hybrid	93.0% (0.11)	95.2% (0.14)	92.7% (0.07)	97.2% (0.23)	91.1% (0.10)	96.9% (0.17)

	Neutral	Cold	Hot	Average
Annealing	88.3%	79.7%	75.1%	81.1%
AIMD	99.7%	98.6%	98.2%	98.8%
Hybrid	98.1%	92.29%	92.7%	94.4%

Sparta focuses on preventing CPU overheating caused by ambient temperature and program execution patterns on edge cloud deployed in natural conditions.

By offering distributed, reliable, and low-latency machine learning services, edge-based ML as a fast-growing area has a great appeal both for AI and system research community. Thus, we also consider the cutting-edge development in machine learning based on edge cloud. [20] explores the building blocks and principles of wireless intelligence at edge networks concerning latency reduction, reliability guarantees, scalability enhancement, and privacy constraints. [21] provides a comprehensive survey of techniques in the scope of machine learning system at the network edge, including distributed training and inference, real-time video analytics and speech recognition, autonomous vehicles and smart cities, etc. [22] presents an approach to estimate the performance of ML application on edge cloud and to load appropriate computing resources for an edge-based application. The above work provide guiding principles and examples for Sparta and serve as one of the key motivations for our work.

5 Conclusion

In this paper, we propose a heat budget-based scheduling framework, called Sparta, aiming to prevent edge cloud CPU overheating in executing machine learning applications. Sparta's scheduler integrates three components – data plane, decision plane, and control plane: Decision plane extrapolates the initial CPU frequency from historical benchmark data and dynamically adjusts it based on real-time data monitored by data plane, while control plane modified the CPU frequency via DVFS throughout the execution. Sparta strives to accelerate the execution of applications without sacrificing the CPU overheating protection.

We present the design principles and implementation details of Sparta's components and operating modes that address the drawback we encounter in the testing phase. Our empirical evaluation demonstrates Sparta effectively protects CPU from overheating, putting **94.4%** temperature samples under the threshold in Hybrid mode. In the meantime, it speeds six benchmarks' execution up to **1.04x - 1.32x** in all three thermal environments compared to Annealing and AIMD.

As part of future work, we plan to investigate using non-uniform distributions in generating random values for exploration in Annealing mode that potentially improves the PTBT metrics. We also plan to extend the deployment of Sparta at edge cloud clusters and investigate its performance in the distributed execution of training and inference process.

Acknowledgments This work has been supported in part by NSF (CNS-2107101, CNS-1703560, CCF-1539586, ACI-1541215), ONR NEEC (N00174-16-C-0020), and the AWS Cloud Credits for Research program. This work was performed in part at the University of California Natural Reserve System Sedgwick Reserve DOI: 10.21973 / N3C08R.

References

1. N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob and M. Imran, "The Role of Edge Computing in Internet of Things," in IEEE Communications Magazine, vol. 56, no. 11, pp. 110-115, November 2018, doi: 10.1109/MCOM.2018.1700906.
2. Sedgwick Natural Reserve Homepage `https://sedgwick.nrs.ucsb.edu` Last accessed 30 Apr 2021
3. `https://www.intel.com/content/www/us/en/support/articles/000005597/processors.html` Last accessed 30 Apr 2021
4. Liu, Yongpan, Huazhong Yang, Robert P. Dick, Hui Wang, and Li Shang. "Thermal vs energy optimization for dvfs-enabled processors in embedded systems." In 8th International Symposium on Quality Electronic Design (ISQED'07), pp. 204-209. IEEE, 2007.

5. Wang, Lizhe, Gregor Von Laszewski, Jay Dayal, and Fugang Wang. "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS." In 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 368-377. IEEE, 2010.
6. `https://github.com/lm-sensors/lm-sensors` Last accessed 30 Apr 2021
7. `https://www.tensorflow.org/` Last accessed 30 Apr 2021
8. Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks. MIT Press, Cambridge, MA, USA, pp. 255–258, 1998.
9. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.
10. A. R. Elias, N. Golubovic, C. Krintz and R. Wolski, "Where's the Bear? - Automating Wildlife Image Processing Using IoT and Edge Cloud Systems," 2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI), pp. 247-258, 2017.
11. Keras Image Data Generator `https://keras.io/preprocessing/image/#imagedatagenerator-class` Last accessed 30 Apr 2021
12. Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998
13. P. Kontschieder, M. Fiterau, A. Criminisi and S. R. Bulò, "Deep Neural Decision Forests," 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1467-1475, doi: 10.1109/ICCV.2015.172.
14. `https://archive.ics.uci.edu/ml/datasets/census+income` Last accessed 30 Apr 2021
15. `https://www.bgc-jena.mpg.de/wetter/` Last accessed 30 Apr 2021
16. `https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html` Last accessed 30 Apr 2021
17. Seyed Majid Zahedi, Songchun Fan, Matthew Faw, Elijah Cole, and Benjamin C. Lee. 2017. Computational Sprinting: Architecture, Dynamics, and Strategies. ACM Trans. Comput. Syst. 34, 4, Article 12 (January 2017), 26 pages. DOI:https://doi.org/10.1145/3014428
18. X. Chen, Q. Shi, L. Yang and J. Xu, "ThriftyEdge: Resource-Efficient Edge Computing for Intelligent IoT Applications," in IEEE Network, vol. 32, no. 1, pp. 61-65, Jan.-Feb. 2018, doi: 10.1109/MNET.2018.1700145.
19. Md Razon Hossain, Md Whaiduzzaman, Alistair Barros, Shelia Rahman Tuly, Md. Julkar Nayeen Mahi, Shanto Roy, Colin Fidge, Rajkumar Buyya, A scheduling-based dynamic fog computing framework for augmenting resource utilization, Simulation Modelling Practice and Theory, Volume 111, 2021, 102336, ISSN 1569-190X, https://doi.org/10.1016/j.simpat.2021.102336.
20. Park, Jihong & Samarakoon, Sumudu & Bennis, Mehdi & Debbah, mérouane. (2019). Wireless Network Intelligence at the Edge. Proceedings of the IEEE. 107. 10.1109/JPROC.2019.2941458.
21. Murshed, M. G. Sarwar & Murphy, Christopher & Hou, Daqing & Khan, Nazar & Ananthanarayanan, Ganesh & Hussain, Faraz. (2019). Machine Learning at the Network Edge: A Survey.
22. B. D. Cruz, A. K. Paul, Z. Song and E. Tilevich, "Stargazer: A Deep Learning Approach for Estimating the Performance of Edge- Based Clustering Applications," 2020 IEEE International Conference on Smart Data Services (SMDS), 2020, pp. 9-17, doi: 10.1109/SMDS49396.2020.00009.