Carbon Aware Serverless System

Jaykumar Patel
patel.jay4802@utexas.edu
jnp2369

Afnan Mir
afnanmir@utexas.edu

Nidhi Dubagunta
nidhi.dubagunta@utexas.edu

Abstract

Serverless computing is a method that enables developers to build and deploy applications without having to provision resources or manage infrastructure. However, existing serverless systems do not consider the carbon footprint of function invocations when allocating or configuring resources, such as memory and vCPU. In this paper, we propose a carbon-aware serverless system that focuses on minimizing the carbon emissions of function invocations while continuing to meet service level objectives (SLOs) by finding an optimal configuration for vCPU count and operational frequency of the vCPU cores. We first perform a measurement study by running a variety of workloads with different resource configurations to understand the relationship between resource usage, energy consumption, and performance. We also collect measurements for energy consumption during different stages in the serverless function invocation lifecycle. Leveraging these insights, we aim to incorporate a Bayesian Optimization algorithm into Openwhisk, an open-source and serverless cloud platform, to build a serverless system that performs carbon-aware dynamic frequency scaling (DFS) and resource allocation.

1 Introduction

2 Motivation

3 Related Work

4 Measuring Carbon Emission

Before we conduct the measurement study, we need to define how to measure carbon emissions. According to Gupta et al. [1], the carbon footprint of a server can be calculated using the following formula where OP_{CF} is the operational carbon footprint, CI_{use} is the carbon intensity of the energy source, and $Energy$ is the operational energy:

$$OP_{CF} = CI_{use} \times Energy \tag{1}$$

The carbon intensity refers to how *clean* the energy source is. For example, coal has a higher carbon intensity than solar. We assume that the energy source of the serverless system is constant, and therefore, the carbon intensity is constant. We can then use the operational energy as a proxy for the carbon footprint.

4.1 Measuring Energy

We first need a way to measure the energy consumption of a serverless function invocation. EnergAt [2] is a tool that measures fine-grained energy consumption at the thread-level while taking into account NUMA effects. An EnergAt process continuously samples a target container, where each sample returns energy readings over an interval of at least 50ms. Therefore, EnergAt will get continuous back-to-back samples of energy readings over time. The energy consumption of a target container is the sum of the energy readings over the duration of the function invocation. One limitation of EnergAt is that every target container for which we want to measure energy consumption must be instrumented with its own EnergAt process. This is not feasible as the number of EnergAt processes and resource contention scales with the number of target containers.

To mitigate this issue, we created EnergyLiteDaemon. EnergyLiteDaemon is a daemon that samples multiple target containers in a round-robin fashion, where each sample returns energy readings over an interval of at least 50ms. This allows us to measure energy consumption of multiple target containers with a single, light-weight, process. Due to the round-robin fashion, however, the sampling frequency per target container decreases as a function of the number of target containers. If there are multiple containers, the sample are no longer back-to-back – there may be some time delay between two samples. To solve this, we complete the following steps to interpolate the total energy consumption of a target container:

Variable	Values
Function Type	Float Matrix Multiplication, Image Processing, Video Processing, Encryption, Linpack
vCPU Count	1, 2, 3, 5, 7, 10, 13, 16, 19, 22, 25, 28, 31
vCPU Frequency (GHz)	1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2. 2.4

Table 1: Measurement Study Configurations – Energy Variation with vCPU and Frequency Allocation

Power Calculation: We compute the average power for every sample by dividing the energy consumed over the interval of the sample by the duration of that interval. This allows us to get a power-time which contains n points, where n is the number of samples.

Trapezoidal Integration: Given a power-time scatter-plot, we compute the total energy consumption of a target container by performing trapezoidal integration. Essentially, all of the points in the power-time scatter-plot are connected by straight lines, and the area under the curve is calculated. This area is the total energy consumption of the target container.

A toy example of energy measurement using EnergAt and EnergyLiteDaemon is shown in Appendix A.1.

Due to EnergyLiteDaemon's round-robin sampling approach, which reduces its overhead, it may not capture all energy readings for a container. Even with trapezoidal integration, the energy consumption will not be exact. To determine the accuracy of EnergyLiteDaemon, we ran 20 stress docker containers, with varying CPU usage and duration. Figure 1 shows the energy consumption of the containers as measured by EnergAt and EnergyLiteDaemon. The absolute percentage error of EnergyLiteDaemon is less than 15%, which is acceptable for our purposes.

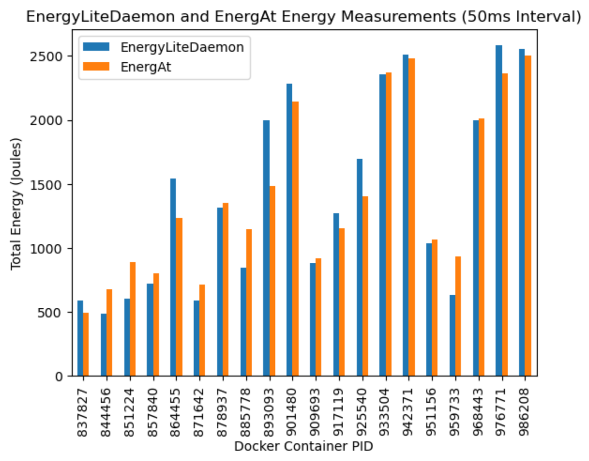


Figure 1: Energy measurements of EnergAt and EnergyLiteDaemon for 20 stress docker containers.

5 Measurment Study

We conduct two measurement studies to answer the following questions: 1. How does energy vary with changes in vCPU and frequency allocation? 2. What is the energy consumption over the lifecycle of a serverless function invocaiton?

To perform the measurement study, we use OpenWhisk, which is an open-source serverless platform, to run function invocations. We use EnergyLiteDaemon to measure the energy consumption of the function invocations.

5.1 Energy Variation with vCPU and Frequency Allocation

We ran a variety of workloads on a variety of vCPU and frequency configurations. We collected the vCPU utilization, energy consumption, and performance in terms of duration. The resource configurations are shown in Table 1.

We analyzed the data to understand how energy consumption varies with changes in vCPU and frequency allocation. We found that for functions that are highly parallelizable, such as float matrix multiplication, energy consumption decreases with the number of vCPUs, since the function duration decreases drastically. For functions that are not parallelizable, such as encryption, vCPU allocation does not affect energy consumption. For both types of functions, energy consumption as a function of frequency displays a convex curve, where energy consumption decreases with frequency until a certain point, and then increases with frequency. This is because the energy consumption of a vCPU is proportional to the frequency at which it runs, but the duration of the function is inversely proportional to the frequency at which the vCPU runs. More details are in Appendix A.2.

These trends support the idea that we can use machine learning to learn the relationship between vCPU, frequency, and energy consumption, and use this model to minimize energy consumption while meeting SLOs.

5.2 Energy Consumption Over the Lifecycle of a Serverless Function Invocation

6 Next Steps

6.1 Machine Learning Model

6.2 Adapting OpenWhisk

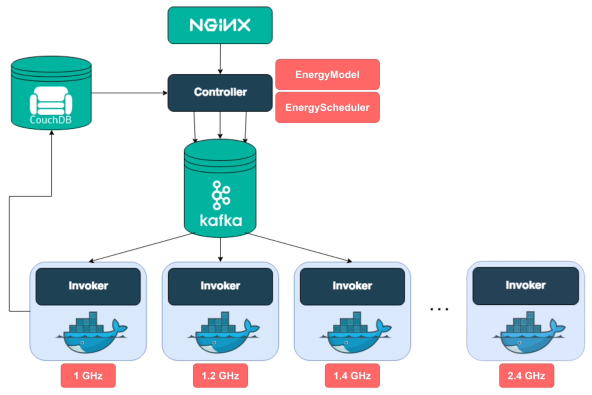


Figure 2: Adapted OpenWhisk System Overview

We will adapt OpenWhisk to incorporate Bayesian Optimization to drive a carbon-aware DFS and vCPU allocation model. Figure 2 shows the system overview.

The Bayesian Optimization model will be contained in a shim layer, called the EnergyModel. Whenever a user invokes a function, the EnergyModel is be used to determine the optimal vCPU and frequency allocation (using Bayesian Optimization). The EnergyModel will be trained on the data collected from the measurement study.

We will also create a custom scheduler, called EnergyScheduler, that will replace OpenWhisk's current scheduler. The EnergyScheduler will use the output of EnergyModel to determine the optimal vCPU to allocate for a function invocation. There will also be multiple invoker nodes, each running at a different frequency. The EnergyScheduler will send the request to the invoker that is running at the frequency predicted by EnergyModel.

References

[1] U. Gupta, M. Elgamal, G. Hills, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu. Act: designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 784–799, New York, NY, USA, 2022. Association for Computing Machinery.

[2] H. Hè, M. Friedman, and T. Rekatsinas. Energat: Fine-grained energy attribution for multi-tenancy. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, Hot-Carbon '23. ACM, July 2023.

A Appendix

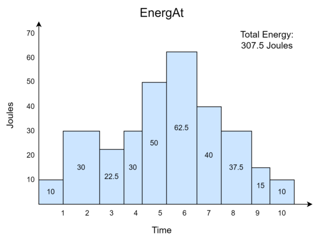
A.1 Energy Measurement

A toy example of the energy measurement using EnergAt and EnergyLiteDaemon is shown in Figure 3. Figure 3a shows the energy readings of EnergAt. Figure 3b shows the energy readings of EnergyLiteDaemon for one container. EnergyLiteDaemon doesn't capture all energy readings for a container due to the round-robin sampling approach. Figure 3c shows EnergyLiteDaemon's power calculation of of each energy sample. Figure 3d shows the energy interpolation of EnergyLiteDaemon using trapezoidal integration.

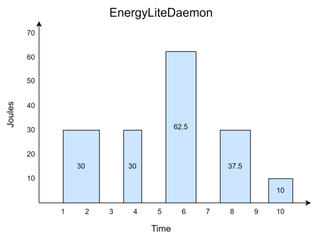
A.2 Energy Variation with vCPU and Frequency Allocation

The energy usage, duration, and CPU usage for Float Matrix Multiplication, Image Processing, and Encryption as a function of vCPU allocation are shown in Figure 4. For Float Matrix Multiplication, energy usage initially decreases and then flat-lines as the vCPU allocation increases. This is because matrix multiplication is highly parallelizable and results in higher vCPU utilization, leading to a decrease in duration and energy usage. For Image Processing, energy usage has no apparent correlation with vCPU allocation. This is because most of image processing tasks are non-parallelizable. For Encryption, energy usage, duration, and vCPU utilization do not change with vCPU allocation. This is because encryption is non-parallelizable and does not benefit from additional vCPUs.

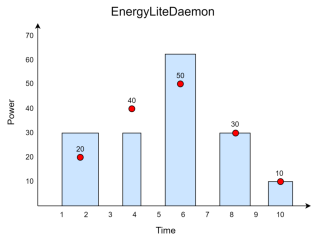
The energy usage and duration for Float Matrix Multiplication, Image Processing, and Encryption as a function of vCPU frequency are shown in Figure 5. For all functions, energy usage displays a convex curve as a function of frequency. This is because the energy consumption of a vCPU is proportional to the frequency at which it runs, but the duration of the function is inversely proportional to the frequency at which the vCPU runs. Therefore, energy usage decreases with frequency until a certain point, and then increases with frequency.



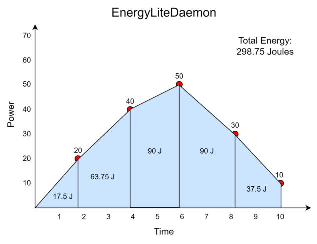
(a)



(b)



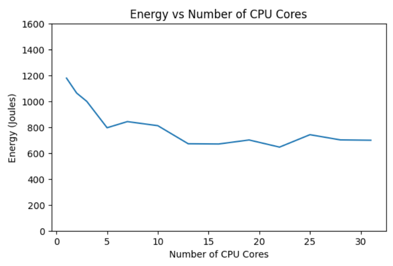
(c)



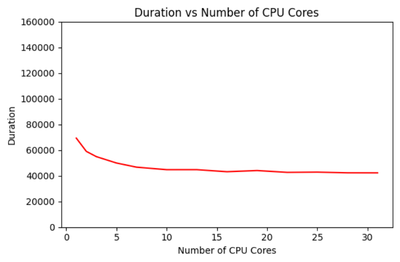
(d)

Figure 3: Toy example of energy measurement using EnergAt and EnergyLiteDaemon. (a) EnergAt energy readings. (b) EnergyLiteDaemon energy readings for one container. (c) EnergyLiteDaemon power calculation (red dots). (d) EnergyLiteDaemon energy interpolation using trapezoidal integration.

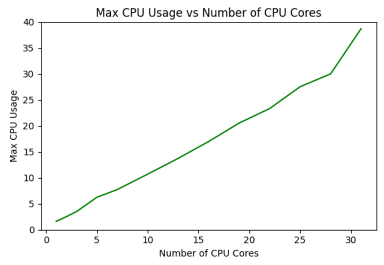
Float Matrix Multiplication



(a)

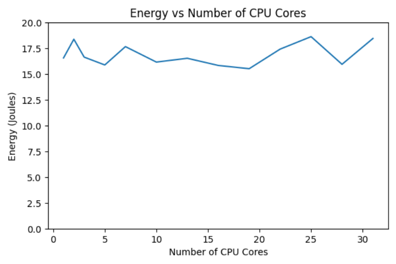


(b)

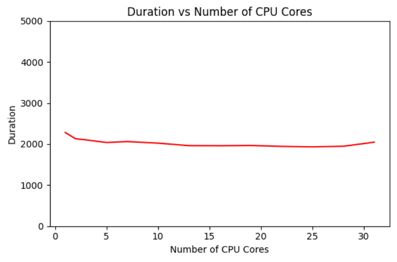


(c)

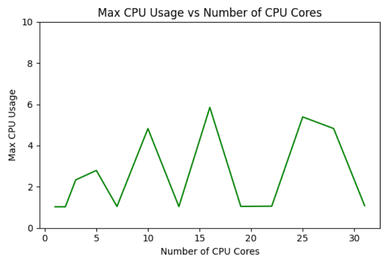
Image Process



(d)

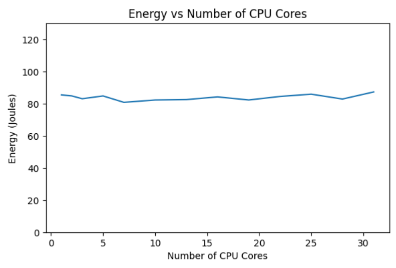


(e)

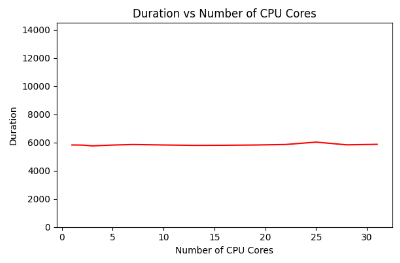


(f)

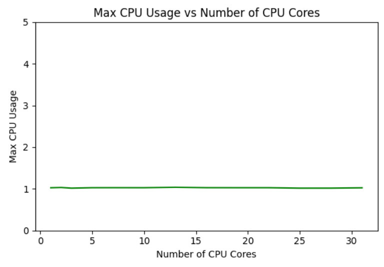
Encryption



(g)



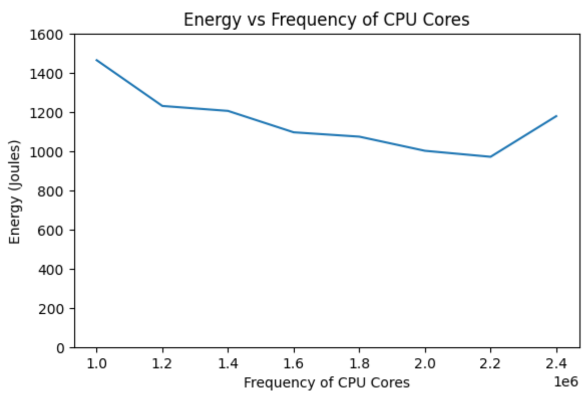
(h)



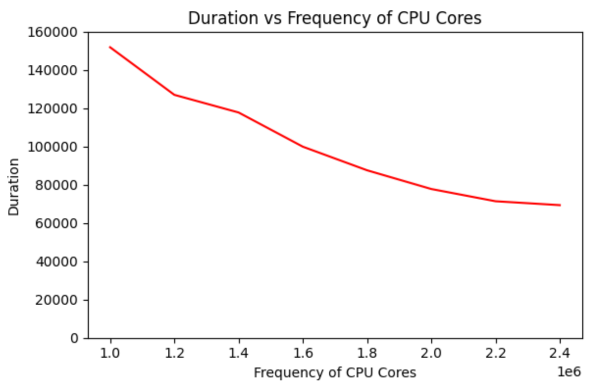
(i)

Figure 4: Energy Usage (a), Duration (b), and CPU Usage (c) for Float Matrix Multiplication, Image Processing, and Encryption as a function of vCPU allocation.

Float Matrix Multiplication

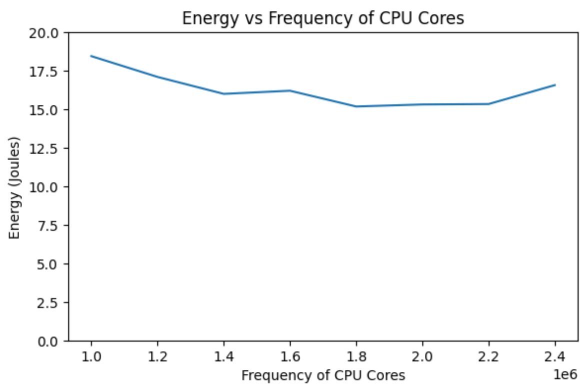


(a)

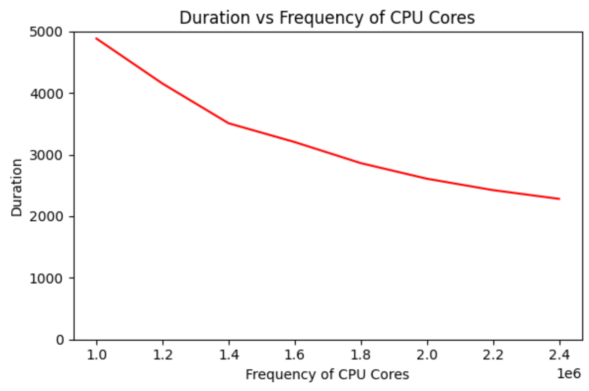


(b)

Image Process

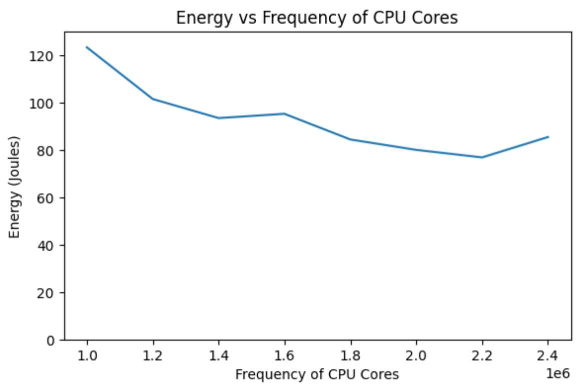


(c)

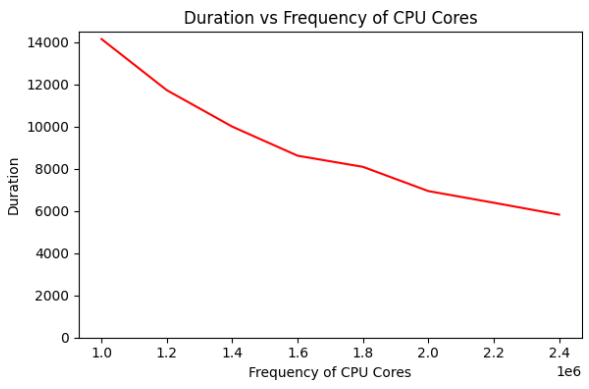


(d)

Encryption



(e)



(f)

Figure 5: Energy Usage (a), Duration (b), and CPU Usage (c) for Float Matrix Multiplication, Image Processing, and Encryption as a function of vCPU allocation.