IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 24 December 2023, accepted 31 January 2024, date of publication 13 February 2024, date of current version 22 February 2024.

*Digital Object Identifier 10.1109/ACCESS.2024.3365674*

RESEARCH ARTICLE

# A Cloud Monitor to Reduce Energy Consumption With Constrained Optimization of Server Loads

**ERGUN BIÇICI**
Huawei Turkey Research and Development Center, 34768 Istanbul, Turkey
e-mail: ergun.bicici@huawei.com

**ABSTRACT** As the energy consumption of cloud computing increases, resource optimization becomes more important in this dynamically changing computation environment. Cloud monitor is an effort for better job distribution models and cloud load optimization on cloud computing platforms to reduce energy consumption, decrease computation and waiting time in the queue, and improve the overall utilization of resources. We use linear programming to optimize the distribution of jobs whose arrival and duration times are Poisson distributed to the servers. Our simulations show 1) improvements in job distributions with better correlation with the energy consumption of servers, 2) possible increased utilization of resources in some settings, and 3) significant reductions in energy consumption that reach 42%. We present the mathematical formulation of our model as well as how to represent it as a linear programming problem. The constraints introduced allow the mathematical definition of the optimal distribution, optimal redistribution, and optimal elasticity for cloud computing. Cloud monitor has potential to contribute to more sustainable and efficient cloud computing systems.

**INDEX TERMS** Cloud computing, cloud load optimization, energy reduction.

## I. INTRODUCTION

Cloud computing is now consuming 1.5% of the global electricity [1] and it is estimated that in 2030, data centers will use around 3% to 13% [2]. Cloud computing load optimization is gaining interest with a shift towards serverless infrastructures [3]. Optimizing and scheduling container loads on the cloud can be solved using integer linear programming [4], in which a scheduling model based on the allocation of virtual machines (VMs) to available nodes and their redistribution under certain conditions is defined. In [4], integer linear programming was used to satisfy the relevant constraints that determine the capacity of the data center, nodes, and task requests of the VMs. In a simulation of energy savings for different scheduling methods where VMs start with $t_b$ inter-arrival times and last for $t_\mu$ duration, using migration can decrease the energy consumption by up to 42% for $t_b = 25$ seconds and $t_\mu = 30$ seconds [4], which does not satisfy the stability condition of the queueing theory (Section II). Simulations were run with 100 heterogeneous physical nodes, each with one CPU with 1000, 2000, or 3000 Million Instructions Per Second (MIPS) processing power, 8 GB RAM, and 1 TB disk. For utilization, two threshold settings can be used where the lower is for deciding the evacuation of all VMs from the node and switching it off if the utilization is lower to decrease energy consumption and the higher threshold is for redistributing some VMs to prevent delays in the service [5].

Dynamic load-balancing is a technique for the distribution of computing resources and workloads between systems, networks, and servers, and current nature-inspired approaches were reviewed in [6]. A linear programming approach for distributing the workload among a minimum number of servers was developed to reduce the costs of a usage-based pricing model for cloud computing [7]. A mixed integer linear programming model to optimize the selection of cloudlets, clouds that are well-connected and distributed, and computing resource allocation was proposed [8]. A cloudlet is a computer or a cluster of computers with an internet connection [9].

A machine learning-based approach for workload forecasting and energy state estimation in cloud data centers is

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

proposed in [10] to aid resource management decisions. For workload prediction, they found that Gated Recurrent Unit (GRU) achieve the best results where they choose regression-based methods to estimate numerical output variables like CPU utilization. For energy state estimation, they use semi-supervised affinity propagation based on transfer learning (TSSAP) for classification when identifying the VM energy state classes such as low, high, and critical. In [10], they model workload forecasting and energy state estimation as separate tasks and they do not show performance improvements in an overall cloud architecture where their prediction models are used. Whereas, we show improvements in a cloud computing environment when our scheduling algorithm is used.

The non-intrusive power disaggregation (NIPD) technique [11] uses power mapping functions between the states of servers and their power consumption to disaggregate power consumption data into individual components such as servers, storage, and networking equipment. They use dstat tool [12] to collect 6 metrics: total CPU utilization, total memory utilization, disk reading/writing, and network traffic receiving/sending statistics. The technique enables more efficient energy management in data centers and is found to have an error rate of 2% at server level. However, NIPD is only an estimation technique for data centers that uses real-time usage data, which need not be available for prediction purposes and does not provide a solution to dynamic load balancing.

Another approach for maximizing revenue and minimizing power consumption on the cloud uses Lyapunov optimization for VM scheduling [13]. They optimize for increasing the revenue and subtracting the cost from power usage efficiency. Their VM scheduling algorithm prioritizes VMs according to their queue backlogs. A reliability-aware server consolidation strategy is proposed to perform energy-efficient server consolidation while considering reliability and profitability [14].

An algorithm that prioritizes the tasks regarding their execution deadline is proposed in [15] and the Dynamic Voltage and Frequency Scaling (DVFS) method is used to reduce the consumed energy of the machines for processing low-priority tasks. DVFS is a power management technique that adjusts the voltage and frequency of a processor or system to match the current workload. This technique is used to reduce power consumption and improve energy efficiency. Their simulations show that energy consumption is optimized by 12%. However, their approach is heuristic in nature using fixed thresholds and workflow based, which hinders its applicability. Another DVFS based method using a multi-criteria scheduling algorithm to manage energy consumption on the cloud system through energy scaling decisions is proposed [16].

In [17], an Adaptive Multi-Objective Teaching-Learning Based Optimization (AMO-TLBO) algorithm for dynamic resource allocation in cloud computing is presented. They try to decrease the makespan or the overall completion time of workflow tasks and the total monetary costs from computation and communication and increase resource utilization. In all of these metrics, they obtain improvements compared with the other three methods they compared with over simulations with 100 tasks. However, this approach is also heuristic in nature and workflow based, which again hinders its applicability.

**TABLE 1. Related work categories.**

| Topic | References |
|---|---|
| general optimization | [13, 14] |
| genetic algorithms | [19] |
| linear programming | [4, 7, 8] |
| machine learning based | [10, 19] |
| DVFS based | [15, 16] |
| heuristic based | [11, 15, 17] |

The components of applications can be served with microservice components on the cloud which communicate over well-defined APIs. An integer linear programming model for CPU optimization of microservices is provided in [18] where 50% reductions in CPU reservations is achieved.

Another approach optimize cloud computing resources according to actual demand to reduce the monetary costs [19]. They predict the usage of the resource for the next 7 days and using a combination of multiple predictions from models including decision trees and neural networks, they calculate a cost-optimal cloud resource configuration every hour using a particle swarm optimization algorithm. Over the 10 month period covered by the tests, the costs are reduced by 85% with savings of 6128 Euros.

In Table 1, we present the related work in tabular format.

The demand for a resource can increase, decrease, or remain constant. Even if demand does not increase, we would like to optimize the distribution of consumers to resources and adapt the cloud accordingly. These trends also help determine the most likely scenarios to be prepared for.

The cloud monitor uses results from queueing theory (Section II) during statistical simulation experiments that provide possible contexts for distribution and redistribution where server and job specifications are obtained with normally distributed statistics. Jobs are started and moved according to the energy efficiency of the cloud computing environment optimized according to relevant variables and constraints (Section III). Section V discusses the simulation design and metrics. Statistical variations provide us with the complexity we are interested in modeling, while we monitor the performance and its improvement using various evaluation metrics and present the results of our experiments (Section VI). Section VII concludes.

## II. QUEUEING THEORY IN CLOUD MONITOR

According to queueing theory, the utilization factor is $\rho = \frac{\lambda}{n\mu}$ where $\lambda$ is the arrival rate and $n\mu$ is the transmission rate in packets per second with $n$ channels. Therefore, $\lambda = 1/t_b$ for $t_b$ represents the time between jobs and $\mu = 1/t_\mu$ for $t_\mu$ represents the average duration of the jobs. A queueing model

with $n$ servers, Poisson arrivals, and exponential service times is referred to as M/M/m and the relevant equations are [20]:

$$\begin{aligned}
\rho &= \frac{\lambda}{n\mu} \text{ utilization} \\
\rho &< 1 \text{stability condition [21]} \\
\pi_k &= \begin{cases} \pi_0 \prod_{i=0}^{k-1} \frac{\lambda}{(i+1)\mu} k \le n \\ \pi_0 \prod_{i=0}^{n-1} \frac{\lambda}{(i+1)\mu} \prod_{i=n}^{k-1} \frac{\lambda}{n\mu} k > n \end{cases} \text{steady-state prob.} \\
\pi_k &= \begin{cases} \pi_0 \left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!} \text{for } k \le n \\ \pi_0 \left(\frac{\lambda}{\mu}\right)^k \frac{1}{n!\, n^{k-n}} \text{for } k > n \end{cases} \text{steady-state prob.} \\
1 &= \pi_0 \left[\sum_{k=0}^{n-1} \left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!} + \left(\frac{\lambda}{\mu}\right)^n \frac{1}{n!(1-\rho)}\right] \text{steady-state prob.} \\
p_Q &= \sum_{k=n}^{\infty} \pi_k = \frac{\pi_0 (n\rho)^n}{n!(1-\rho)} \quad \text{prob. of waiting in the queue}
\end{aligned} \tag{1}$$

Poisson distribution is used to model the probability of independent events within an interval with a known average arrival rate, $\lambda$[1]:

$$P(k \text{ jobs in 1 minute}) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{2}$$

We used a job postponement model that postpones non-allocated jobs by $\frac{3\sqrt{i}}{\lambda}$ for $0 \le i \le n^2$ postponements:

$$\begin{aligned}
\int_{i-1}^{i} \sqrt{x}dx \le \sqrt{k} \le \int_{i}^{i+1} \sqrt{x}dx \\
\int_{0}^{n^2-1} \sqrt{x}dx \le \sum_{i=1}^{n^2} \sqrt{i} \le \int_{1}^{n^2} \sqrt{x}dx \\
\frac{2(n^2-1)^{3/2}}{3} \le \sum_{i=1}^{n^2} \sqrt{i} \le \frac{2n^3}{3} \\
t_{\text{wait}} \simeq \frac{1}{\lambda} \frac{4n^3}{n^2+1} \quad \text{average waiting time} \\
T \simeq \frac{1}{\mu} + \frac{1}{\lambda} \frac{4n^3}{n^2+1} \quad \text{average task time} \\
T = \frac{k}{\lambda} \quad \text{Little's theorem} \\
k \simeq \frac{\lambda}{\mu} + \frac{4n^3}{n^2+1} \quad \text{\# jobs in the system} \\
k_Q \simeq \frac{4n^3}{n^2+1} \quad \text{\# jobs in the queue}
\end{aligned} \tag{3}$$

For each task, the number of feasible servers changes based on the capacities of the servers and requirements of the tasks.

[1] https://en.wikipedia.org/wiki/Poisson_distribution

Owing to this dynamism, the theoretical results can both over- and under-estimate T and k. We also recorded the expected experimental queueing statistics based on the averages of the random variable values and actual queueing statistics based on the throughput of the system. For instance, when $n_i$ stores the number of feasible servers for task $i$, the expected cloud queue utilization and waiting time are determined as follows:

$$\mathrm{E}[\rho] = \left(\sum_{i=1}^{m} \frac{\lambda_i}{n_i \mu_i}\right) / m \tag{4}$$

$$t_{\text{wait},i} \simeq \frac{2}{\lambda_i} \frac{(n_i^2-1)^{3/2} + n_i^3}{n_i^2+1} \tag{5}$$

$$\mathrm{E}[t_{\text{wait},i}] \simeq \left(\sum_{i=1}^{m} t_{\text{wait},i}\right) / m \tag{6}$$

## III. OPTIMIZING CLOUD COMPUTING LOADS

Optimization of the job load on the cloud and related scheduling decisions regarding where to allocate jobs and which server to empty to improve efficiency can be solved with integer linear programming [4]. Motivated by similar goals, we define and identify relevant variables and a set of constraints for our cloud computing environment for energy efficiency [22]. We maintain the number of threads ($p$), memory ($m$), disk size ($d$), and watts ($w$) of jobs and servers. The relevant variables are as follows:

| | |
|---|---|
| $n$ | number of servers |
| $x_u[j] = 1$ | server $j$ is used |
| $x_a[i,j] = 1$ | job $i$ is in server $j$ (distribution, allocation) |
| $x_m[i,j,k] = 1$ | job $i$ leaves server $j$ to go to server $k$ (redistribution, moves) |
| $p_{j_i}, m_{j_i}, d_{j_i}, w_{j_i}$ | consumption of job $j_i$ |
| $p_j, m_j, d_j, w_j, c_j$ | servings of server $j$ |
| $\hat{p}_j, \hat{m}_j, \hat{d}_j, \hat{w}_j$ | currently consumed resources of server $j$ |
| $f_w(i,j,k)$ | watt estimate of moving job $i$ from $j$ to $k$ |
| $f_t(i,j,k)$ | time estimate of moving job $i$ from $j$ to $k$ in seconds |
| $f_t(i)$ | estimated remaining runtime of job $i$ in seconds |

### A. SCHEDULING JOBS

Table 2 lists the constraints used for the optimal distribution and optimal redistribution of jobs to the computing nodes. A computation center with heterogeneous computers might prefer minimizing the total energy consumption, which can incentivize distribution and redistribution towards nodes that are more energy efficient and a more homogeneous computational center might prefer maximizing idle energy consumption so that more nodes will be running empty. Linear programming was used for optimization. We are interested in learning the performance gains from the redistribution. Figere 1 present examples of redistributions and the distribution of the jobs to servers before and after.

**TABLE 2. Constraints used during optimization.**

| | |
|---|---|
| Distribution | |
| $\min \sum_{j=1}^{n} w_j x_u[j]$ | minimize energy used by the servers |
| s.t. $\sum_{j=1}^{n} x_a[i,j] = 1$ | each job $i$ is assigned to a unique server. |
| $\sum_{i} p_{j_i} x_a[i,j] \leq (p_j - \hat{p}_j)$ | allocated server $j$ has space for processing |
| $\sum_{i} m_{j_i} x_a[i,j] \leq (m_j - \hat{m}_j)$ | allocated server $j$ has space for memory |
| Redistribution | |
| $\min \sum_{i,k} x_m[i,j,k](w_k - w_j)$ | minimizing energy consumption. |
| s.t. $\sum_{j=1, j\neq k}^{n} x_m[i,j,k] \leq 1$ | job $i$ can leave only once. |
| $x_m[i,j,k] + x_m[i',k,l] \leq 1$ | . . . and jobs in the target cannot leave. |
| $\sum_{i} p_{j_i} x_m[i,j,k] \leq (p_k - \hat{p}_k)$ | reassigned $k$ has space for processing |
| $\sum_{i} m_{j_i} x_m[i,j,k] \leq (m_k - \hat{m}_k)$ | reassigned server $k$ has space for memory |
| $\sum_{i} \sum_{k} x_m[i,j,k] = \sum_{i} x_a[i,j]$ | all jobs leave a redistribution source |
| $f_t(i) \geq f_t(i,j,k)$ | leaving job's remaining runtime is larger |
| $x_m[i,j,k] f_t(i)(w_k - w_j) + f_w(i,j,k) \leq 0$ | overall energy costs decrease |

### B. OPTIMIZATION COST FUNCTIONS

We consider the power consumption for both the distribution and redistribution cost functions. If we use only the power consumption of servers when selecting which server to allocate to, our linear programming solution fails when the available server has the highest power consumption for two reasons:

- server processing and memory capacities enter into the optimization only as constraints to satisfy
- the result from linear programming can contain partial distributions to satisfy the constraints.

We would like to obtain a cost function that considers these constraints. The variables used are:

| | |
|---|---|
| $p_v$ | $p_v = [p_{v_1}, p_{v_2}, \ldots, p_{v_m}]^T$ for VM threads |
| $\mathbf{p}$ | $p = [p_1 - \hat{p_1}, p_2 - \hat{p_2}, \ldots, p_n - \hat{p_n}]^T$ for server threads |
| $m_v$ | $m_v = [m_{v_1}, m_{v_2}, \ldots, m_{v_m}]^T$ for VM memories |
| $\mathbf{m}$ | $m = [m_1 - \hat{m_1}, m_2 - \hat{m_2}, \ldots, m_n - \hat{m_n}]^T$ for server memories |
| $\texttt{cpw}(c,w) = \frac{c}{w}$ | performance per watt is the computation performance per watt |
| $\texttt{wpc}(c,w) = \frac{w}{c}$ | inverse of $\texttt{cpw}(c,w)$ |
| $\mathbf{c}$ | $c_j = \texttt{wpc}(\hat{c_j}, \hat{w_j})$ |

The cost we optimize considers the watts per computation, the processors used, and the memory used by the tasks.

Equation (7) attempts to fit to the server that has the most space while also considering the energy used.

$$c' = c\Big( \max\left(\texttt{sign}(p_v - p), \texttt{sign}(m_v - m)\right) \mid p_v - p \mid \mid m_v - m \mid \Big) \tag{7}$$

Equation (8) attempts to fit to the servers that have the best fit while also considering the energy used.

$$c' = c\Big( \max\left(\texttt{sign}(p_v - p), \texttt{sign}(m_v - m)\right) \mid f(p_v - p) f(m_v - m) \mid \Big) \tag{8}$$

With Equation (8), we consider $\epsilon$-sensitive sigmoid loss, which is in $[-1, 1]$:

$$f(x) = \begin{cases} 2\sigma(x) + 1 + \epsilon & \text{for } x \geq \epsilon \\ 1 & \text{for } |x| \leq \epsilon \\ 1 + \epsilon/2 & \text{otherwise} \end{cases} \tag{9}$$

where $\epsilon > 0$. A graphical representation is shown in Figure 2. We want the loss to be greater than $\epsilon$ outside the $\epsilon$-insensitive region for continuity:

$$2\left(\frac{1}{1 + e^{-x}} - 0.5\right) \geq \epsilon \tag{10}$$

$$x \geq \log(1 + \epsilon) - \log(1 - \epsilon) \tag{11}$$

Since $x \geq \epsilon \geq \log(1 + \epsilon) - \log(1 - \epsilon)$, our loss function is always greater than $\epsilon$ outside the $\epsilon$-insensitive region.

## IV. LINEAR PROGRAMMING FORMULATION

Linear programming is a convex programming model that is in $O(n^2 m)$ for $m$ parameters for the constraints of vector dimensions of size $n$ with the following optimization [23]:

$$\text{minimize} c^T x$$
$$\text{s.t.} a_i^T x \leq b_i, i = 1, \ldots, m$$

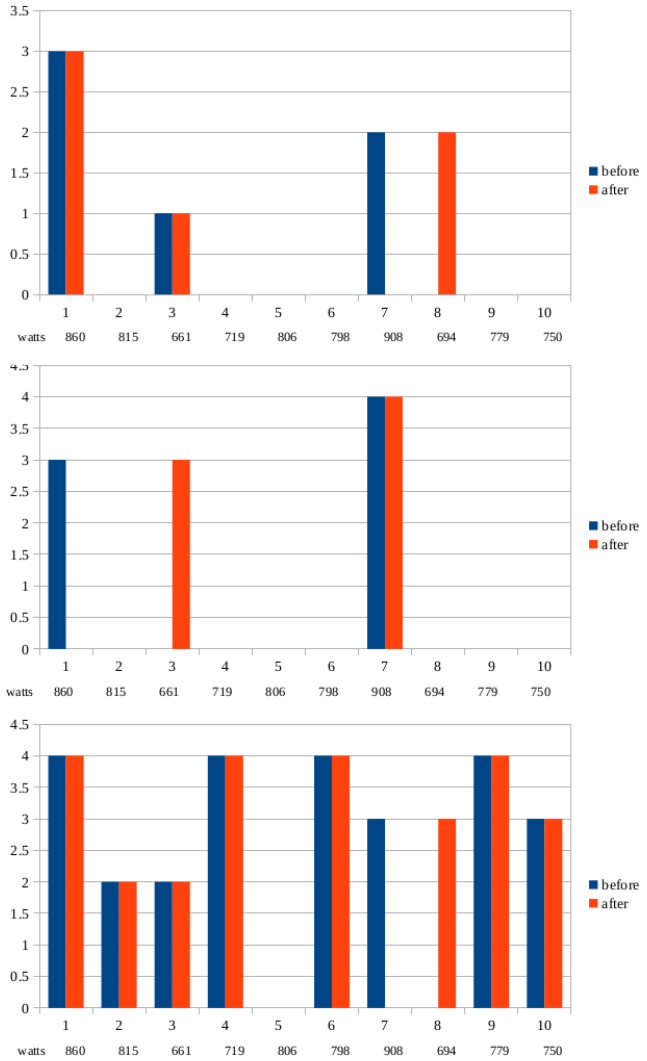


**FIGURE 1. Examples of redistribution and distribution before and after. x-axis is the number of CPUs used for each task and y-axis represents the servers.**

where $c, a_1, \ldots, a_m \in \mathbb{R}^n$ and $b_1, \ldots, b_m \in \mathbb{R}$ are program parameters.

The variables used are in Table 3.

$x_{mn}$ is initialized as $x_{mn} = \mathbf{1}_n \otimes \mathbf{0}_m$. $\texttt{vec}(.)$ produces vectorization of the argument by stacking its columns. $\otimes$ is the Kronecker product where $A \otimes B \in \mathbb{R}^{mp \times nq}$ for $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times p}$.

### A. OPTIMAL DISTRIBUTION

Optimal distribution is found with the linear program:

$$\begin{aligned} &\text{minimize} \quad c_{mn}^T x_{mn} \\ &\text{s.t.} \quad (D_{n,n} \otimes \mathbf{1}_m) x_{mn} = \mathbf{1}_n \\ &\qquad (D_{n,n} \otimes p_v^T) x_{mn} \le p \\ &\qquad (D_{n,n} \otimes m_v^T) x_{mn} \le m \\ &\qquad -(D_{n,n} \otimes \mathbf{1}_m) x_{mn} \le -\left\lceil \frac{\sum_{i=1}^{m} \hat{p}_j}{\max_j p_j} \right\rceil \end{aligned} \tag{12}$$

The total computational cost is in $(mn)^2 4n$. Linear programming does not necessarily provide integer distributions. Therefore, we run the optimization once to identify the possible servers to allocate to and then test positive entries for feasibility according to the distribution constraints to pick the minimum cost solution.

### B. OPTIMAL REDISTRIBUTION

- The optimal redistribution finds $x_r$ where $x_r[n\,i + j]$ indicates that job $i$ is redistributed to server $j$ where the relevant equations are presented in Table 4.
  Optimal redistribution variables are listed below:

$\left\lceil \diagdown_a \right\rceil = \texttt{diag}(a)$

$\left\lceil \diagdown_{\mathbf{1}_m^T x_0^T} \right\rceil^{-1}$ → inverse of the current job distribution to servers

$\mathbf{1}_n^0$ → currently unused servers

$\left\lceil \diagdown_{\mathbf{1}_n^0} \right\rceil$ → diagonal

$\overrightarrow{\mathrm{v}}(.)$ → function that vectorizes the argument

$\underline{x_0} = \left\lceil \diagdown_{\mathbf{1}_m^T x_0^T} \right\rceil^{-1} x_0$ → normalized initial distribution

$t_{n\max} = \left(x_0 \left\lceil \diagdown_{t_m} \right\rceil\right)_{\max^-}$
→ set as the maximum of servers (row, $\max^-$)

$\underline{t_m}_{\max} = \left(\left\lceil \diagdown_{\mathbf{1}_m^T x_0^T} \right\rceil^{-1} \left\lceil \diagdown_{t_{n\max}} \right\rceil x_0\right)_{\max^|}$
→ set as the maximum at the allocated server (column, $\max^|$)

$x_{0,\underline{t_m}_{\max}} = \left\lceil \diagdown_{\mathbf{1}_m^T x_0^T} \right\rceil^{-1} \left\lceil \diagdown_{t_{n\max}} \right\rceil x_0$
→ set as the allocation with maximums

$(D_m \otimes \mathbf{1}_n^T)\, x_r$ → jobs that migrate

$(\mathbf{1}_m^T \otimes D_n)\, x_r$ → servers that are redistributed to

$x_0\, (D_m \otimes \mathbf{1}_n^T)\, x_r$
→ number of redistributing jobs at servers where jobs migrate

$x_0\, (\mathbf{1}_m^T \otimes D_n)\, x_r$
→ jobs at those servers that are redistributed to

$T_S = x_{0,\underline{t_m}_{\max}} (D_m \otimes \mathbf{1}_n^T)$
→ time at emptied servers, S for source

$T_T = \underline{t_m^T}_{\max} \otimes \left\lceil \diagdown_{\mathbf{1}_n^0} \right\rceil$
→ time at empty moved to servers, T for target

The relevant constraints (e.g. if a job is redistributed, the target server cannot be emptied) are presented in Table 5. Explanations:

- The constraint in Equation (15) is for redistribution from a single server.
- The constraint in Equation (16) is more restrictive than that in Equation (17) because a single server

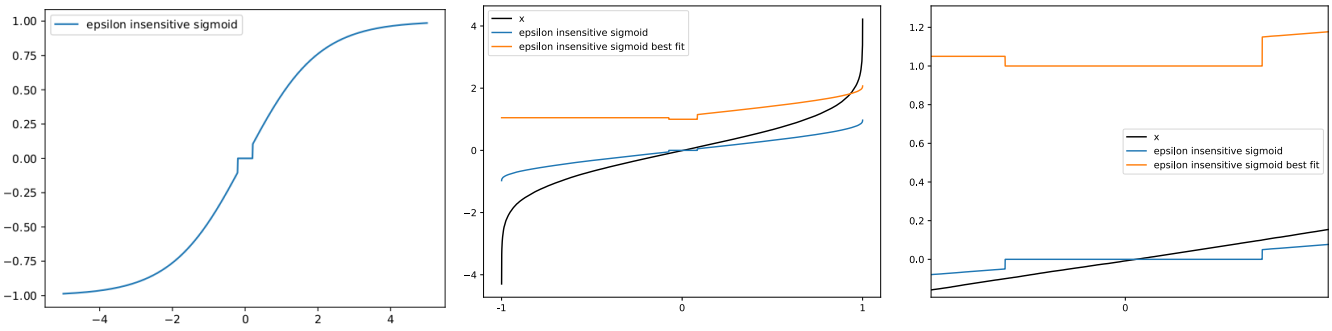


**FIGURE 2.** $\epsilon$-sensitive sigmoid loss for spacey-fit (top) and best-fit cost function distribution. y-axis is $f(x)$ and x-axis is $x$. The last plot zooms.

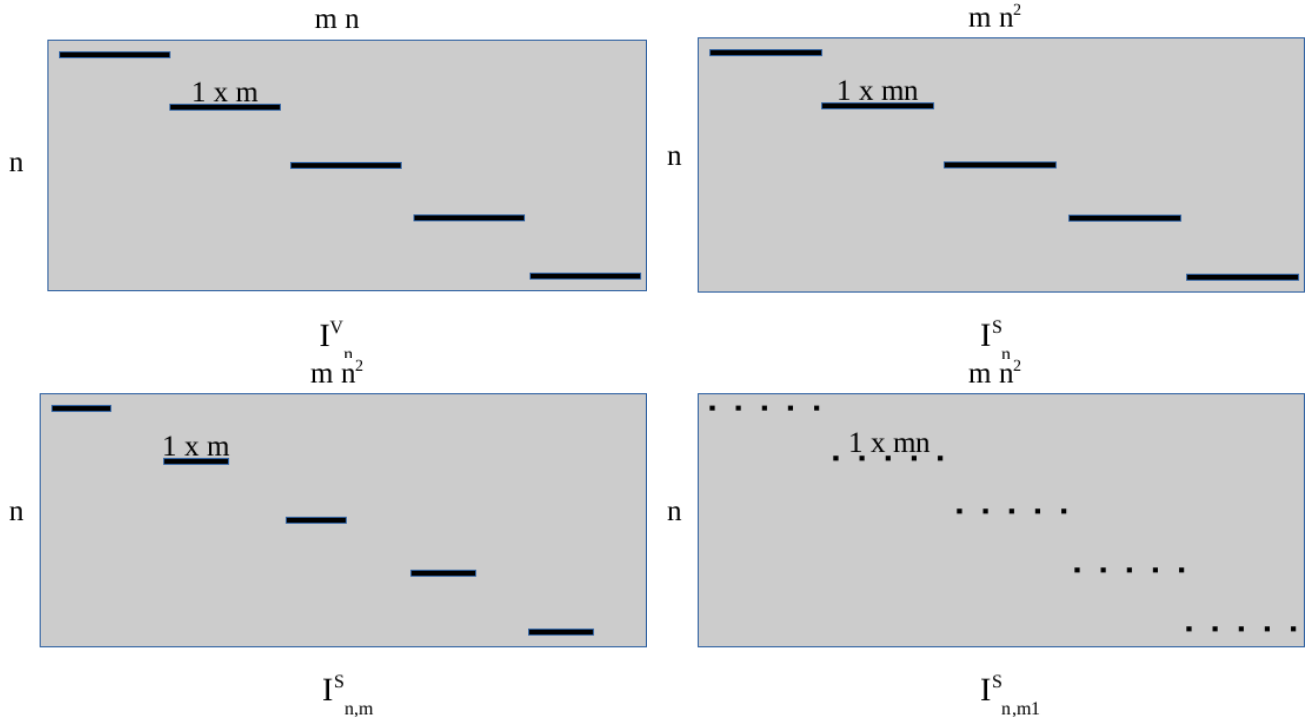


**FIGURE 3.** Indicator matrices: (top left) $I^V_{n\times mn} \in \mathbb{R}^{n\times mn} = D_{n,n} \otimes \mathbf{1}_m$, (top right) $I^S_{n\times mn^2} \in \mathbb{R}^{n\times mn^2} = D_{n,n} \otimes (\mathbf{1}_n \otimes \mathbf{1}_m)$, (bottom left) $I^S_{n\times mn^2,m} \in \mathbb{R}^{n\times mn^2} = D_{n,n} \otimes (b^1_n \otimes \mathbf{1}_m)$, (bottom right) $I^S_{n\times mn^2,m1} \in \mathbb{R}^{n\times mn^2} = \mathbf{1}_n \otimes (b^1_n \otimes D_{m,m})$.

**TABLE 3.** Linear programming variables.

| | |
|---|---|
| $\mathbf{x}_{mn}$ | allocation vector, $\mathbf{x}_{mn} = \mathrm{vec}(\mathbf{x}) = [x_{1,1}, x_{2,1}, \ldots, x_{m,1}, x_{1,2}, \ldots, x_{m,n}]^T \in \mathbb{R}^{mn\times 1}$ |
| $\mathbf{x}_{mn^2}$ | allocation vector, $\mathbf{x}_{mn^2} = \mathrm{vec}(\mathbf{x}) = [x_{1,1,1}, x_{2,1,1}, \ldots, x_{m,1,1}, x_{1,2,1}, \ldots, x_{m,n,n}]^T \in \mathbb{R}^{mn^2\times 1}$. |
| $I^V_{n\times mn}$, $I^S_{n\times mn^2}$ | indicator matrices (Figure 3) |
| $\mathbf{c}_{mn}$ | $\mathbf{c}_{mn} = \mathbf{1}_m \otimes [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n]$ where $\mathbf{c}_j = \mathrm{wpc}(\hat{c}_j, \hat{w}_j)$ |
| $\mathbf{c}_{mn^2,w}$ | contains costs for migration in watts computed using $f_w(.)$ |
| $\mathbf{c}_{mn^2,t}$ | contains time costs for migration computed using $f_t(.)$ |
| $\mathbf{c}_{mn^2,t_c}$ | contains remaining computation time costs computed using $f_t(.)$ |
| $\mathbf{\Omega}_n$ | $\mathbf{\Omega}_n = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n]$ |
| $\mathbf{t}_m$ | contains remaining computation time costs computed using $f_t(.)$ |
| $\mathbf{t}_{m\times n}$ | contains costs for redistribution time computed using $f_t(.)$ [2] |
| $\mathbf{\Omega}_{m\times n}$ | contains costs for redistribution watts computed using $f_w(.)$ [2] |

$j$ can affect up to $p_j$ jobs. However, when we want $k > 1$ redistribution sources, we use also Equation (16).

– Partial cost computation can also be considered to model the cost of moving a job to a running server.

**TABLE 4.** Optimal redistribution equations.

$$\text{minimize}\left(\underbrace{\overrightarrow{\nabla}(\mathbf{\Omega}_{m\times n})}_{\text{moving watts}}+\right.$$

$$\left.\mathbf{\Omega}_n^T\left(\underbrace{\mathbf{t}_{m\max}^T\otimes\left\lceil \mathbf{1}_n^0\right\rfloor}_{\text{finish time at empty moved to servers = } T_T}-\underbrace{\mathbf{x}_{0,\mathbf{t}_{m\max}}(D_m\otimes\mathbf{1}_n^T)}_{\text{finish time at emptied servers = } T_S}\right)\mathbf{x}_r\right. \tag{13}$$

$$\begin{aligned}
\text{s.t.}&(D_m\otimes\mathbf{1}_n^T)\,\mathbf{x}_r\le\mathbf{1}_m && \text{job } i \text{ can leave only once}\\
&(\mathbf{p}_v^T\otimes D_n)\,\mathbf{x}_r\le\mathbf{p} && \text{server has space for processing}\\
&(\mathbf{m}_v^T\otimes D_n)\,\mathbf{x}_r\le\mathbf{m} && \text{server has memory for processing}\\
&\frac{1}{\mathbf{1}_m^T\mathbf{x}_0^T}\mathbf{x}_0\,(D_m\otimes\mathbf{1}_n^T)\,\mathbf{x}_r\le 1 && \text{redistributing from a single server}\\
&\mathbf{x}_0\,(D_m\otimes\mathbf{1}_n^T)\,\mathbf{x}_r=\mathbf{x}_0\,\mathbf{1}_m && \text{all jobs are redistributed from S}\\
&\left((D_m\otimes\mathbf{1}_n^T)\times\mathbf{x}_0^T\,(\mathbf{1}_m^T\otimes D_n)\right)\mathbf{x}_r=\mathbf{0}_m && \text{jobs that move} \neq \text{jobs at T}\\
&\left((\mathbf{1}_m^T\otimes D_n)\times\mathbf{x}_0\,(D_m\otimes\mathbf{1}_n^T)\right)\mathbf{x}_r=\mathbf{0}_n && \text{S} \neq \text{T}\\
&\left(\overrightarrow{\nabla}(\mathbf{\Omega}_{m\times n})+\mathbf{\Omega}_n^T(T_T-T_S)\right)\mathbf{x}_r\le 0 && \text{redistribution watt constraint}\\
&\left(\underbrace{(D_m\otimes\mathbf{1}_n^T)\times\overrightarrow{\nabla}(\mathbf{\Omega}_{m\times n})}_{\text{moving watt}}+\underbrace{\left\lceil \mathbf{t}_m\right\rfloor\otimes\mathbf{\Omega}_n^T-\underline{\mathbf{x}_0}^T\left\lceil \mathbf{\Omega}_n\right\rfloor\mathbf{x}_0\left(\left\lceil \mathbf{t}_m\right\rfloor\otimes\mathbf{1}_n\right)}_{\text{finishing watt at T minus at S}}\right)\mathbf{x}_r\le\mathbf{0}_m && \text{redistribution watt constraint (setting 1 \& 2)}\\
&\left(\underbrace{(D_m\otimes\mathbf{1}_n^T)\times\overrightarrow{\nabla}(\mathbf{t}_{m\times n})}_{\text{moving time}}-\underbrace{\left\lceil \mathbf{t}_m\right\rfloor(D_m\otimes\mathbf{1}_n^T)}_{\text{finishing time}}\right)\mathbf{x}_r\le\mathbf{0}_m && \text{redistribution time constraint (setting 2)}
\end{aligned}$$

**TABLE 5.** Optimal redistribution constraints.

$$(D_m\otimes\mathbf{1}_n^T)\,\mathbf{x}_r\le\mathbf{1}_m \quad \text{jobs can migrate once} \tag{14}$$

$$\frac{1}{\mathbf{1}_m^T\mathbf{x}_0^T}\mathbf{x}_0\,(D_m\otimes\mathbf{1}_n^T)\,\mathbf{x}_r\le 1 \quad \text{redistributing from a single server} \tag{15}$$

$$\left((\mathbf{1}_m^T\otimes D_n)\times\mathbf{x}_0\,(D_m\otimes\mathbf{1}_n^T)\right)\mathbf{x}_r=\mathbf{0}_n \quad \text{servers that are moved to and from cannot be the same} \tag{16}$$

$$\left((D_m\otimes\mathbf{1}_n^T)\times\mathbf{x}_0\,(\mathbf{1}_m^T\otimes D_n)\right)\mathbf{x}_r=\mathbf{0}_m \quad \text{jobs cannot move and be at servers that are moved to} \tag{17}$$

$$\mathbf{x}_0\,(D_m\otimes\mathbf{1}_n^T)\,\mathbf{x}_r=\mathbf{x}_0\,\mathbf{1}_m \quad \text{number of jobs that migrate is equal to servers' distribution} \tag{18}$$

- The total computational cost is in $(mn)^2(5n+m+mn)$.
- Computational complexity of $A\otimes B$ where $A\in\mathbb{R}^{m,n}$ and $B\in\mathbb{R}^{p,r}$ is in $O(mnpr)$ therefore, we attempt to decrease $\otimes$ operations.
- Redistribution times are summed and finish times are maximized within all jobs of a server that is being redistributed.
- Minimization does not consider the energy cost of jobs moving to servers already running. This can be added in proportion.
- The normalization in $T_T=\underline{t_m}_{\max}^T\otimes\left\lceil \mathbf{1}_n^0\right\rfloor$ works because redistributions are restricted to source from a single server. $T_T=t_m^T\otimes\left\lceil \mathbf{1}_n^0\right\rfloor$ sums the finish times of the moving jobs at $T$. We are interested in their maximum because of their parallelism.

- The optimal redistribution of servers finds $x_r$ where $x_r[n\,i+j]$ indicates that server $i$ is redistributed to server $j$ where the relevant equations are presented in Table 4. The variables for the optimal redistribution of servers are as follows:

  $x_0'\rightarrow$ current allocation with only used $n_1$ servers

  $\left\lceil {}_{n_1\rightarrow n}\right\rfloor\rightarrow$ maps $n_1$ S servers to T servers

  $T_S=\left\lceil {}_{n_1\rightarrow n}\right\rfloor\left(\left\lceil {t_n}_{\max}{}^T\right\rfloor\otimes\mathbf{1}_n^T\right)$

  $\rightarrow$ time at emptied servers, S for source

  $T_T={t_n}_{\max}{}^T\otimes\left\lceil \mathbf{1}_n^0\right\rfloor$

  $\rightarrow$ time at empty moved to servers, T for target

**TABLE 6.** Optimal redistribution of servers.

$$\text{minimize} \quad \left( \underbrace{\overrightarrow{\nabla}(\mathbf{x}_0' \boldsymbol{\Omega}_{m \times n})}_{\text{moving watts}} + \boldsymbol{\Omega}_n^T (T_T - T_S) \right) \mathbf{x}_r \tag{19}$$

| | |
|---|---|
| s.t. $\mathbf{1}_n'^T (D_n' \otimes \mathbf{1}_n^T)\, \mathbf{x}_r \leq 1$ | server $i$ can leave only once and $\lvert S \rvert = 1$ |
| $\left\lfloor \mathbf{1}_n^0 \right\rfloor (D_n \otimes \mathbf{1}_n^T)\, \mathbf{x}_r \leq \mathbf{0}_n$ | unused servers cannot leave |
| $(\mathbf{p}'^T \otimes D_n)\, \mathbf{x}_r \leq \mathbf{p}$ | server has space for processing |
| $(\mathbf{m}'^T \otimes D_n)\, \mathbf{x}_r \leq \mathbf{m}$ | server has memory for processing |
| $\left( (\mathbf{1}_n^T \otimes D_n) \times (D_n \otimes \mathbf{1}_n^T) \right) \mathbf{x}_r = \mathbf{0}_n$ | S $\neq$ T |
| $\overrightarrow{\nabla}\big( \underbrace{\boldsymbol{\Omega}_n}_{\text{T watts}} - \underbrace{\boldsymbol{\Omega}_n' \otimes \mathbf{1}_n^T}_{\text{S watts}} \big) \mathbf{x}_r \leq 0$ | move to T only if $\Omega_T \leq \Omega_S$ |
| $\left( \overrightarrow{\nabla}(\mathbf{x}_0' \boldsymbol{\Omega}_{m \times n}) + \boldsymbol{\Omega}_n^T (T_T - T_S) \right) \mathbf{x}_r \leq 0$ | redistribution watt constraint |
| $\big( \underbrace{\mathbf{x}_0' \boldsymbol{\Omega}_{m \times n}}_{\text{moving watts}} + \underbrace{\left\lfloor \boldsymbol{\Omega}_n \right\rfloor (T_T - T_S)}_{\text{finishing watts}} \big) \mathbf{x}_r \leq \mathbf{0}_n$ | redistribution watt constraint (setting 1 & 2) |
| $\big( \underbrace{\mathbf{x}_0' \mathbf{t}_{m \times n}}_{\text{moving time}} + \underbrace{T_T - T_S}_{\text{finishing time}} \big) \mathbf{x}_r \leq \mathbf{0}_n$ | redistribution time constraint (setting 2) |

### C. OPTIMAL ELASTICITY

The optimal elasticity of jobs finds $x_r$ where $x_r[n\,i + j]$ indicates that job $i$ is redistributed to server $j$ where the relevant equations are given in Table 7.

Variables about optimal elasticity are:

$T_S = \underline{x_{0,t_m}}_{\max} (D_m \otimes \mathbf{1}_n^T)$ time at emptied servers
S for source

$T_T = \underline{t_m^T}_{\max} \otimes \left\lfloor \mathbf{1}_n^0 \right\rfloor$ time at empty moved to servers
T for target

Explanations:

- Only the jobs at those servers whose utilization values are above the service-level agreement (SLA) determined maximum are considered candidates for redistribution and only some of those jobs need to be redistributed.
- We do not have watts or time as constraints in the optimization however we minimize them.
- $\mathbf{a}$ is a scaling factor for each server, where $a_i = \alpha(2 - u_i) > 1$ for source servers is inversely proportional to their remaining utilization and 1 for others.
- As long as we redistribute from a single server, normalization of $T_T$ works.
- S $\neq$ T since the candidate pools for them are different.

## V. SIMULATION DESIGN AND METRICS

We kept statistics and logs about the cloud's performance.

Load utilization: processor and memory load proportion vs. processor and memory load in the queue. We also compared the difference with the load when using a uniform distribution.

Energy reduction: compared with uniform distribution, how much energy can we save with optimal distribution according to energy consumption for different SNR, signal to noise ratio, $SNR = \frac{\mu}{\sigma}$, scenarios for watt distribution of servers.

The SNR of the energy distribution and constraints affects whether redistribution is possible. Optimizing for watts per processing power also makes sense. For instance, this is higher for GPUs, which have recently been used more frequently for computing.

When the servers are full or $\rho > 1$, we see a meteor shower pattern $(\ldots - +)$ where a '.' represents no change, '-' represents job removal, and '+' represents job addition:

```
.....-+.....................................
...-+-+.....................-+.......
....-+.................-+.............
..............-+..........-+....-+.......
-+.........-+-+.................-+..
.....--++.....................-+.....
.................-+...................-+
........-+..........-+.................
.......-+...-+..
```

## VI. RESULTS AND EVALUATION

We stored the kilowatt energy used, average queue wait time, parallel computation and memory utilization, correlation of server watts with server load, queueing theory related utilization, and total time spent in four experimental settings:

- random distribution,

**TABLE 7.** Optimal elasticity equations.

$$\text{minimize} \quad \mathbf{a}\,\Omega_n^T \Bigg( \underbrace{\mathbf{t}_{m_{\max}}^T \otimes \left\lceil \mathbf{1}_n^0 \right\rceil}_{\text{finish time at empty moved to servers} = T_T} - \underbrace{\mathbf{x}_{0,\mathbf{t}_{m_{\max}}} (D_m \otimes \mathbf{1}_n^T)}_{\text{finish time at emptied servers} = T_S} \Bigg) \mathbf{x}_r \tag{20}$$

$$\text{s.t.} (D_m \otimes \mathbf{1}_n^T)\ \mathbf{x}_r \le \mathbf{1}_m \quad \text{job } i \text{ can leave only once}$$
$$(\mathbf{p}_v^T \otimes D_n)\ \mathbf{x}_r \le \mathbf{p} \quad \text{server has space for processing}$$
$$(\mathbf{m}_v^T \otimes D_n)\ \mathbf{x}_r \le \mathbf{m} \quad \text{server has memory for processing}$$
$$\frac{1}{\mathbf{1}_m^T \mathbf{x}_0^T} \mathbf{x}_0\ (D_m \otimes \mathbf{1}_n^T)\ \mathbf{x}_r \le 1 \quad \text{redistributing from a single server}$$

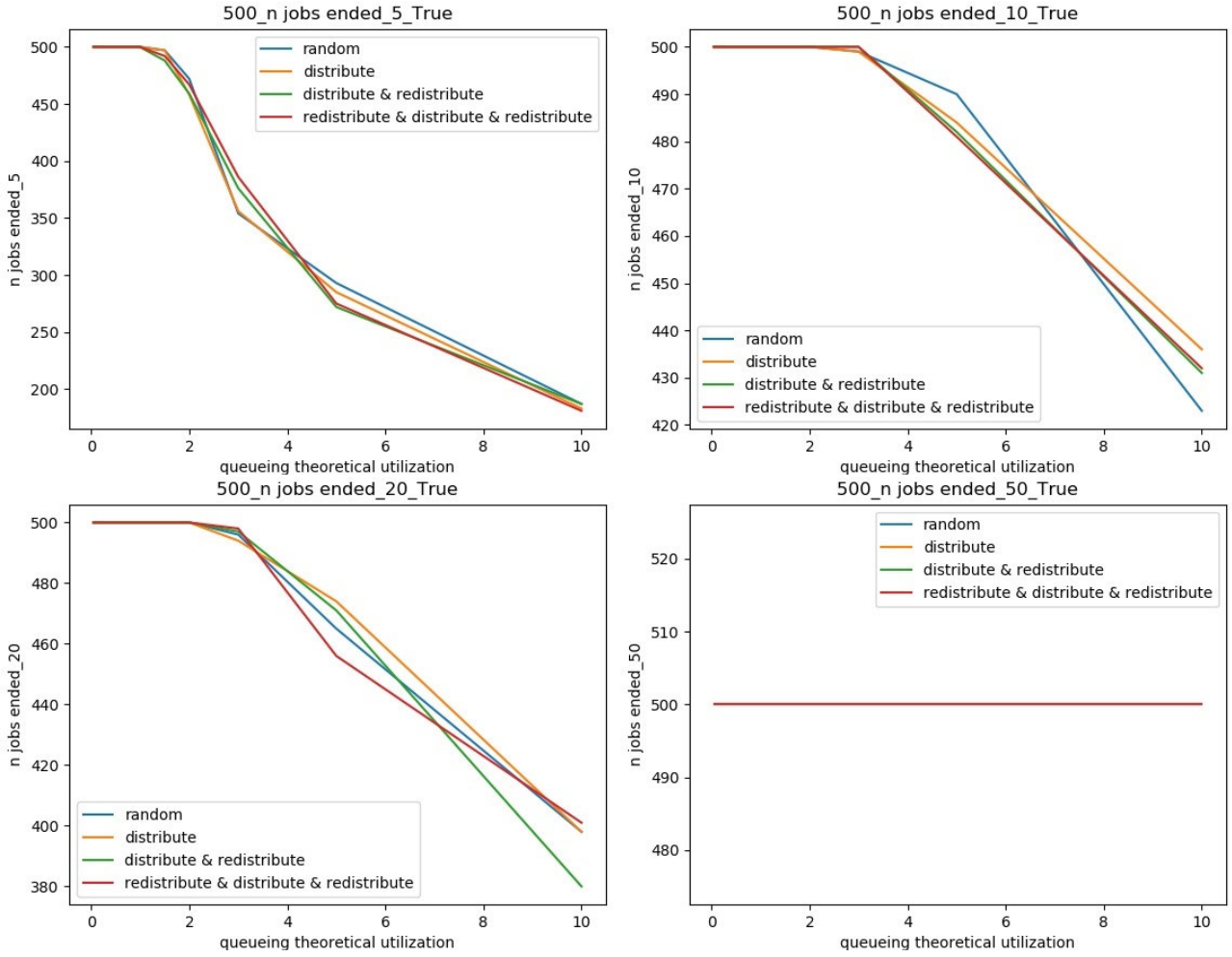


**FIGURE 4.** The number of jobs that end decrease with increasing multiplier $\rho$ when jobs are dropped after postponement limit.

- optimal distribution,
- optimal distribution and redistribution after jobs end,
- optimal distribution and redistribution after jobs end and before jobs are allocated.

The number of servers used can be 5, 10, 20, or 50. We selected queueing theoretical utilization, $\rho$, from [0.05, 0.15, 0.25, 0.5, 0.99, 1.5, 2, 3, 5, 10] as a multiplier to set $t_b = \frac{t_\mu}{n\rho}$, which for $\rho > 1$, owing to low $t_b$, increase the drop rate when we drop jobs after the postponement limit of $n^2$. The number of jobs that survived or ended during the simulation is shown in Figure 4. In the following results, we disabled dropping jobs to compare their performance with an equal number of jobs processed.

Our findings in simulations of 500 jobs are:

- Queueing theory actual utilization improves with increased number of servers and when the number

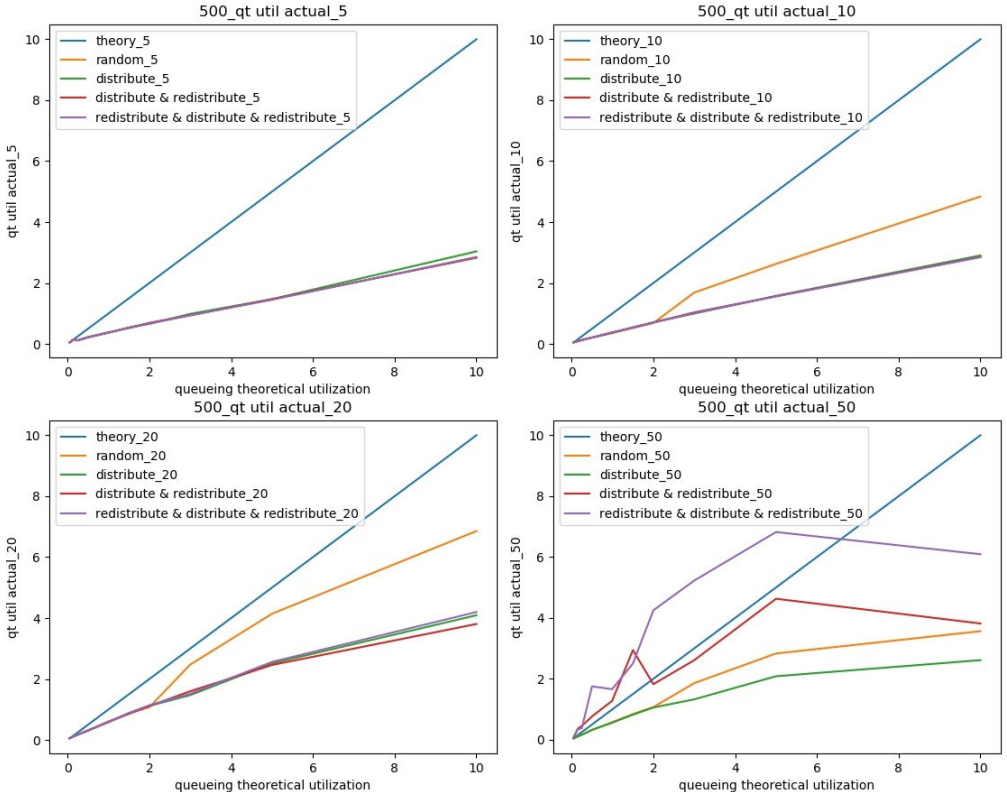


**FIGURE 5.** Queueing theory actual utilization improve with increasing number of servers.

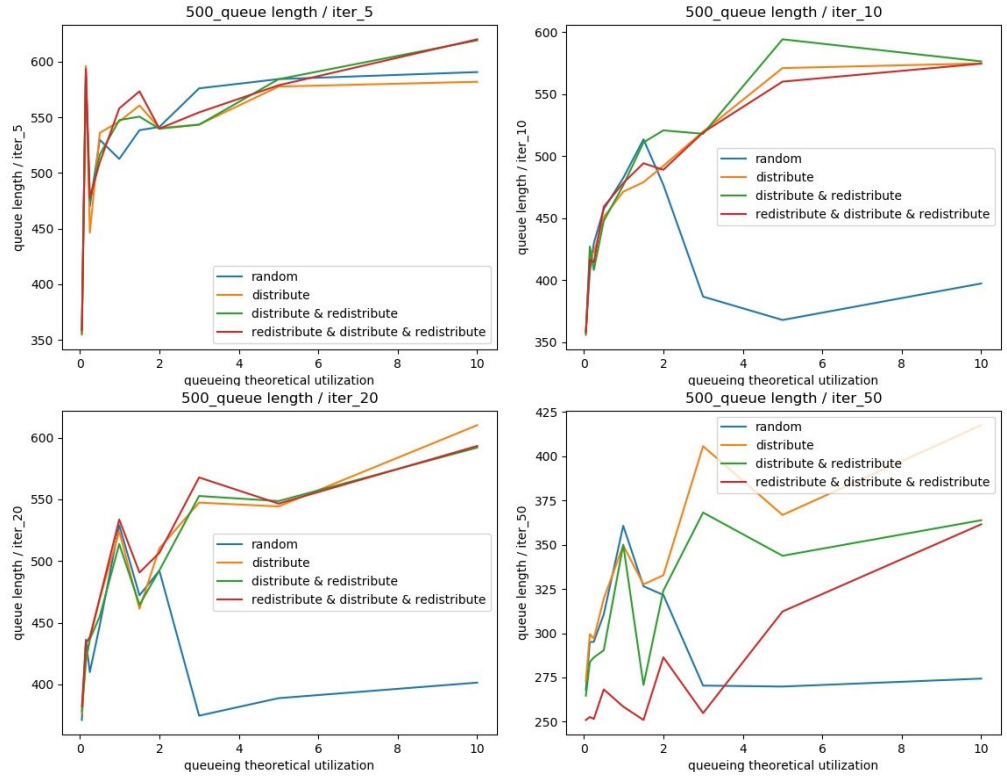


**FIGURE 6.** Average queue length decrease with increasing number of servers.

of servers is 50, the theoretical utilization is passed (Figure 5).

- The average queue length decreases with increasing number of servers and improves over random

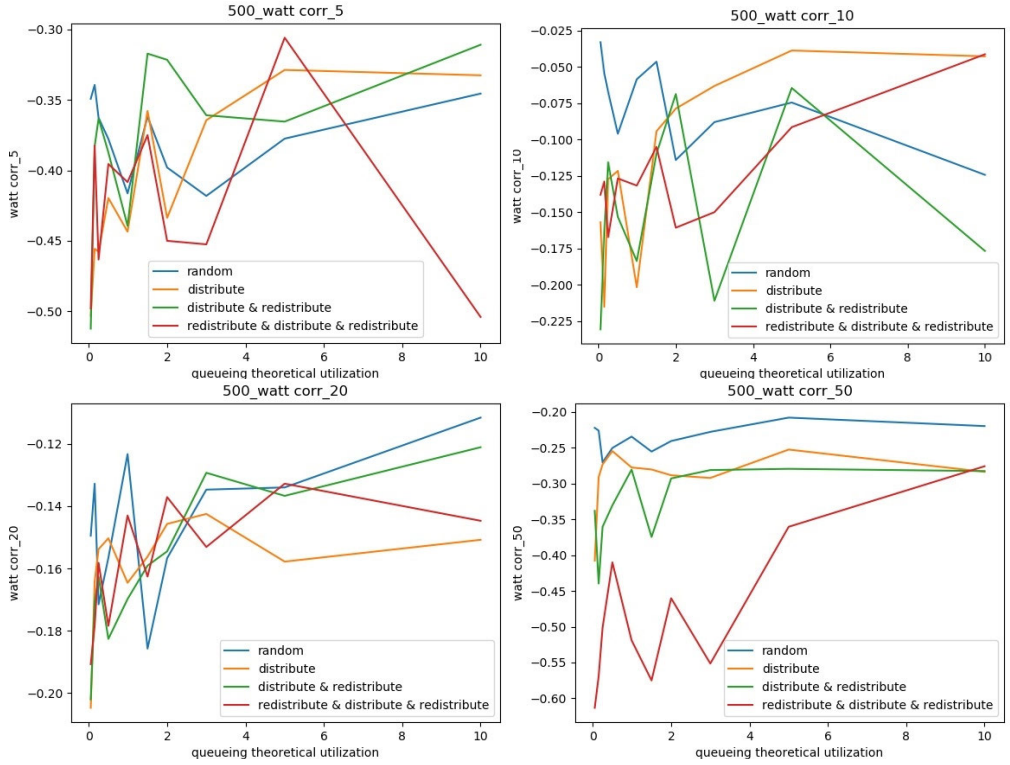


**FIGURE 7.** Correlation of the job distribution with watt of servers improve with increasing number of servers.

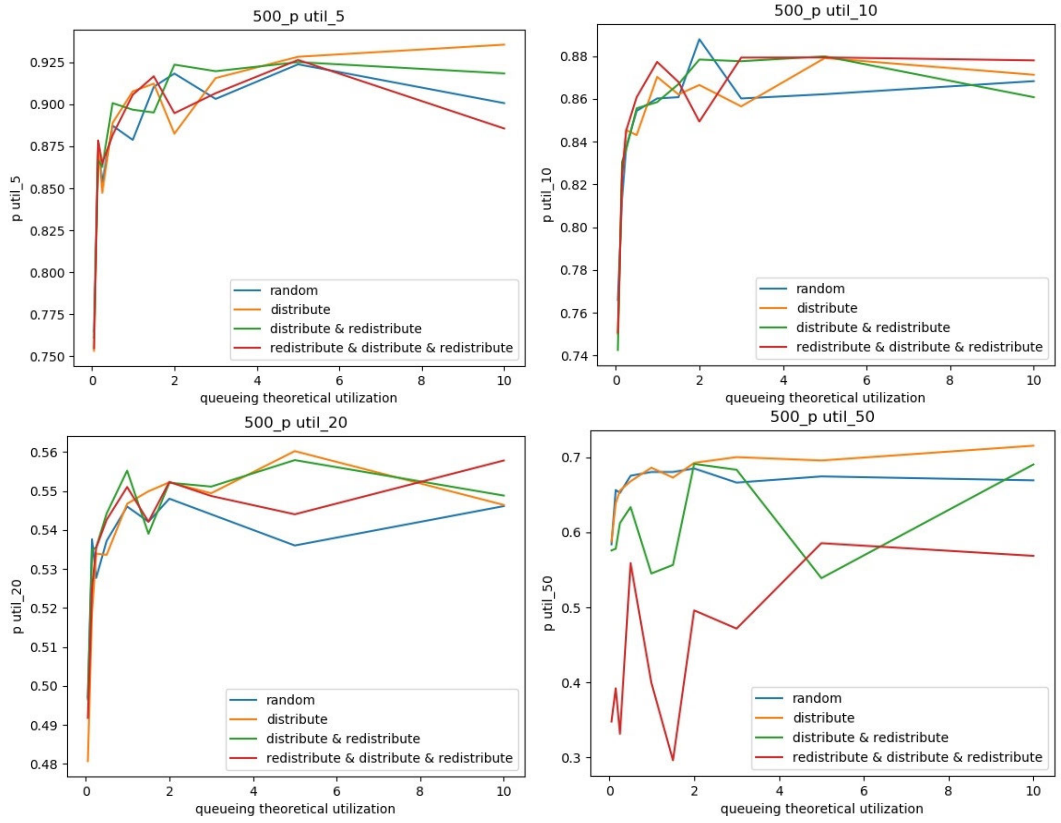


**FIGURE 8.** Utilization of parallel computation decrease with increasing number of servers.

distribution when the number of servers is 50 (Figure 6).

- The correlation of the job distribution with the watts of servers improves with increasing

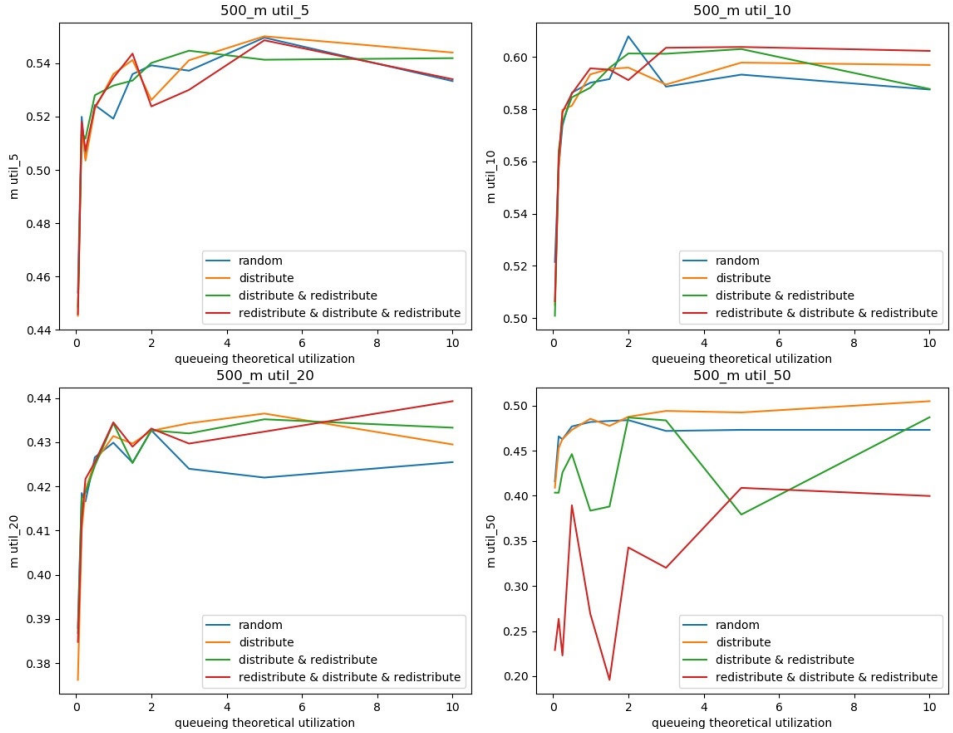


**FIGURE 9.** Utilization of memory stay around 0.5 with increasing number of servers.

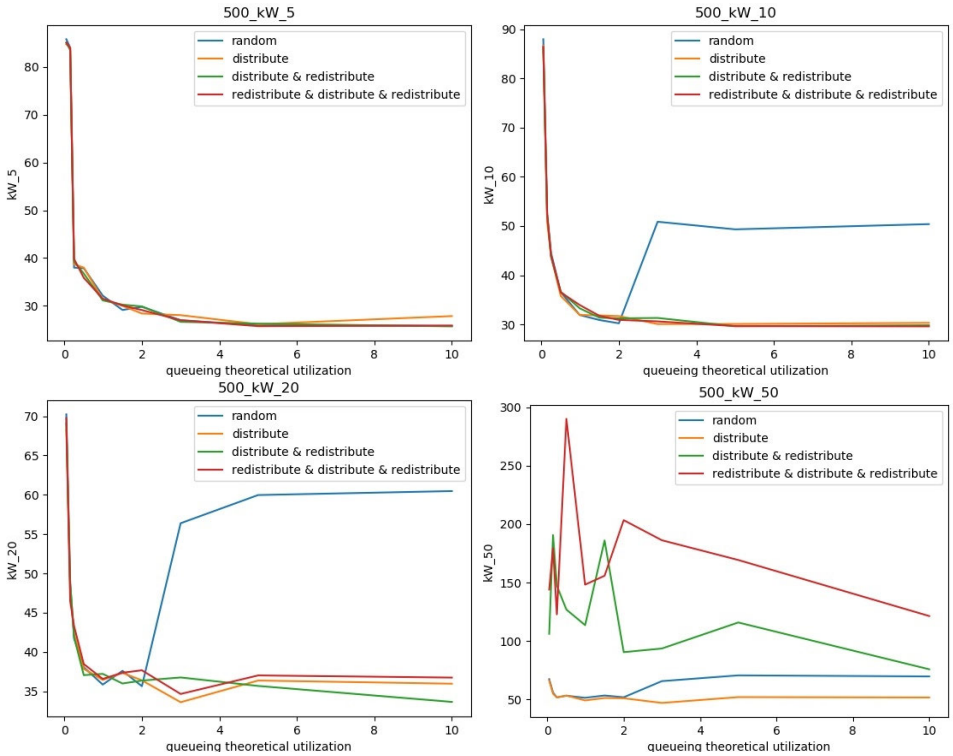


**FIGURE 10.** Energy used in kW is significantly reduced (up to 42% with 20 servers) compared with random distribution.

number of servers (Figure 7). This correlation is an important indicator of reduction in the energy.

- Utilization of parallel computation decreases and of memory stays around 0.5 with increasing number of servers (Figure 8 and 9).

- The energy used in kW was significantly reduced (up to 42% with 20 servers) compared with the random distribution (Figure 10).
- The average waiting time in the queue during the simulation exhibited a trend similar to that of the kW used.
- Total simulation time show similar trend with the kW used.

These findings demonstrate the effectiveness of the cloud monitor in optimizing job distribution, reducing energy consumption, and improving resource utilization in cloud computing environments. Overall, the results of the simulations provide evidence of the cloud monitor's ability to achieve significant reductions in energy consumption and improvements in resource utilization.

## VII. CONCLUSION

Cloud monitor is an effort for better job distribution models on cloud computing platforms to reduce energy consumption, decrease computation and waiting time in the queue, and improve the overall utilization of resources. We present our linear programming formulation and the relevant equations. Our simulations show (i) improvements in job distributions with better correlation with the energy consumption of servers, (ii) possible increased utilization of resources in some settings, and (iii) significant reductions in energy consumption that reach 42%. We provide important results for the optimization of cloud computing platforms for energy efficiency and resource utilization, which contribute to the development of more sustainable and efficient cloud computing systems.

## ACKNOWLEDGMENT

The author did not use any AI generated text. The work was conducted while the author was employed by TÜBİTAK.

## REFERENCES

[1] G. Kamiya. (2022). *Data Centres and Data Transmission Networks, IEA*. [Online]. Available: https://www.iea.org/reports/data-centres-and-data-transmission-networks

[2] A. Andrae and T. Edler, "On global electricity usage of communication technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, Apr. 2015.

[3] M. Maximilien, D. Hadas, A. Danducci II, and S. Moser. (Oct. 2022). *The Future is Serverless*. [Online]. Available: https://developer.ibm.com/blogs/the-future-is-serverless/

[4] C. Ghribi, "Energy efficient resource allocation in cloud computing environments," Ph.D. thesis, Inst. Nat. des Telecommun., Paris, France, 2014.

[5] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proc. 10th IEEE/ACM Int. Conf. Clust., Cloud Grid Comput.*, 2010, pp. 826–831.

[6] S. Rani, D. Kumar, and S. Dhingra, "A review on dynamic load balancing algorithms," in *Proc. Int. Conf. Comput., Commun., Intell. Syst. (ICCCIS)*, Nov. 2022, pp. 515–520.

[7] V. Borovskiy, J. Wust, C. Schwarz, W. Koch, and P. D. A. Zeier, "A linear programming approach for optimizing workload distribution in a cloud," in *Proc. 2nd Int. Conf. Cloud Comput., GRIDs*, Sep. 2011, pp. 127–132.

[8] L. Liu and Q. Fan, "Resource allocation optimization based on mixed integer linear programming in the multi-cloudlet environment," *IEEE Access*, vol. 6, pp. 24533–24542, 2018.

[9] Y. Jararweh, L. Tawalbeh, F. Ababneh, A. Khreishah, and F. Dosari, "Scalable cloudlet-based mobile computing model," *Proc. Comput. Sci.*, vol. 34, pp. 434–441, Jan. 2014.

[10] T. Khan, W. Tian, S. Ilager, and R. Buyya, "Workload forecasting and energy state estimation in cloud data centres: ML-centric approach," *Future Gener. Comput. Syst.*, vol. 128, pp. 320–332, Mar. 2022.

[11] G. Tang, W. Jiang, Z. Xu, F. Liu, and K. Wu, "NIPD: Non-intrusive power disaggregation in legacy datacenters," *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 312–325, Feb. 2017.

[12] Die.net. (2023). *Linux Man Page: Dstat*. [Online]. Available: http://linux.die.net/man/1/dstat

[13] F. Liu, Z. Zhou, H. Jin, B. Li, B. Li, and H. Jiang, "On arbitrating the power-performance tradeoff in SaaS clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 10, pp. 2648–2658, Oct. 2014.

[14] W. Deng, F. Liu, H. Jin, X. Liao, and H. Liu, "Reliability-aware server consolidation for balancing energy-lifetime tradeoff in virtualized cloud datacenters," *Int. J. Commun. Syst.*, vol. 27, no. 4, pp. 623–642, Apr. 2014.

[15] A. Javadpour, A. K. Sangaiah, P. Pinto, F. Ja'fari, W. Zhang, A. M. H. Abadi, and H. Ahmadi, "An energy-optimized embedded load balancing using DVFS computing in cloud data centers," *Comput. Commun.*, vol. 197, pp. 255–266, Jan. 2023.

[16] M. Nazeri and R. Khorsand, "Energy aware resource provisioning for multi-criteria scheduling in cloud computing," *Cybern. Syst.*, pp. 1–30, May 2022.

[17] A. Moazeni, R. Khorsand, and M. Ramezanpour, "Dynamic resource allocation using an adaptive multi-objective teaching-learning based optimization algorithm in cloud," *IEEE Access*, vol. 11, pp. 23407–23419, 2023.

[18] R. Erdei and L. Toka, "Minimizing resource allocation for cloud-native microservices," *J. Netw. Syst. Manag.*, vol. 31, no. 2, p. 35, Feb. 2023.

[19] P. Osypanka and P. Nawrocki, "Resource usage cost optimization in cloud computing using machine learning," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 2079–2089, Jul. 2022.

[20] D. P. Bertsekas and R. G. Gallager, *Data Networks*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1992.

[21] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation With Computer Science Applications*, 2nd ed. Hoboken, NJ, USA: Wiley, 2006.

[22] E. Biçici, "Enerji harcamalarýný azaltmak için bulut monitorü ('a cloud monitor for reducing energy consumption),"' in *Proc. 1st Symp. Cloud Comput. Big Data*, Antalya, Turkey, Oct. 2017, pp. 117–122.

[23] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

**ERGUN BIÇICI** received the B.S. degree from Bilkent University, in 2000, the M.S. degree from North Carolina State University, in 2002, and the Ph.D. degree in computer sciences and engineering from Koç University, Istanbul, Turkey, in 2011.

From 2012 to 2015, he was a Postdoctoral Researcher with the ADAPT Research Center, Dublin City University, Dublin, Ireland. From 2015 to 2016, he was a Research Scientist with DFKI, Germany. From 2017 to 2018, he was a Researcher with TÜBİTAK. From 2018 to 2021, he was a Research Scientist with Boğaziçi University, Istanbul, Turkey. Since 2022, he has been a Senior AI Research Engineer with the Huawei Turkey Research and Development Center. He is the author of more than 60 publications. His research interests include machine translation, machine learning, natural language processing, cloud computing, recommendation systems, and artificial intelligence.