Section A-Research paper

IMPROVING ENERGY EFFICIENCY IN CLOUD DATA CENTERS THROUGH AN BROWNOUT SOFTWARE SYSTEM STRATEGY BASED ON CONTAINERS

Thalari Prasanna[1]and Kishan Pal Singh[2]
[1]Department of Computer Engineering & Applications, Mangalayatan University, Aligarh, UP
[2]Department of Mechanical Engineering, Mangalayatan University, Aligarh, UP
Email: prasurani.t@gmail.com
doi: 10.48047/ecb/2023.12.si4.1637

Article History: Received: 05.01.2023 **Revised:** 20.02.2023 **Accepted:** 05.03.2023

Abstract

The excessive energy usage of cloud data centers has been a major cause for concern among researchers. This is at least in part due to the excessive waste of computer resources. In addition to energy consumption, data centers must deal with unexpected loads that lead to overloads and performance degradation. The concept of brownout, which involves the dynamic deactivation of application optional components (also known as containers/micro services), has emerged as a viable approach to managing overloads and energy consumption. In contrast, methods like consolidating virtual machines and adjusting voltage and frequency dynamically have trouble working when the entire data center is under stress. In this research, we propose a unified strategy for managing cloud data center energy consumption and brownouts using containerization. In addition, we use real-world trace data to evaluate the effectiveness of the scheduling options proposed by the prototype system.

Keywords: Cloud Computing, brownout, artificial Intelligence, Cloud data centers, Energy efficiency

1. INTRODUCTION

Cloud computing describes the use of shared networked computing resources rather than a user-managed local server to provide on-demand processing and data storage (cloud storage). Many large clouds have multiple data centers serving as home to the same or different functions. Pay as you go pricing is commonly used in cloud computing to ensure continuity. While it may reduce up-

front costs, it may lead to higher operating costs for consumers.

Cloud data centers and other large-scale infrastructures need vast amounts of energy. This thesis proposes a method based on the idea of brownout software systems and makes use of containerization technology in order to address the problem of energy consumption. During times of low demand or resource limitations, non-critical components or services' performance is dynamically reduced as part of the brownout software system method. By using this strategy, needless resource use and energy consumption can be reduced without affecting the system's general operation. [1]

On the other hand, containers offer a quick and easy way to deploy and manage applications in cloud settings. Resource allocation can be enhanced, improving energy efficiency, by enclosing application components and their dependencies in separate containers.

It looks into a number of things, such as:

a) Energy consumption analysis: The study looks at energy consumption trends and pinpoints areas that could use serious improvement. It determines the services that are candidates for brownouts and containerization by analyzing the resource usage of various services.

b) Brownout software system design: In this study, a design for the brownout software system is suggested. This design contains mechanisms for tracking resource utilization, spotting periods of low demand, and dynamically modifying the performance of non-critical parts or services.

c) Containerization framework: The study investigates containerization frameworks and technologies appropriate for cloud data centers. It looks into the use of containers to control resource allocation, encapsulate application components, and enable dynamic performance adjustments.

d) Energy efficiency evaluation: Comprehensive experiments and simulations are used to test the suggested technique. The brownout software system solution based on containers is implemented, and the energy consumption of the system is measured and compared before and after. The effects on energy savings, system performance, and user experience are evaluated.

e) Scalability and robustness analysis: To ensure the suggested strategy's efficacy in massive cloud data center environments, its scalability and robustness are evaluated. The thesis examines the system's capacity to deal with a range of workloads, resource requirements, and probable breakdowns.

- **Energy Consumption Analysis**

In order to improve cloud data centers' energy efficiency, energy consumption analysis must evaluate and comprehend the patterns of energy use inside the system. [2] The analysis's goals are to spot high energy consumption locations, figure out what causes them, and find areas where improvements can be made.

a) Data collection: Gathering pertinent information about energy use inside the cloud data center is the first stage in an examination of energy consumption. Usually, this entails placing energy-monitoring devices, like smart meters or energy sensors, at various locations throughout the infrastructure. These programs capture data on power usage in real-time, including total power used, power used by certain components (such as servers or cooling systems), and variations in energy demand over time.

b) Energy profiling: After the data has been gathered, it is crucial to develop an energy profile that describes the patterns of energy use of the cloud data center. In order to do this, it is necessary to analyze the data gathered to spot trends, peak usage times, and fluctuations in energy consumption. The system's overall energy requirements can be understood and potential areas for improvement can be found by profiling the energy usage.

c) Component-level analysis: Energy consumption analysis delves into the energy usage of individual components within the cloud data center. This includes servers, networking equipment, storage devices, cooling systems, and other infrastructure elements. The analysis aims to identify the components that consume the most energy and understand the factors influencing their energy consumption. This may involve examining power usage during different operational states (e.g., idle, active, peak load) and identifying any inefficiencies or energy-intensive operations.

d) Workload analysis: Understanding the connection between energy consumption and system workload is the main goal of workload analysis. It is feasible to detect energy-hungry jobs or applications by looking at the energy usage trends under various workloads, such as varying degrees of user demand or resource utilization. This analysis provides information on workload management strategies that can maximize energy efficiency by establishing the relationship between workload characteristics and energy usage.

e) Energy hotspots identification: The goal of energy consumption analysis is to pinpoint specific regions of the cloud data center that use a lot of energy or are inefficient. These locations, which are sometimes referred to as "energy hotspots," may

include data centers with ineffective cooling systems, servers that aren't being used to their full potential, power-hungry programs, or ineffective power distribution systems. The ability to target interventions and optimization efforts to cut energy use and boost overall efficiency is made possible by identifying these hotspots.

f) Benchmarking and comparisons: Energy consumption study often involves comparing the cloud data center's energy performance to that of similar facilities as well as to industry standards and best practices. This enables evaluations against comparable data centers or benchmarks to spot areas for development. Setting targets for energy efficiency, monitoring progress, and identifying possible energy-saving strategies used by top-performing data centers are all made easier by benchmarking.

g) Energy efficiency metrics: To measure the success of optimization efforts, energy consumption analysis uses a variety of energy efficiency metrics. Energy Proportional Computing (EPC), Effectiveness of Energy Reuse (ERE), and Efficiency of Data Center Infrastructure (DCiE), and Power Usage Effectiveness (PUE) are examples of common measures. These metrics offer concrete assessments of energy efficiency and act as benchmarks for progress over time.

- **Brownout Software System Design**

The process of developing a system architecture and methods that allow for the dynamic adjustment of non-critical components or services during times of low demand or resource limitations is known as brownout software system design. This design strategy tries to maximize the use of resources and energy in cloud data centers.

A brownout software system's design involves a number of important factors. First, a monitoring system that continuously tracks the system's resource utilization and performance indicators must be created. Data is gathered by this monitoring system on things like CPU and memory usage, network traffic, and application response times. The system can determine times of low demand or resource availability by examining these variables [3].

Second, the design of the brownout software system comprises a decision-making element that assesses the gathered monitoring data and chooses the appropriate time to start the brownout process. This component makes decisions regarding the removal of non-critical functionality by using established policies or algorithms to evaluate the system's current condition. These choices may be influenced by user preferences, energy consumption thresholds, or task levels[4].

The system architecture contains techniques for dynamically changing the performance

of non-critical components or services after the choice to start a brownout has been made. The frequency of resource-intensive tasks may be decreased, non-essential operations may be given less priority, or resource utilization may be constrained via throttling techniques. The architecture makes sure that vital features are kept while non-important components run at reduced levels by dynamically changing the system's behavior.

The design may include strategies like modular software architectures, microservices, or containerization to enable efficient brownout adaptations. These methods enable fine-grained control over individual parts, making it simpler to deliberately decrease their resource utilization without affecting the functionality of the system as a whole. In particular, containerization offers application component isolation and encapsulation, enabling effective resource management and correction depending on brownout tactics [5].

The design of the brownout software system also takes into account mechanisms for adaptability and feedback. It continuously assesses the success of brownout decisions and modifies the behavior of the system in response to real-time feedback and user demands. As long as performance and user experience are kept at acceptable levels, the system can optimize energy usage.

- **Containerization Framework**

A set of tools, technologies, and procedures known as a containerization framework make it easier to deploy, manage, and isolate application components inside of containers. Containerization is an essential part of the brownout software system strategy based on containers in cloud data centers because it offers a simple and effective method for packaging and running software applications [7].

A containerization system, at its core, offers a setting in which applications and their dependencies can be contained within separate containers. The application code, libraries, runtime environments, and other necessary configurations are all included in these containers, which function as self-contained entities. In order to abstract away the underlying infrastructure and enable portability across many contexts, the framework makes sure that each container runs independently of the host system.

The container runtime, which controls the lifespan of containers and offers the necessary isolation and resource allocation, is one of the essential components of a containerization system. Docker and containerd are two well-known container runtimes. With the help of these runtimes, containers can be created, executed, and terminated while remaining isolated from other containers and the host system[8].

A containerization framework often comprises a container orchestration system in addition to the container runtime. A cluster of machines' worth of containers are deployed, scaled, and networked using this method. The widely used container orchestration software Kubernetes handles workload distribution, automates container

administration duties, and assures high availability and scalability [9].

A containerization framework additionally offers a collection of tools and utilities to simplify container operations. These tools make the processes of building and configuring containers simpler, automate the deployment and scaling procedures, and allow for the monitoring and logging of containerized applications. Helm, Prometheus, and Docker Compose are a few examples of such tools.

Containerization frameworks must also prioritize security. They include tools for managing container images securely, enforcing container isolation, and preventing unauthorized access. Applications that use containers benefit from security features including user namespaces, container network restrictions, and container image signing.

A containerization framework also includes container registries, which allow for the sharing and storing of container images. Versioning, access control, and distribution functions are offered by these registries, which serve as repositories for container images. Popular container registries include Docker Hub and Google Container Registry[10].

- **Energy Efficiency Evaluation**

Evaluation of energy efficiency entails determining the efficacy of energy-saving tactics and optimization methods used in cloud data centers. This evaluation procedure intends to measure how these actions affect user satisfaction, system performance, and energy consumption reduction. A container-based brownout software system's efficacy can be assessed with the use of data gleaned from a thorough investigation of energy efficiency [12].

Before putting the energy-saving methods into action, the evaluation process normally starts with creating a baseline measurement of energy use. This baseline offers a standard by which the success of optimization efforts can be measured. For the purpose of gathering precise and current data on power use, energy meters, power monitoring devices, and energy sensors are utilized.

After the baseline has been established, the evaluation process involves applying the container-based brownout software system method and measuring the energy savings that ensue. The investigations entail tracking the cloud data center's energy use under various conditions, such as varied workloads, various containerization schemes, or various degrees of brownout adjustments [15].

Various performance metrics are gathered during the evaluation to evaluate the effect of the energy-saving methods on system performance. Response times, throughput, latency, and resource usage are a few examples of these metrics. These measurements can be compared before and after the brownout software system strategy has been implemented to get understanding of the trade-offs between energy savings and system performance.

Another critical component of the evaluation is the user experience. The effect of energy-saving initiatives on user satisfaction can be evaluated via user surveys, feedback, or quality of service indicators. It is important to consider the user's perspective to make sure that improvements in energy efficiency do not result in a decrease in the quality or responsiveness of the services provided by the cloud data center.

Benchmarking the cloud data center's energy efficiency against industry standards or best practices is another step in the evaluation process. This benchmarking enables comparisons with comparable data centers or benchmark models to pinpoint areas that still require improvement. It helps set reasonable energy efficiency objectives and provide a context for analyzing the data center's performance in comparison to its competitors.

To establish the financial advantages of energy efficiency measures, the study may also involve a cost analysis. This analysis takes into account things like possible return on investment, operating cost savings, and energy savings. By highlighting the financial benefits in addition to the environmental benefits, it aids in justifying the installation of the brownout software system strategy based on containers.

- **Scalability and Robustness Analysis**

In order to increase energy efficiency in cloud data centers, scalability and robustness analysis focuses on evaluating the system's capacity to manage a range of workloads, resource demands, and probable breakdowns while maintaining optimal energy consumption levels. This investigation confirms the scalability, capability, and reliability of the brownout software system strategy based on containers.

Analyzing a system's scalability entails determining how well it can handle growing workloads without sacrificing energy efficiency. It evaluates how well the containerization architecture and brownout software system can scale up or down in response to shifting needs. As the workload increases, this analysis takes into account variables including the system's reaction time, throughput, and resource distribution. It seeks to pinpoint any performance degradations, resource limitations, or bottlenecks that might impede scaling. It is possible to take the necessary steps to ensure efficient energy use even during times of high demand by being aware of the system's scaling restrictions.

In a robustness analysis, the system's resilience to errors or unfavorable circumstances is assessed. It evaluates the system's capacity for failure recovery, performance maintenance, and continuous operation while consuming the least amount of energy possible. This analysis takes into account things like system resilience tactics, disaster recovery plans, and fault tolerance methods. Its objective is to locate probable weak spots, single sites of failure, or performance degradation under failure conditions. The system may be strengthened and made more energy-efficient by

recognizing and fixing these problems, assuring uninterrupted service delivery.

Stress testing the system under difficult circumstances is another step in the scalability and robustness examination. To assess the system's behavior and performance, testing may involve subjecting it to demanding workloads, resource limitations, or simulated failure scenarios. The analysis evaluates how the system reacts under such stressful circumstances, taking into account its energy consumption patterns, resource use, and capacity for recovery and service level maintenance. Stress testing the system identifies any scalability or robustness flaws, enabling for the appropriate enhancements or modifications to be done.

The analysis also takes into account how scalability and robustness measures affect energy usage. It assesses if the added scalability and resilience mechanisms improve energy efficiency or have a negative impact on energy use. By balancing scalability, robustness, and energy efficiency, this study ensures that the system can manage a range of workloads and failures while using the least amount of energy possible.

2. LITERATURE REVIEW

M. Xu, R. Buyya, and Co. (2019)[14]: This research proposes a software-based brownout solution for container-based clouds as a means of dealing with overloads and reducing energy consumption. We detail its Docker Swarm container-based architecture and implementation. The existing Docker Swarm can be merged with the suggested method with no changes to its specifications. To demonstrate BrownoutCon's potential in delivering energy-efficient services during brownouts, we developed a number of ways to regulate containers and conducted trials on the French Grid'5000 cloud infrastructure. Research indicates that our software's current rules can reduce energy consumption by 10% - 40% compared to historical norms without compromising service quality.

Researchers from Marahatta et al.(2019)[6] The rapid growth in both size and number of cloud data centers (CDCs) can be directly attributed to the increasing demand for both cloud and high-performance computing. Due to the inefficient use of resources and excessive energy consumption, this has the unintended consequence of posing new problems. Thus, ensuring energy efficiency in CDCs and maximizing resource usage are necessary. Task scheduling is a practical strategy for achieving resource utilization and energy efficiency in CDC. Although a number of task scheduling strategies have been put forth in the literature, it seems that none of them include a classification-based merging notion for real-time jobs. Thus, this study presents an energy-efficient dynamic scheduling method (EDS) of real-time jobs for virtualized CDC. According to a previous scheduling record, the heterogeneous tasks and virtual machines in the scheduling scheme are first categorized. Then, related jobs are combined and planned to take full advantage of the host's operational status. In order to achieve energy preservation while building and

removing the virtual machines, energy efficiency and optimal operating frequencies of diverse physical hosts are also used. According to experimental findings, EDS greatly outperforms conventional scheduling methods in terms of overall scheduling performance, CDC resource use, task guarantee ratio, mean reaction time, and energy consumption.

Venkatesan, N.(2022)[13] Cloud computing is used to manage and store the vast amount of data generated by IoT. Other drawbacks of cloud computing include: B. Real-time program latency increases, slowing response time. This can result in even more energy and material wasted. To solve this problem, researchers developed a new interpretation of cloud computing: fog/edge computing. Despite providing a solution, this paradigm for real-time applications is required due to the low CPU power available in fog nodes and the short battery life of edge devices. This research looked at energy consumption, SLA, and execution to propose a controller for a cloud-fog environment container-based micro-service application. The outcomes show that the brownout technique reduces SLA violations and energy usage more than the execution time does.

According to Saboor et al.(2022)[11] The proposed framework suggests four essential agent services: intelligent partitioning for microservice classification; dynamic allocation for distributing microservices among containers ahead of execution; resource optimization for redistributing workloads and ensuring optimal resource use; and mutation actions for modifying the microservices in response to cloud data center workloads. The suggested framework was partially tested in a novel simulation environment, proving its viability and usefulness in a cloud computing setting. As a result, it was found that the number of calls made over the network, energy consumption, and carbon emissions could be reduced by using the proposed service.

2. MODELLING

We shall introduce our system's models in this section and outline the issue we hope to solve.

- **Power Consumption**

We use the derived server power model. Power of the server i is $P_i(t)$ that the CPU use dominates:

$$P_i(t) = \begin{cases} P_i^{idle} + u_i \times p_i^{dynamic}, N_i > \\ 0, N_I = 0 \end{cases} \quad (1)$$

$P_i(t)$isconsists of both active and passive energy. Server utilization directly correlates to the dynamic power consumption, while the static power consumption is assumed to be constant.u_i. If a server does not host any containers or microservices, it is turned down to save energy. All containers and microservices running on the server consume the same CPU time as the server itself:

$$u_i = \sum_{j=1}^{N_i} u(MS_{ij}(t)) \quad (2)$$

whereMSi,j alludes to the jthmicroservice on server I, Ni addresses the quantity of

microservices conveyed to server I. Furthermore, u(MSi,j (t)) alludes to the computer chip use of the holder or microservice at a particular period t.

The total energy used with M servers throughout the time interval t is then:

$$E(t) = \sum_{i=1}^{M} \int_{t-1}^{t} P_i(t)dt$$
(3)

- **Quality of Service**

We use a variety of QoS measures as shown below to model the QoS need in our system.

We classify hosts into two states according to their utilization:

Overload and not overload. Host performance is affected by congestion. A host is considered overloaded when its utilization exceeds the preset utilization threshold. We use the introduced metric, called Overloaded Time Ratio (OTR), to assess this QoS indicator as being independent of workloads:

$$OTR(u_t) = \frac{t_0(u_t)}{t_a}$$
(4)

If u_t is the CPU utilization threshold at which the host becomes overloaded, to is the time period during which the host is overcrowded (which is relevant to u_t), and ta is the aggregate time period during which the hosts are overloaded. This statistic is set up as a Quality-of-Service cap on how high OTR can go. If the system's SLA is 10%, for example, no host will be overburdened for more than 10% of the time. The limitation of the SLA is stated as follows:

$$\frac{1}{M}\sum_{i=1}^{n=M} OTR_n(u_t) \leq 0.1$$
(5)

where M is the total number of servers in the facility. As mentioned earlier, brownout-based techniques periodically check host status and trigger brownouts if there are too many clients. Therefore, the rate of occurrence of voltage sags is also captured by this metric.

We measure the time it takes to respond to a request as "response time". It also evaluates how long it takes to respond relative to the fastest k percentile of requests. where k is 90, 95, 99, or some other number. So if the 95th percentile response time is 1 second, 95% of the requests will be executed within that period.

SLA Violation Ratio indicates the percentage of failed requests due to system overload when consumers send Numa requests, and Num_{err}The percentage of incorrect responses is displayed as:

$$\textbf{SLAV R} = \frac{Num_{err}}{Num_a}$$

- **Optimization Objective**

As mentioned earlier, it is important to reduce overall power consumption while maintaining QoS by avoiding congestion, providing fast response times, and reducing error rates. So we face an optimization dilemma.

$$\min \sum_{t=1}^{T} E(t)$$

$$\frac{1}{M}\sum_{t=1}^{T} OTR_n\,(u_t) \leq \alpha$$

$$R_{avg}^{t} \leq \beta, R_{95t\square}^{t} \leq \emptyset$$

SLAV R$\leq \gamma$

Where $\sum_{t=1}^{T} E(t)$is the data center's overall energy consumption, α is the maximum acceptable mean reaction time under stress.,$R_{avg}^{t}$is the normal response time and β is the allowed commonplace response time;R_95th^t is the restriction of 95th percentile response time and φ is the allowed the 95th percentile response time, and γ is allowed SLA encroachment extent.

4. SCHEDULING POLICY

We shall introduce our brownout-based scheduling principles in this section. In order to more effectively use host resources, we need an auto-scaling method before using the brownout approach.

- **Auto-scaling Policy**

The auto-scaling method is used, which relies on a predetermined threshold-based strategy. In profiling testing, we determine when the host is too busy to respond to requests in a timely manner by establishing a threshold for request overload. The auto-scaling method's master node first counts the number of active hosts, then uses profiling data to determine whether requests have reached an unsafe level of volume, and then uses this information to determine the current request rate. One advantage of moving the timeframe is that more recent values are given more weight. Ma' is the ratio of the current request rate to the congestion threshold and is used to determine the number of hosts currently required. If the number of hosts required exceeds the number of active hosts, add additional hosts to the system to provide the required services or put additional computers into sleep mode to reduce power consumption. The primary node then checks the total number of online hosts.

- **Initial Deployment**

Using the service and initial deployment configuration data defined in the Docker compose file. a storage space for information related to the Ubuntu-based recommendation engine service. The recommender system is set up as a removable microservice with two copies. This service can also only be deployed to Docker worker nodes, which severely limits its potential deployment environments. Data from the user database service, which can't be deployed anywhere else but the Docker master node.

- **Optimization Deployment with Scheduling Policies based on Brownout**

We have suggested the following three brownout-based regulations:

- **Lowest Utilization Container First (LUCF)**

At the point when a host is over-burden, the Least Use Holder First strategy picks a gathering of compartments with the most reduced use, bringing down the usage underneath the over-burden limit. Let oclit

be the discretionary compartment list on have howdy at time span t. Let P(oclit) to be the power set of oclit, the LUCF finds the deactivated compartment list dclit, which is remembered for P(oclit). The rundown of latent compartments lessens the worth inconsistency between the expected use decrease u_i^r and its use u(dclit) Condition characterizes the rundown of deactivated holders.

$$dcl_{i,t} = \begin{cases} \{HP \le TP, |u_i^r - u(dcl_{i,t})| \to min\} \\ \emptyset, \end{cases}, IfP_i(t) \ge TP, IfP_i(t) < TP \quad (6)$$

HP is the expected not entirely settled by the host use model. HUM(hi, ui− u(dclit)) that gathers the host power subject to the host's use ui − u(dclit); TP is the power edge of the over-burden h_i .

- **Minimum Number of Components First Policy (MNCF)**

As expressed in Condition, the Base Number of Compartments First (MNCF) strategy picks the most modest number of holders while bringing energy utilization down to deactivate less administrations. Since the MNCF technique is generally like the LUCF calculation that was recently presented, we wo exclude the MNCF pseudocode here.

$$dcl_{i,t} = \begin{cases} \{HP \le TP, |u(dcl_{i,t})| \to min\} \\ \emptyset, \end{cases}, IfP_i(t) \ge TP, IfP_i(t) < TP \quad (7)$$

- **Random Selection Container Policy (RSC)**

To eliminate energy use, the Irregular Determination Holder strategy (RSC) utilizes an arbitrary choice of various discretionary compartments. RSC is displayed in Condition and depends on a discrete irregular variable (X) with uniform circulation that haphazardly picks a subset of dcl_{it}.

$$dcl_{i,t} = \begin{cases} \{HP \le TP, X = U(0, |ocl_{i,t}| - 1)\} \\ \emptyset, \end{cases}, IfP_i(t) \ge TP, IfP_i(t) < TP \quad (8)$$

5. PERFORMANCE EVALUATION

On the INRIA Grid'5000 testbed, we are experimentally evaluating our methods for Wikipedia web workload.

- **Workload**

To recreate the workload of Wikipedia users on October 17, 2007, we use real trace from Wikipedia requests. We use 5% of the total size of the user requests when scaling the workload set to accommodate our research. We generate the requests using JMeter, a toolbox for load testing and performance monitoring, by replaying the Wikipedia trace n_r

Table 1: Predicted and actual requests rates

Time Interval	0	200	400	600	800	1000	1200	1400
Actual	210	215	150	200	210	300	350	250
Predicated	205	210	160	200	205	310	340	260

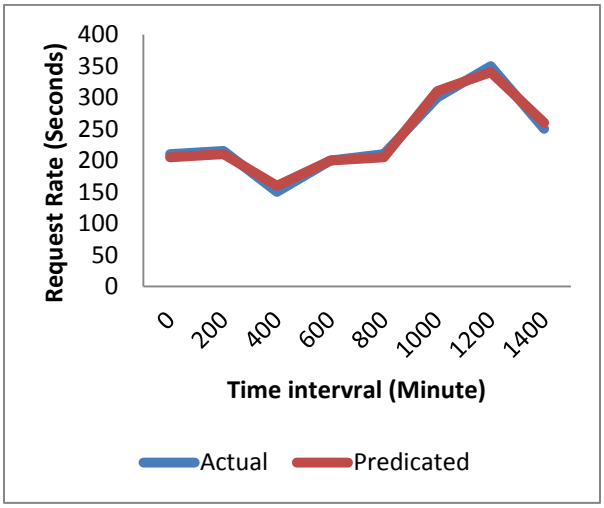

Figure 1: Predicted and actual requests rate

in view of a sliding window, is the anticipated solicitation rate. Let Lw to be the window size, and nr(t) to be the solicitation rate at t, we gauge n_r as:

$$n_r(L_w) = \frac{1}{L_w} = \sum_{t=0}^{Lw-1} n_r(t) \qquad (9)$$

We chose a sliding window size of 5 for our tests. The expected rates and the actual rates for requests per second during the day are fairly close.

- **Testbed**

Grid'5000, a French trial framework stage, fills in as our testbed. We execute the bunch with the APIs for estimating power at the office in Lyon, which is situated in southeastern France. Each of the hubs in the model framework on the Lattice 5000 groups are sent with Dockerswarm, and their obligations are as per the following, as shown by the engineering of the framework:

- The brownout regulator, which stores booking arrangements, the Java Runtime, and the Ansible tool stash, are administrations that should be conveyed on the expert hub, and this hub is at first positioned to serve in this limit.
- Worker nodes: These nodes are employees that carry out duties aside from those of the database and master nodes. There are many worker nodes in our system.
- Database services are deployed to a special worker node that handles just database operations.

The workload trace and JMeter are installed on the request node, which is used to send requests to the cluster. This node's placement is irrelevant to the simulation of user behavior. As part of our study, we additionally deploy this node in the Lyon cluster to mitigate the impact of uncontrolled data flow leaving the cluster.

The following describes the hardware information of the nodes we chose:

- Sun Fire V20z is the machine model. This model's maximum power output is 237 Watts, while its sleep mode power output is 10 Watts;
- Debian Linux as the operating system;
- Processor: AMD Operon 250, 2 cores, 2.4 GHz;
- 2 GB of memory

The Docker Swarm master node is one of the nodes, and the worker nodes are the

other nodes. To reduce the effects of CPU usage and network traffic, all necessary apps, Java, Docker, Ansible, and JMeter are just a few of the pre-installed programs.

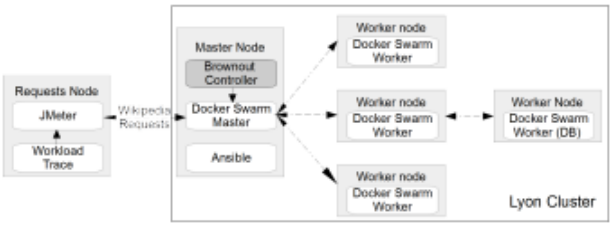

Figure 2: Architecture of Prototype System

6. RESULTS

In order to gauge the efficacy of our recommendations, we compare their results to those of three standard policies.

First, there's the Non-Power-Aware (NPA) policy, in which hosts are left on all the time and no optimizations are made to save energy. NPA has access to 13 different nodes.

The brownout (BOB) strategy prioritizes actual utilization over reservations to reduce response times and brownouts. BOB's brownout feature is based on response time. If the response time is faster than expected, this method will gradually increase application usage. BOB is limited to 10 knots and can experience traffic jams.

Third, there is the Auto Scaling (Auto-S) policy. It adds and removes hosts to the service on-the-fly. We also offer Auto-S 10 nodes to help handle peak loads. Our recommended policy allows access to the same data as BOB and Auto-S.

The experiments below focus primarily on examining congestion thresholds and option usage rates.

What we mean by the term "overloaded threshold" is the percentage of CPU time used that indicates whether or not a host is overworked. This is the default as studies have shown it to reduce energy consumption. From 60% to 90%, it varies by 10% intervals. With lower overloaded thresholds, say 50%, hosts are more likely to be labeled as overloaded, leading to inefficient resource consumption.

Percentage of elective use: As a result, you can see how much energy can be saved by reducing the CPU time spent on optional containers. This metric is being investigated because of its potential impact on energy use. From 10% to 40%, it fluctuates by 10% points. We settled on these parameters because of the disastrous effects of extremely high levels of optional usage, such as 50%, on both revenue and user experience.

7. CONCLUSION

Overcrowding is a well-known problem in cloud data centers, and blackouts have proven to be an effective solution. Implementing brownouts can further reduce energy consumption. In this post, we designed a brownout-based architecture that temporarily powers off containers of applications or microservices that are not actively used to save power. This framework provides an integrated approach to power outage handling and energy management in container-based cloud environments. We

also outline a number of guidelines for finding suitable inactivation containers and evaluating their effectiveness in sham systems. Compared to baseline values, our recommended policies improve energy efficiency, response times, and service level agreement (SLA) violation rates in real-world test environments.

8. FUTURE PLAN

In the future, we plan to investigate how the brownout technique can be applied to other model-based processes such as (1) map-reduce. The second is a multitasking application and the third is a stream-oriented application workload.

REFERENCES

1. (2014) Data center energy: Reducing your carbon footprint — data center knowledge.
2. A. Belog1azov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," Concurrency and Computation: Practice and Experience, vol. 24, no. 13, pp. 1397–1420, 2012.
3. A. Beloglazov and R. Buyya, "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints," IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 7, pp. 1366–1379, 2013.
4. A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," Future Generation Computer Systems, vol. 28, no. 5, pp. 755–768, 2012.
5. C. Klein, M. Maggio, K.-E. Arz ˚ en, and F. Hern ´ andez-Rodriguez, ´ "Brownout: building more robust cloud applications," in Proceedings of the 36th International Conference on Software Engineering, 2014, pp. 700–711
6. Marahatta, A., Pirbhulal, S., Zhang, F., Parizi, R. M., Choo, K. K. R., & Liu, Z. (2019). Classification-based and energy-efficient dynamic task scheduling scheme for virtualized cloud data center. IEEE Transactions on Cloud Computing, 9(4), 1376-1390.
7. P. Delforge. (2014) Data center efficiency assessment - scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers. [Online].
8. Q. Chen, J. Chen, B. Zheng, J. Cui, and Y. Qian, "Utilizationbasedvm consolidation scheme for power efficiency in cloud data centers," in 2015 IEEE International Conference on Communication Workshop (ICCW). IEEE, 2015, pp. 1928–1933.
9. R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in 10th IEEE

International Conference on High Performance Computing and Communications, 2008, pp. 5–13.

10. S. Newman, Building Microservices. ” O’Reilly Media, Inc.”, 2015.
11. Saboor, A., Hassan, M. F., Akbar, R., Shah, S. N. M., Hassan, F., Magsi, S. A., &Siddiqui, M. A. (2022). Containerized microservices orchestration and provisioning in cloud computing: A conceptual framework and future perspectives. Applied Sciences, 12(12), 5793.
12. T. Bawden. (2016) Global warming: Data centres to consume three times as much energy in next decade, experts warn. [Online].
13. Venkatesan, N. (2022). Energy and QoS Optimization in Fog/Edge Computing Via Brownout (Doctoral dissertation, Dublin, National College of Ireland).
14. Xu, M., &Buyya, R. (2019). BrownoutCon: A software system based on brownout and containers for energy-efficient cloud computing. Journal of Systems and Software, 155, 91-103.
15. Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser, “Renewable and cooling aware workload management for sustainable data centers,” in ACM SIGMETRICS Performance Evaluation Review, vol. 40, no. 1. ACM, 2012, pp. 175–186.