IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 29, 2022, accepted March 2, 2022, date of publication March 8, 2022, date of current version March 11, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3157435

A Fair, Dynamic Load Balanced Task Distribution Strategy for Heterogeneous Cloud Platforms Based on Markov Process Modeling

STAVROS SOURAVLAS[1,2], (Member, IEEE), SOFIA D. ANASTASIADOU[2], NICOLETA TANTALAKI[3], AND STEFANOS KATSAVOUNIS[4]

[1]Department of Applied Informatics, University of Macedonia, 54636 Thessaloniki, Greece
[2]Department of Midwifery, School of Health Sciences, University of Western Macedonia, 50200 Ptolemaida, Greece
[3]Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
[4]Department of Production and Management Engineering, Democritus University of Thrace, 69100 Xanthi, Greece

Corresponding author: Stavros Souravlas (sourstav@uom.edu.gr)

ABSTRACT Load balancing techniques in cloud computing can be applied at three different levels: Virtual machine load balancing, task load balancing, and resource load balancing. At all levels, load balancing should also be implemented in an efficient manner, to increase system performance. In this paper, we propose a fair, in terms of added workload per VM, task load balancing strategy, that aims to improve the average response time and the makespan of the system in the cloud environment. The problem is formulated as an irreducible finite state Markov process, which is known to have a balance equation for each state. From the balance state probabilities we derive the expected utilizations for the virtual machines (VM), which play a vital role in our task allocation approach. In our model, the Load Balancer (LBer) acts as a central server, which uses our proposed fair task allocation scheme to distribute the incoming tasks in a fair, balanced manner among the virtual machines, taking into account their current state as well as their processing capabilities. Our scheme has been compared to recent algorithms that use the particle swarm optimization and the Honey bee foraging scheme to achieve load balancing. Our experimental results show that our proposed scheme outperforms other state of the art schemes in terms of makespan, average response time, and resource utilization and provides lower degree of imbalance.

INDEX TERMS Load balancing, cloud computing, scheduling, Markov modeling, distributed environment, expected utilization, expected processing capacity.

I. INTRODUCTION

Cloud computing is a very popular internet-based technology, which provides resources and computer services on demand to customers with different needs [1], [2]. The provision of all the services are divided into three categories: Infrastructure as a service (IaaS), Software as a service (SaaS) and Platform as a servicePaaS [3]. Load balancing (LB) refers to the allocation of the workload in a distributed system like the cloud, in such a manner that the system resources almost equally loaded, that is, no resource is over- or under-loaded. In this manner, the overall system performance, which is determined by parameters like the makespan, the average response time, or the total execution time, improves dramatically.

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

The load balancing schemes can be broadly classified as proactive and reactive: the *proactive* approaches act in advance in order to prevent overloads, while the *reactive* approaches act after the overload problem appears. A narrower classification then divides the LB schemes in the following categories: *Virtual machine load balancing (VMLB)* schemes, which distribute the VMs from overloaded nodes to less loaded nodes [4], [5], *task load balancing (TLB)* schemes, which evenly distribute the tasks among the VMs [6], [7], and *resource load balancing (RLB)* schemes, which focus on the management of the available resources like servers, network links [8], CPU, memory and bandwidth [10]. The LB strategies are further divided in static and dynamic: the *static* strategies set up pre-specified workload distributions that remain unaltered during runtime, while the *dynamic* strategies have the capability of adapting to

system changes and distribute the workload during runtime. Apparently, the static LB strategies fail to respond to system changes, thus resulting in overall poor system performance. This fact turns the spotlight on the dynamic LB strategies. Dynamic load balancing is one of the most important aspects of scheduling in cloud systems. The workloads between several VMs must be distributed in such a manner that the response times and makespan values are reduced. An efficient load balancing algorithm, prevents over-utilization or even exhaustion of the available resources.

Quite often, large numbers of arriving tasks can cause resource exhaustion. In such a scenario, the VM is unable to handle a percentage of these tasks, which remain unprocessed and unaccomplished. Thus, proper VM selection is required during task distribution, which is usually based on the current workload. However, the current workload alone may not be a good criterion. This is because the workload on different machines may vary because of differences in their computing capacities [1] and because they handle tasks of different sizes and processing requirements [9].

In this work, we introduce a new dynamic task load balancing scheme, to efficiently distribute the tasks among the system's VMs. The novel idea introduced in our scheme is that it considers *fairness* as the main criterion of task distribution among the available VMs. A fair task distribution scheme: (1) assigns the new workload to each VM proportionally to its current processing capacity and (2) this assignment is implemented in such a manner that all the VMs are expected to be equally utilized after the distribution of the new tasks.

Our fair task distribution scheme is based on employing a *queuing network* model, where the LBer acts as a central server that feeds the VMs. Each VM is modeled as a queue where the assigned tasks are inserted. We employ a closed network model and we work on a time slot basis. For a short period of a time slot, we can assume that the system performance is not affected by external factors like the rate at which the users submit their tasks. The main contributions of our fair task distribution policy are summarized as follows:

- We use a closed network model and thus, our LB strategy is independent of the task arrival rates produced by the users community.
- The time-slot by time-slot system monitoring can help in fixing imbalances on time.
- The Markov process model is easy to apply and it is computationally efficient.
- The proposed scheme can be applied to heterogeneous clouds with different configurations (task sizes, service rates, and utilizations).
- The need for task migrations is present only in case when a VM is found to be performing unreliably.

The remaining of this article is organized as follows: Section 2 presents a review of the related work. In Section 3, we describe the system model and provide two motivating examples to expose the difficulties of balancing the load in a fair way. In Section 4 we present our approach to task distribution by incorporating our fair distribution strategy into a Markov process model. Section 5 presents our experimental results and Section 6 concludes this paper and gives future work directions.

II. RELATED WORK

Load balancing is one of the most important topics in cloud computing and many papers have focused on it. There is a large number of surveys that present different taxonomies of LB schemes [2], [11]–[19], among many others. In this section, we discuss papers that belong to the categories of virtual machine LB, resource LB and task LB.

VMLB distributes the VMs from overloaded to less loaded nodes. A number of VMLB strategies have been proposed; in [4], the authors propose a system for dynamic well-organized load balancing using VMware workstation and a genetic algorithm to implement migrations by examining each VM's fitness (overloaded VMs have low fitness). Machine learning algorithms have also been used to group the VMs based on resource (like RAM or CPU) utilization [20]–[23]. Other VMLB strategies are based on active monitoring to locate the least loaded virtual machine among all the virtual machines. The dynamic resource reconfiguration is performed according to the real time requirements [24], [25]. The aforementioned strategies, just like the majority of LB strategies, are reactive [2]. An interesting proactive, predictive VMLB approach can be found in [5], where the ant colony optimization is combined with the particle swarm optimization to achieve VMLB. Another predictive strategy introduces a rule-based load-balancing algorithm based on the predictions of an end-to-end system called Cicada [26].

The RLB strategies focus on the management of the available resources. Some strategies employ game theoretical approaches to RLB. For example, [1], [27], and [28]–[30] formulate the problem into a non-cooperative game among the multiple servers, where each server is informed with information for the other servers. Tang *et al.* [8] propose an approach for OpenFlow network models, which is implemented on a time slot basis. Chen *et al.* [31] consider the problem of LB in a multi-objective framework, where initially the problem of resource allocation for emergent demands is resolved. In [32] the authors present a load-balancing framework with the objective of minimizing the operational cost of data centers using a genetic algorithm for resource allocation. Weight factors have also been employed for resources like physical memory, bandwidth, number of processors, and processor speed [10]. The aforementioned strategies are reactive. Also, proactive methods can be found in the literature [33]–[35]. More specifically, Singh *et al.* [33] base their proactive approach on autonomous agents, and whenever the load of a VM approaches a threshold, the agent looks for an alternative VM in another data center; Xiao *et al.* [34] employ a non-cooperative game theoretical approach for RLB, while in [35], a predictive strategy, which efficiently predicts the future need of resources is proposed.

TABLE 1. Reactive TLB approaches.

Reference	Single/Multi Objective	Objective(s)	Technique Used	Information Considered
[39]	Single	MS	Honey bee foraging	Capacity, Load
[40]	Single	MS	Honey bee foraging	Capacity, Load
[41]	Single	RT	Simulated annealing	MIPS, execution cost, delay
[42]	Multi	MS,RU	Particle swarm opt.	MIPS, task length, number of tasks
[43]	Multi	MS, RU	Heuristic	Task numbers and sizes
[44]	Multi	MS, RU	Min-Min & genetic opt.	Load
[45]	Multi	MS, RU,	Timing violations monitoring	Load, task length
[46]	Multi	MS, RU	Agent based	Load
[47]	Multi	MS, DI	Honey bee foraging with VM pre-selection	Capacity, Load, and fairness
[48]	Multi	MS, PC	Adaptive Dragonfly	Number of tasks and VMs
[54]	Multi	RT, PC	Particle Swarm Opt.	Load, trust
Our work	Multi	MS, RT, DI, RU	Markov Process	Load, time slot, fairness, expected utilization

Abbreviations: MS=Makespan, RU=Resource Utilization, RT=Response Time, PC=Processing Costs, DI=Degree of Imbalance

Task LB has been widely studied over the past years. As with the VMLB and RBL schemes, the reactive approaches are far more widely used. The proactive TLB approaches try to detect task overloads before they actually happen [7], [36]–[38]; The main drawback of the proactive task load balancing approaches that have been proposed is that they are used in a rather traditional way and they introduce no novel concepts [2].

The reactive TLB approaches respond to a load unbalancing situation. A variety of reactive LB techniques have been proposed [39]–[48], the majority of which are multi-objective, in the sense that they aim at enhancing many metrics like makespan, response time, execution time, throughput, etc. A few single objective schemes have been proposed: More specifically, in [39], the authors implement a honey bee inspired TLB scheme; the basic idea is taken from the food finding behavior of the honey bees. The algorithm tries to optimize the makespan (overall task completion time). A similar heuristic approach is taken by Gupta *et al.* [40] to adjust the scheduling load. In [41], a Simulated Annealing (SA) approach is taken to balance the load of the cloud infrastructure and reduce the response time.

The multi-objective LB strategies are implemented in such a way, that more than one factors that characterize LB are improved. More specifically, Pradhan *et al.* [42] propose a particle swarm optimization (PSO) load balancing technique, which aims at minimizing the makespan. The same objective can be found in [43], where a heuristic-based load-balancing algorithm (HBLBA) is proposed. The TLB is expressed as an optimization problem, which aims at minimizing the makespan while maximize the resource utilization. In [44], the proposed algorithm is an extension to the Min-Min schedule using a genetic optimization algorithm, which employs the computerized search based on natural selection and genetics. The result is an improvement on the makespan and resource utilization. In [45], the authors base their strategy on monitoring possible violations of the SLA requirements by examining if the task completion time is higher than a defined deadline.

An agent based strategy is proposed in [46]; the agents assigned to resources learn to select the best sequence of the tasks that can optimize the total makespan of the workflow, enhance utilization of resources, and improve load balancing between resources. In [47], the notion of fair distribution of the workload among the VMs is considered. In contrast to the single honey bee foraging behavior (which first uses round robin to assign tasks to virtual machines and then balance their workload), this scheme first makes the VM selection by checking its load. Then, for an incoming task, it checks how the VM status changes if this task is assigned. In this way, the under-loaded VMs are selected to accommodate the incoming tasks, in order to reduce the makespan and increase the degree of load balancing. In [48], the Adaptive Dragonfly algorithm (ADA) combines the dragonfly and the firefly algorithms. ADA uses a multi-objective function to optimize the makespan, the processing costs and load. Table 1 summarizes the techniques being used by reactive TLB state-of-the-art strategies, their objectives and the system information they take into account.

Another interesting approach to task distribution is the so-called *cooperative strategy*. In this approach, communication and cooperation between the participants is required to accomplish a complex task, which is divided into smaller and simpler sub-tasks. Such tasks may require specific amount of users, with specific machine characteristics [49]–[56]. Most of the aforementioned strategies focus on a trade-off between quality and cost and first they try to find the appropriate users that will cooperate to accomplish the task, before distributing the tasks. The issue of trust among the cooperative parts is taken into account, so that the cooperation produces better results. However, these schemes do not take into account the task allocation problem in a large-scale scenario ([54] is an exception) and they miss explicit fairness mechanisms.

The proposed scheme is a novel dynamic, reactive TLB scheme, which employs a Markov model to the problem of load balancing. A time slot-by-time slot approach is taken to monitor the current load it's novelty is that the task distribution is based on considering *fairness* as the basis for the task distribution strategy. Our scheme is multi-objective and

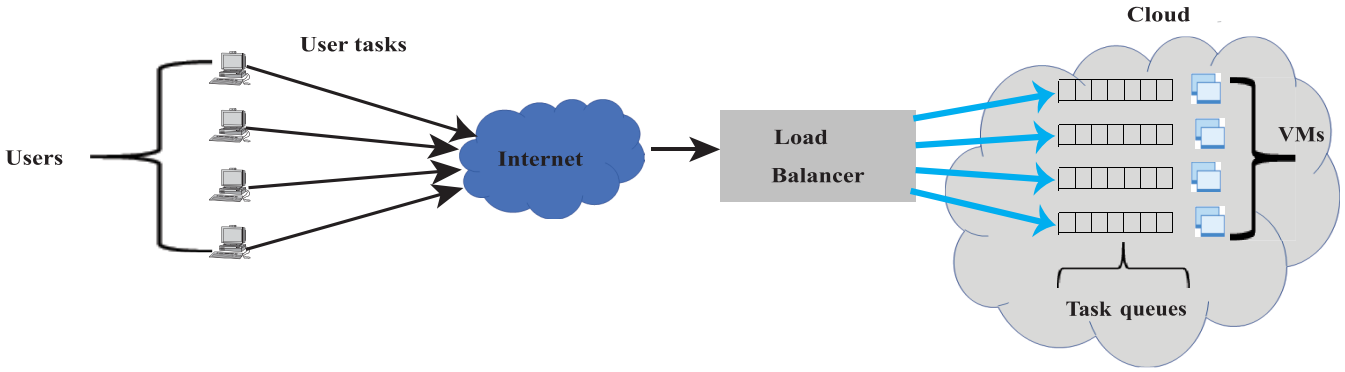

FIGURE 1. The system model.

tries to enhance 4 metrics: makespan, average response time, degree of imbalance, and resource utilization.

III. PRELIMINARIES

In this section, we first present the framework of our work and then a number of motivation examples that illustrate the difficulties of load balanced task allocation and the advantages of our work.

A. THE SYSTEM MODEL

Fig. 7 shows the system model of the proposed LB scheme. The user community generates tasks, which are assigned among the VMs of the cloud data center. The VMs are responsible for processing the user tasks. Each user submits different numbers and sizes of tasks, which have to be distributed among the VMs in such a way that their load remains as balanced as possible, in order to achieve good performance.

1) MARKOV PROCESS MODEL

From Fig.1, it is clear that each VM is modeled as a queue system; the user tasks to be processed by a VM enter its queue, so each VM can be considered as a single-server model. A central server, the *load balancer* (LBer) is the input to the VMs. We let s represent the overall number of VMs plus the LBer. Particularly, throughout this work, we use the index value of 1 for the LBer and the remaining $s-1$ index values for the VMs. The *state* of our network with s elements (the LBer and the VMs) is given by a vector $\mathbf{N} = (N_1, N_2, \ldots N_s)$, where N_k is the number of tasks being processed in an element i and N is the overall number of tasks, that is $\sum_{i=1}^{s} N_k = N$. For $k = 1$ (the LBer), processing refers to the allocation of tasks, while for $k = 2, \ldots, s$ (the VMs) it refers to task accomplishment.

The Markov proposes model is irreducible, that is, each state can be reached from any other state with non-zero probability. Therefore, the equilibrium state probability distribution can be derived. If μ_k is the expected processing capacity for VM_k (the processing capacity under a certain load), the equilibrium state probability is

$$P(\mathbf{N}) = x_1^{N1} x_2^{N_2} \cdots x_s^{N_s} P(N, 0, \ldots, 0) \tag{1}$$

where $P(N, 0, \ldots, 0)$ is the probability that all the tasks are still located in the LBer (all the tasks are unallocated), $P(\mathbf{N}) = p(N_1, N_2, \ldots, N_s)$, $x = \mu_1/\mu_i$, μ_1 is the expected processing capacity in the LBer, and

$$\sum_{\text{all N}} p(\mathbf{N}) = 1 \tag{2}$$

In [57], the authors have proven that, the *equilibrium state probability* for a server unit is given by

$$p(\mathbf{N}) = \frac{1}{F(N)} \prod_{k=2}^{s} y_k^{N_s} \tag{3}$$

where

$$y_k = \frac{\mu_1}{\mu_k} b_{1k}, \quad k = 2, \ldots, s, \tag{4}$$

is the *distribution factor* between the LBer and VM_k, b_{1k} is the probability of distributing one task to VM_k under the current system configuration, and $F(N)$ expresses all the possible combinations of task placements within each server unit:

$$F(N) = \sum_{\text{for all N}} \prod_{k=2}^{s} y_k^{N_k}, \tag{5}$$

The exponent terms N_k in Eq. 5 are all the possible numbers of tasks that may be allocated in VM_k. This number can vary from 0 to N (in which case VM_k has N tasks and all the others are empty). If we consider *only the states* where $N_1 > 0$, that is, at least one task remains undistributed by the load balancer we obtain $F(N-1)$:

$$F(N-1) = \sum_{\substack{\text{for all N,} \\ N_1 > 0}} \prod_{k=2}^{s} y_k^{N_k}, \tag{6}$$

and the expected utilization for the LBer can be computed by dividing $F(N-1)$ by $F(N)$, where $F(N)$ includes also the states where N_1 may be reduced to 0:

$$p_1 = \frac{F(N-1)}{F(N)}. \tag{7}$$

This value shows the percentage of time the LBer is expected to be busy in distributing the N tasks and increases with N until reaching 1. When p_1 is known, the expected utilizations of the VMs for all the possible task placements can be computed as [57]:

$$p_k = p_1 y_k. \tag{8}$$

while the expected utilization p_k for VM_k is the percentage of time the VM is expected to be busy in processing a portion of the N tasks.

2) SERVER AVAILABILITY

In this work, we use a variation of the VM availability model proposed in [1]:

$$\mu_k = \overline{\mu}_k - \ell_k, \quad k = 2, \ldots, s \tag{9}$$

where μ_k is the expected processing capacity (in number of tasks) of a VM_k, $\overline{\mu}_k$ is its maximum processing capacity, ℓ_k is the current workload, and $\overline{\mu}_k > \ell_k$. The maximum processing capacity refers to the processing capacity without considering the load impact, while the expected processing capacity is the capacity affected by the current load. A fair assumption that holds throughout the model is that the maximum processing capacity of each VM is independent of the corresponding rate of the other VMs. This independence assumption is necessary, since clouds are usually heterogeneous and integrate different components.

3) MODEL PARAMETERS

This section describes some of the model parameters, which will be used to evaluate our strategy and make comparisons.

The *computing power* (CP) of each VM, is expressed in Millions of Instructions Per Second (MIPS), is given by

$$CP_k = \sum_{\text{all cores}} \text{core cp.} \tag{10}$$

that is, the total computing power of all the cores assigned to VM_k. The *execution time* of a task N is given by

$$\text{ET}_k = \frac{task\_size}{CP_k}, \quad k = 2, \ldots, s \tag{11}$$

that is, the size of the task divided by the CP of the executing VM k. The *makespan* is the maximum of the completion times (CT) of all the N tasks assigned to the VMs:

$$\text{MS} = \max(\text{CT}_i), \quad i = 1, \ldots, N \tag{12}$$

Finally, the response time of a task i within a VM's queue is the time elapsed from the task submission up to the time it starts its execution. The execution starts after the completion of $i-1$ tasks, which are ahead of task i in the queue:

$$\text{RT}_i = \sum_{j=1}^{i-1} (CT_{i-1}) - AT_i, \quad i = 1, \ldots, N \tag{13}$$

Table 2 summarizes the notations used in this work.

TABLE 2. Notations used in this paper.

Notation	Description
$s-1$	Total number of virtual machines
$\mathbf{N}$	Network state vector
N_k	The number of tasks assigned to VM k
μ_k	The expected processing capacity of each VM_k
μ_1	The expected processing capacity of a LBer
$\overline{\mu}_k$	The max. processing capacity of a VM_k
b_{1k}	The probability of distributing tasks to VM_k
y_k	The distribution factor for VM_k
ℓ_k	Indicates the workload at VM_k
p_1	Expected utilization of the LBer
p_k	Expected utilization of VM_k
CP_k	Total computing power of VM_k
ℓ_k	Current load of VM_k
ET_i	Execution time for task i
CT_i	Completion time for task i
WT_i	Waiting time for task i
AT_i	Arrival time for task i
RT_i	Response time for task i

TABLE 3. Parameters for the motivating examples.

Ex.	Task size (in MI)	# of Tasks	# of VMs	Comp. Power (in MIPS)
1	10000	40	4	(2000, 1800, 1100, 500) MIPS
2	From 4000 to 20000	40	4	(2000, 1800, 1.100, 500) MIPS

B. MOTIVATING EXAMPLES

Consider a small set of four VMs, as shown in Fig. 7 The user community generates a number of tasks, which are assigned among the VMs of the cloud data center. Each user submits different numbers and sizes of tasks, which are queued in each VM. The parameters for the examples that follow are given in Table 2.

EXAMPLE 1: Consider an example with an heterogeneous system of 4 virtual machines, VM_1-VM_4 with computing power of 2000, 1800, 1100, and 500 MIPS, respectively. Assume that in the beginning of a slot, 20 tasks have remained unprocessed, 5 in each VM. For simplicity, assume that these tasks arrive at time $t = 0$. In the new slot, another 40 tasks have been produced and need to be distributed. The new slot starts at time $t = 20$. A simple active monitoring scheduler would keep information about each VM and the number of tasks currently assigned on each VM. Upon the arrival of new tasks, it would select the VMs with the fewer tasks. In our example, such a scheme would distribute these tasks equally among the VMs, that is, 10 tasks per VM. The execution time of each task would be $\frac{10000}{2000} = 5$ sec for tasks in VM_1, $\frac{10000}{1800} = 5.55$ sec for tasks in VM_2, $\frac{10000}{1100} = 9.09$ sec for tasks in VM_3, and $\frac{10000}{500} = 20$ sec for tasks in VM_4. The makespan will be the completion time of the last task running in the slowest VM_4, $15 \times 20 = 300$ sec. By applying Eq. 13, we get each task's response time, for the simple active monitoring scheduler. These values are given in Table 4. The average response time for all the tasks is 49.43. The total execution time is the sum of the execution times for each task (as computed by Eq.11): $\frac{(15\times20000)}{2000} + \frac{(15\times20000)}{1800} + \frac{(15\times20000)}{1100} + \frac{(15\times20.000)}{500} = 594.7$. Thus, the average execution time is $594.7/60 = 9.91$.

TABLE 4. Response times for the tasks of Example 1.

Average	Simple Active Monitoring Scheduler				Fair Scheduler			
N	VM_1	VM_2	VM_3	VM_4	VM_1	VM_2	VM_3	VM_4
1	0	0	0	0	0	0	0	0
2	5	5.55	9.09	20	5	5.55	9.09	20
3	10	11.1	18.18	40	10	11.1	18.18	40
4	15	16.65	27.27	60	15	16.65	27.27	60
5	20	22.2	36.36	80	20	22.2	36.36	80
6	0	2.2	16.36	60	0	2.2	16.36	
7	5	7.75	25.45	80	5	7.75	25.45	
8	10	13.3	34.54	100	10	13.3	34.54	
9	15	18.85	43.63	120	15	18.85	43.63	
10	20	24.4	52.72	140	20	24.4	52.72	
11	25	29.95	61.81	160	25	29.95	61.81	
12	30	35.5	70.9	180	30	35.5	70.9	
13	35	41.05	79.99	200	35	41.05		
14	40	46.6	89.08	220	40	46.6		
15	45	52.15	98.17	240	45	52.15		
16					50	57.7		
17					55	63.25		
18					60	68.8		
19					65	74.35		
20					70	79.9		
21					75			
22					80			
sum	275	327.25	663.55	1700	730	671.2	476.3	200
Average	**49.43**				**34.62**			

TABLE 5. Task sizes for Example 2.

Tasks	**1-8**	**9-16**	**17-24**	**25-32**	**33-40**
Size	20000	16000	12000	8000	4000

Our fair task distribution scheme considers the maximum processing capacity and the current load $A\ell_i$ of each VM. In such a scheme (the details are given in the next section), the 40 tasks are distributed as follows:

- VM_1:17 tasks, 22 in total
- VM_2:15 tasks, 20 in total
- VM_3: 8 tasks, 13 in total
- VM_4: 0 tasks, 5 in total

Then, the task execution per VM will be completed at: $(17+5)\times 5 = 110$ for VM_1, $(15+5)\times 5.55 = 111$ for VM_2, $(8+5)\times 9.09 = 118.7$ for VM_3, and $(5)\times 20 = 100$ for VM_4. The makespan is 118.5, so there is a reduction of $(300-118.5)/300 = 60\%$. The response times for the tasks are given in Table 4, for the fair scheduler. The average response time for all the tasks is 34.62, a reduction of about 30% compared to the simple active monitoring scheduler. The total execution time is: $\frac{(22\times 20000)}{2000}+\frac{(20\times 20000)}{1800}+\frac{(13\times 20000)}{1100}+\frac{(7\times 20000)}{500} = 479.29$. Thus, the average execution time is $479.29/60 = 7.98$. The reduction of the average execution time compared to the simple active monitoring task scheduler is about 19%.

EXAMPLE 2: In the second example, the VMs have different computation power and the task sizes differ, as shown in Table 3. Table 5 shows the size of the 40 tasks to be distributed. As in Example 1, 20 tasks have remained unprocessed, 5 in each VM. These tasks arrived at $t = 0$. Also, the batch of these 40 tasks arrive at $t = 20$.

Again, a simple active monitoring scheduler would distribute the incoming tasks equally (10 per VM) and assign the tasks according to their computational intensity and the processing capacity of the VMs. Thus, the most intensive tasks will be assigned to the fastest VM, the next most intensive tasks will be assigned to the next fastest VM, and so on. Table 6 shows this distribution, while Table 7 provides the response times for each task in this schedule.

TABLE 6. Simple active monitoring scheduler for Example 2.

Task sizes	VM_1	VM_2	VM_3	VM_4
Size= 20000	8	–	–	–
Size= 16000	2	6	–	–
Size= 12000	–	4	4	–
Size= 8000	–	–	6	2
Size= 4000	–	–	–	8
Total # of tasks:	10	10	10	10
Total execution time: (*including unprocessed tasks*)	121	107.7	132.7	196

TABLE 7. Response times for the tasks of Example 2.

Average	Simple Active Monitoring Scheduler				Fair Scheduler			
N	VM_1	VM_2	VM_3	VM_4	VM_1	VM_2	VM_3	VM_4
1	0	0	0	0	0	0	0	0
2	5	5.55	9.09	20	5	5.55	9.09	20
3	10	11.1	18.18	40	10	11.1	18.18	40
4	15	16.65	27.27	60	15	16.65	27.27	60
5	20	22.2	36.36	80	20	22.2	36.36	80
6	0	2.2	16.36	60	0	2.2	16.36	
7	10	11	27.26	76	10	11.08	23.63	
8	20	19.8	38.16	92	20	19.96	30.9	
9	30	28.6	49.06	100	30	28.84	38.17	
10	40	37.4	59.96	108	40	37.72	45.44	
11	50	46.2	67.23	116	50	46.6	52.71	
12	60	55	74.5	124	60	53.26	59.98	
13	70	61.66	81.77	132	70	59.92	67.25	
14	80	68.32	89.04	140	80	66.58	74.52	
15	88	74.98	96.31	148	88	73.24	78.15	
16					96	79.90	81.78	
17						86.56	85.41	
18						93.22	89.04	
19						99.88	92.67	
20							96.3	
sum	498	460.66	690.55	1296	594	814.46	1023.21	200
Average	**49.08**				**43.86**			

Thus, VM_1 will be assigned 8 tasks with size 20000 plus two tasks of 16000. The execution time required for these tasks will be: $\frac{(20000\times 8)+(16000\times 2)}{2000} = 96$ sec. Also, the unprocessed 5 tasks require another 25 sec. So, the total execution time for the fastest VM will be 121 sec. The execution times for the remaining VMs are given in the bottom of Table 6. The average execution time is $\frac{121+107.7+132.7+196}{60} = 9.29$.

Our fair distribution scheme, which considers each VM's processing capacity will produce the schedule shown in Table 8. The makespan is 129.09 sec, an improvement of 34% compared to the simple active monitoring scheme. The right side of Table 7 shows the response times for all the tasks. The average response time is 44.74, an improvement of 10% compared to the simple active monitoring schedule. The execution times for the VMs are given in the bottom of Table 8. The average execution time is $\frac{129+127.77+129.09+100}{60} = 8.09$, an improvement of about 12% compared to the average execution time of the simple active monitoring strategy.

IV. OUR APPROACH TO TASK DISTRIBUTION

A. PROBLEM FORMULATION

Objective: The main objective of our approach is to fairly distribute the N incoming tasks submitted to the cloud within a time slot, so that all the VMs would work for the same percentage of time to process the incoming load, based on

TABLE 8. Fair task distribution for Example 2.

Task sizes	VM_1	VM_2	VM_3	VM_4
Size= 20000	8	–	–	–
Size= 16000	3	5	–	–
Size= 12000	–	8	–	–
Size= 8000	–	–	8	–
Size= 4000	–	1	7	–
Size= 10000 (*including unprocessed tasks*)	5	5	5	–
Total # of tasks::	16	19	20	5
Total execution time:	129	127.77	129.09	100

the current system configuration, that is, current VM load and VM processing capacity. This strategy guarantees of load balancing, as we will prove later in this section.

Let us consider an source task distribution R, which is defined as a set of values:

$$R = \{\overline{\mu}_k, \mu_k, \ell_k, p_k\}, \quad k = 2, \ldots, s \tag{14}$$

Our objective can be described as a transition from R to a target distribution R^* such that the resulting expected utilizations are equal among all the VMs

$$R \rightarrow R^* = \begin{cases} \{\overline{\mu}_k, \mu_k, \ell_k, p_k, b_{1k}\} \rightarrow \{\overline{\mu}_k^*, \mu_k^*, \ell_k^*, p_k^*, b_{1k}^*\} \\ \text{all } p_k^* \text{ values are equal}, \quad k = 2, \ldots, s \end{cases} \tag{15}$$

In a new time slot, the LBer has N new tasks to distribute to the VMs, with an expected processing capacity μ_1 and expected utilization p_1. In the target distribution, the N tasks would be distributed to $(k-1)$ VMs, resulting in lower maximum processing capacities $\overline{\mu}_k^*$, lower expected processing capacities μ_k^*, larger current loads ℓ_k^* and *equal* expected utilizations p_k^* for the system's VMs. The probability b_{1k} may increase or decrease, depending on the workload assigned to VM_k.

From Equations 4 and 10, we see that the expected utilizations of the VMs, p_k are:

$$p_k = p_1 \frac{\mu_1}{\mu_k} b_{1k} \tag{16}$$

In this regard, the expected utilizations are a function of the LBer's utilization p_1, and its expected processing capacity over a time slot, μ_1. In the following, we will assume that $\mu_1 = N$ over a period of a time slot, that is, the balancer can distribute N incoming tasks over a period of a time slot. The expected processing capacity of the VMs, μ_k, is computed as $\overline{\mu}_k - \ell_k$ (Eq.9), where the maximum processing capacity $\overline{\mu}_k$ depends on the computing power of each VM. Finally, b_{1k} is equal for all the VMs in the source distribution, but it changes to reflect the different loads that will be assigned to each VM after the task distribution. In the remaining paragraphs of this section, we present how our fair task distribution scheme can be embedded in the Markov process model and we prove that it leads to equal expected utilizations and finally to load balancing.

B. OUR FAIR TASK DISTRIBUTION STRATEGY

The key idea for our task distribution strategy is fairness. By *fairness*, we mean that the new workload of N tasks must be distributed in such a way that (i) each VM takes on an added workload proportional to its current processing capacity and (ii) the expected utilization of all the VMs becomes equal after the distribution, that is, the percentage of time during which the VMs will be busy with processing these newly assigned tasks will be equal. To embed fairness in the Markov process model, we take the following steps:

`STEP 1`: Initially, we compute the u_k term, $k = 2, \ldots s$, for each VM, which is the ratio of its expected processing capacity and the total number of tasks to be distributed during a time slot, μ_1.

$$u_k = \frac{\mu_k}{\mu_1}, \quad k = 2, \ldots, s \tag{17}$$

The larger the u_k values, the larger the workload a VM is able to take over.

`STEP 2`: The *loading balancing factor*, f, can simply be computed as the fraction of μ_1 and the sum of the u_k terms:

$$f = \frac{\mu_1}{\sum_{k=2}^{s} u_k}, \tag{18}$$

`STEP 3`: The workload per VM is computed as

$$\ell_k^* = f \times u_k, \quad k = 2, \ldots, s \tag{19}$$

`STEP 4`: The expected processing capacity after distribution is μ_k^* will be

$$\mu_k^* = \mu_k - \ell_k^*, \quad k = 2, \ldots, s \tag{20}$$

that is, the current processing capacity minus the newly assigned load ℓ^*.

`STEP 5`: The probability of distributing tasks to VM_k changes to $b_{1k}^* = \ell_k^*/\mu_1$; this quantity is the percentage of the total workload that has been assigned to the VM_k during this time slot.

In the following propositions, we prove that if we embed these steps to the Markov process model, we can obtain a well-balanced task distribution.

Proposition 1: The distribution of ℓ_k^* load to each VM, as computed by our fair policy reduces proportionally the expected processing capacities μ_k of the VMs.

Proof: Let us consider the expected processing capacities of two randomly taken VMs, μ_{k1} and μ_{k2}. From Eq. 18 and 19, we get that

$$\ell_k^* = \frac{\mu_1}{\sum_{k=2}^{s} u_k} \times u_k = \frac{\mu_1 u_k}{(u_2 + \cdots + u_k)} \overset{(17)}{=} \frac{\mu_k}{(u_2 + \cdots + u_k)} \tag{21}$$

From Eq. 9 we are aware that the expected processing capacity of each VM decreases as more load is added. Let us consider μ_{k1} and μ_{k2}: We know that μ_{k1} will be reduced by

a factor of $d(\mu_{k1})$ and will become $\mu^*_{k1} = \mu_{k1} - \mu_{k1}d(\mu_{k1})$ and $\mu^*_{k2} = \mu_{k2} - \mu_{k2}d(\mu_{k2})$. Therefore,

$$d(\mu_{k1}) = \frac{\mu_{k1} - \mu^*_{k1}}{\mu_{k1}}, \quad d(\mu_{k2}) = \frac{\mu_{k2} - \mu^*_{k2}}{\mu_{k2}} \tag{22}$$

The numerators of Eq. 22 are equal to ℓ^*_{k1} and ℓ^*_{k2}. Thus, Eq. 22 becomes:

$$d(\mu_{k1}) = \frac{\ell^*_{k1}}{\mu_{k1}}, \quad d(\mu_{k2}) = \frac{\ell^*_{k2}}{\mu_{k2}} \tag{23}$$

The two ratios $d(\mu_{k1}), d(\mu_{k2})$ are equal if

$$\mu_{k2}\ell^*_{k1} = \mu_{k1}\ell^*_{k2} \overset{(21)}{\Rightarrow}$$
$$\mu_{k2}\frac{\mu_{k1}}{(u_2 + \cdots + u_k)} = \mu_{k1}\frac{\mu_{k2}}{(u_2 + \cdots + u_k)}$$

which is always true and this completes the proof. ■

Proposition 2: Our fair task distribution policy produces equal expected utilizations p^*_k among all the VMs.

Proof: We set all the equal $d(\mu_k)$ values that we computed in Proposition 1 as ω. Now, let us consider the expected utilizations of two randomly selected VMs k_1 and k_2. Let us rewrite Eq.16 for the target distribution, for a VM k_1:

$$p^*_{k1} = p_1\frac{\mu_1}{\mu^*_k}b^*_{1k1}$$

We know that $b^*_{1k} = \ell^*_k/\mu_1$ and, in the end of Proposition 1, we have shown that $\mu_{k2}\ell^*_{k1} = \mu_{k1}\ell^*_{k2}$. Also, from Eq. 22 it follows that $\mu^*_{k1} = (1-\omega)\mu_{k1}$. Thus,

$$\begin{aligned} p^*_{k1} &= p_1\frac{\mu_1}{\mu^*_{k1}}b^*_{1k1} \\ &= p_1\frac{\mu_1}{(1-\omega)\mu_{k1}}\frac{\ell^*_{k1}}{\mu_1} \\ &= \left(\frac{\ell^*_{k1}}{\mu_{k1}}\right)\frac{p_1\mu_1}{(1-\omega)\mu_1} \end{aligned}$$

Similarly

$$p^*_{k2} = \left(\frac{\ell^*_{k2}}{\mu_{k2}}\right)\frac{p_1\mu_1}{(1-\omega)\mu_1}$$

As $\mu_{k2}\ell^*_{k1} = \mu_{k1}\ell^*_{k2}$ and $\frac{p_1\mu_1}{(1-\omega)\mu_1}$ is the same for both expected utilizations, it follows that $p^*_{k1} = p^*_{k2}$, to complete the proof. ■

C. LOAD BALANCING ANALYSIS

To prove that our scheme provides load balancing, we use the *load imbalance factor*, which is defined as follows:

$$\delta(t) = \frac{\sum_{s=2}^{k}\left[\overline{\delta(t)} - p_k(t)\right]^2}{s} \tag{24}$$

where s is the number of processing elements and $\overline{\delta(t)}$ is the average expected utilization over a time slot t, given by:

$$\overline{\delta(t)} = \frac{\sum_{s=2}^{k} p_k}{s} \tag{25}$$

The load imbalance factor $\delta(t)$ measures the variance between the average expected utilization and the expected utilization at each time slot. Our scheme is reactive, that is, it divides the time into slots and reacts after the first slot, t_0 that imbalances occur. In [8], the authors provided a definition for a load balanced network:

Definition 1: If $\delta(t) < \Delta$, where Δ is a threshold value, the network is balanced.

We will use this definition to prove that our scheme guarantees load balancing.

Proposition 3: Our fair task distribution strategy guarantees load balancing on a slot-by-slot basis.

Proof: To prove this proposition, we need to prove that, for every time slot t, $\delta(t)$ is bounded by a predefined threshold. Let us start with a time slot t_0 (source distribution), where an imbalance situation is spotted. The load imbalance factor is $\delta(t_0)$ and it is computed by Eq. 24. By the end of the next slot t_1, when the task distribution is completed, the average expected utilization is

$$\overline{\delta(t_1)^*} = \frac{\sum_{s=2}^{k} p^*_k}{s} = \begin{cases} p^*_k, & \text{all } p_k < 1 \\ \dfrac{(s-\alpha)p^*_k}{s}, & \text{for } \alpha \text{ VMs: } p_k \geq 1 \end{cases} \tag{26}$$

Case 1: If all the p_k values are < 1, the VMs can be assigned extra workload. In this case, because we have already proved that the expected utilizations will be equal, the average expected utilization will be p^*_k and thus,

$$\delta(t_1) = \frac{\sum_{s=2}^{k}\left[p^*_k(t) - p^*_k(t)\right]^2}{s} = 0$$

Case 2: If α VMs are already overloaded ($p_k \geq 1$), they take no extra workload and their p_k values will be 0 in this time slot. The remaining $s-\alpha$ VMs will have equal p^*_k values. Thus the average expected utilization will be $\frac{(s-a)p^*_k}{s}$. Let us consider the numerator of Eq. 24:

- If $p^*_k = 0$, then the term produced will be $\overline{\delta(t)} = \frac{(s-a)p^*_k}{s}$
- If $p^*_k > 0$, the term produced will be $< \overline{\delta(t)}$.

Thus, the largest term that can be produced is $\overline{\delta(t)}$ and because there can be (worst case) at most $\alpha = s-1$ zero $p^*_k = 0$ terms,

$$\delta(t) \leq (s-1)\frac{(\overline{\delta(t)})^2}{s} \tag{27}$$

Thus, we set $\Delta = (\alpha - 1)\frac{(\overline{\delta(t)})^2}{s}$ as our threshold value immediately after an imbalance is spotted. We have proved that $\delta(t)$ will never exceed Δ, thus our scheme guarantees load balancing. ■

D. TIME ANALYSIS

To analyze the complexity of our strategy, we need to analyze the complexity of the computations required for the transition from $R \rightarrow R^*$. It is known that the computations for $F(N)$, which are required to find the p values are recursive:

$$F_k(N) = \begin{cases} 1, & N = 0 \\ F_{k-1}(N) + y_k F_{k-1}(N-1), & N > 0 \end{cases} \tag{28}$$

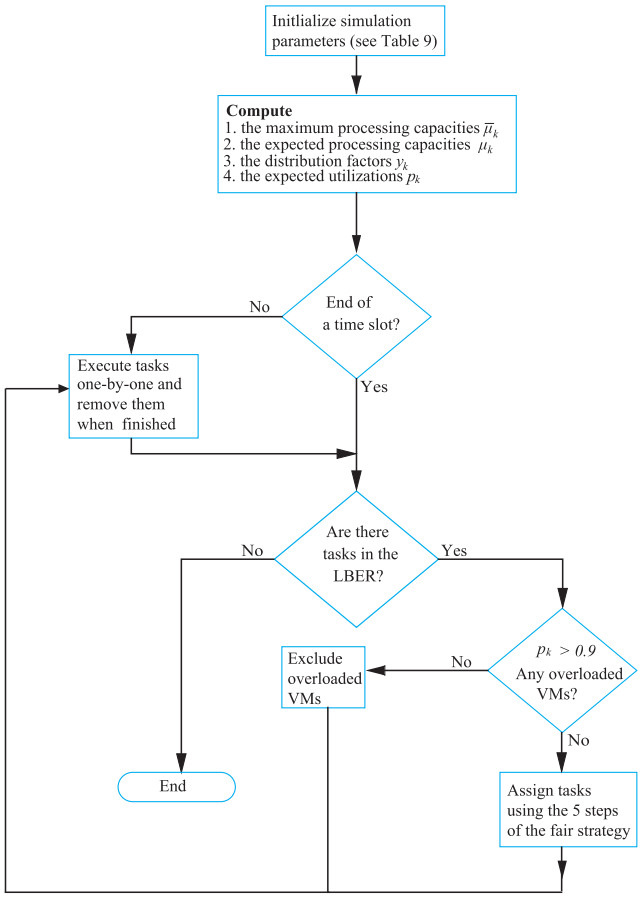

FIGURE 2. Simulator workflow.

and they are clearly $O(N)$. The computations of Equations 17-20 are clearly $O(s)$. That is, our scheme is computationally efficient as the computations grow linearly with the number of incoming tasks N and the number of processing elements s. By making the fair assumption that $N > s$, the complexity is $O(N)$.

E. BACK TO THE MOTIVATING EXAMPLES

In this subsection, we briefly show how our strategy applies in the motivating examples of Section 3.

1) EXAMPLE 1

In slot t_0, $\mu_1 = 40$, $\ell_k = 5$ tasks, and $b_{1k} = \frac{1}{4} = 0.25$ (equal probabilities for all the VMs. The maximum processing capacity for each VM_k has been computed as

$$\overline{\mu}_k = \frac{\mu_1}{M_k}, \quad \text{where } M_k = \frac{CP_{\max}}{CP_k}$$

The M_k term is a factor that expresses the processing capacity of each VM proportionally to the maximum CP that exists among the VMs. In this example, $M_2 = 2000/2000 = 1$, $M_3 = 2000/1800 = 1.11$, $M_4 = 2000/1100 = 1.82$, and $M_5 = 2000/500 = 4$. Thus, the maximum processing capacity for each VM is: $\overline{\mu}_2 = 40/1 = 40$, $\overline{\mu}_3 = 40/1.11 = 36$, $\overline{\mu}_4 = 40/1.82 = 22$ and $\overline{\mu}_5 = 40/4 = 10$ MIPS. The μ_k values are computed by Eq. 9, so $\mu_2 = 40 - 5 = 35$, $\mu_3 = 36 - 5 = 31$, $\mu_4 = 22 - 5 = 17$, and $\mu_5 = 10 - 5 = 5$.

The y values are computed by Eq. 4: $y_2 = \frac{40}{35} \times 0.25 = 0.29$, $y_3 = \frac{40}{31} \times 0.25 = 0.32$, $y_4 = \frac{40}{17} \times 0.59 = 0.29$, and $y_4 = \frac{40}{5} \times 0.25 = 2.0$. For these y values and by applying Eq.7, we find that $p_1 = 0.5$ and from Eq.8, we find the expected utilizations for the VMs: $p_2 = 0.5 \times 0.29 = 0.14$, $p_3 = 0.5 \times 0.32 = 0.16$, $p_4 = 0.5 \times 0.59 = 0.29$, and $p_5 = 0.5 \times 2 = 1$. Because $p_5 = 1$, VM_5 is already overloaded and no load will be assigned to it.

Now, we show the 5 steps of our fair distribution scheme: By applying the μ_k values to Eq. 17, we get the other u values: $u_2 = \frac{35}{40} = 0.875$, $u_3 = \frac{31}{40} = 0.775$, $u_4 = \frac{17}{40} = 0.425$. Finally, u_5 is set to 0 due to the overload detected. From Eq. 18, $f = 40/(0.875 + 0.775 + 0.425 + 0) = 19.27$. Then the loads ℓ_k^* are given by Eq.19: $\ell_2^* = 19.27 \times 0.875 = 16.86$, $\ell_3^* = 19.27 \times 0.775 = 14.93$, $\ell_4^* = 19.27 \times 0.425 = 8.19$, and $\ell_5^* = 0$. Thus, the four VMs are assigned 17, 15, 8, and 0 tasks, respectively.

The expected processing capacities after the distribution will be (see Eq.20) $\mu_2 = 35 - 17 = 18$, $\mu_3 = 31 - 15 = 16$, $\mu_4 = 17 - 8 = 9$, and $\mu_5 = 5 - 0 = 5$. The corresponding b_{1k}^* values are: $b_{12} = \frac{16.86}{40} = 0.421$, $b_{13} = \frac{16.86}{40} = 0.373$, $b_{14} = \frac{8.19}{40} = 0.204$, and $b_{15} = \frac{0}{40} = 0$. By applying these values to Eq.16, we get $p_2 = p_3 = p_4 = p_5 = 0.465$.

Finally, let us see how the load balancing analysis of Section 4.3 applies in this example. In slot t_0, we have (see Eq.22) $\overline{\delta(t_0)} = \Delta = (0.14 + 0.16 + 0.29 + 1)/4 = 0.399$. Then, from Eq. 26, we obtain that
After the task distribution, $\overline{\delta(t_1)} = (0.465 + 0.465 + 0.465 + 0)/4 = 0.348$, thus *(see Eq. 27) $\delta(t_1) \leq 3\frac{(0.348)^2}{4} = 0.091$ By Eq.24 we get:

Apparently, $\delta(t_1) < \Delta$. From the proof of Proposition 3, it is clear that if we apply our scheme for a number of time slots, this inequality will hold and balancing is guaranteed.

2) EXAMPLE 2

In the second example, because the task sizes are different, we make some changes in the source distribution. Specifically, we set μ_1 as the total workload (not the number of tasks as in Example 1) that is to be distributed and we accordingly set the maximum and processing capacities. In this example, the total workload is $8(20000 + 16000 + 12000 + 8000 + 4000) = 48000$. The maximum processing capacities are: $\overline{\mu_2} = 480000$, $\overline{\mu_3} = 480000/1.11 = 432000$, $\overline{\mu_4} = 480000/1.82 = 264000$, and $\overline{\mu_5} = 480000/4 = 120000$. The remaining computations are as in Example 1. The ℓ_k^* values are expressed as a workload size. To find the number of tasks, we use a change-making problem approach that approximates the minimum number of tasks that correspond to this workload. We start from the fastest VM, in order to assign to it the majority of heavy tasks and proceed to the next fastest, and so on. For this example, the distribution (expressed in workload size) is $\ell_2^* = 201169$, $\ell_3^* = 178713$, $\ell_4^* = 100116$, and $\ell_5^* = 0$. For ℓ_2^*, we need 8 tasks of size

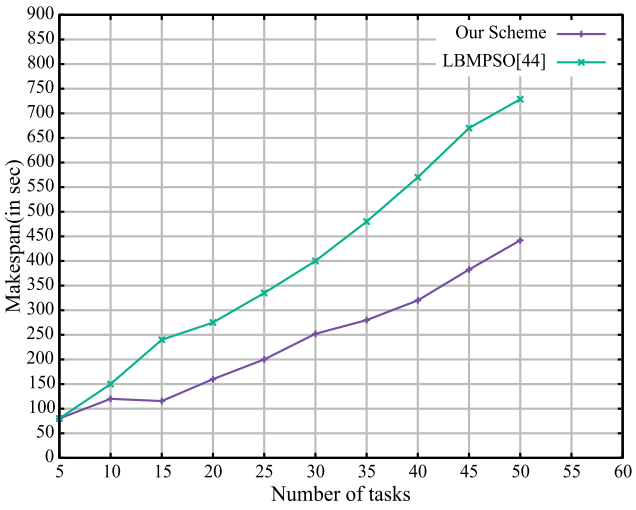

FIGURE 3. Comparison of makespan between our scheme and the LBMPSO scheme [42].

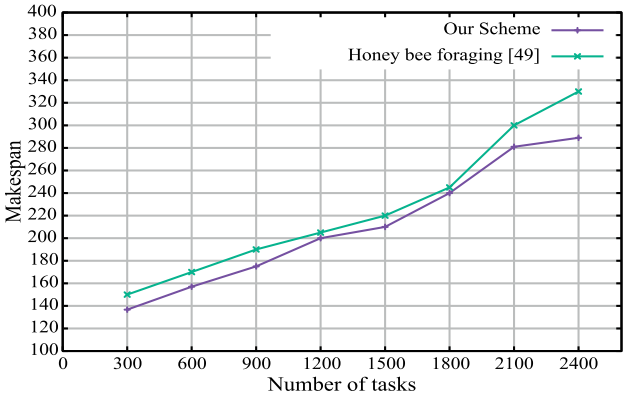

FIGURE 4. Comparison of makespan between our scheme and the HBF scheme [47].

20000 and another 3 of size 16000, for ℓ_3^*, we need 5 tasks of size 16000, another 8 of size 12000 and one task of size 4000. Finally, for ℓ_3^* we need 8 tasks of size 8000, another 7 of size 4000 and one task of size 4000.

V. EXPERIMENTAL RESULTS

The proposed scheduling strategy is evaluated on GRNET's cloud service okeanos-knossos, which provides a wide range of choices to develop, debug and evaluate an experimental system. We ran 3 experiments to compare strategy with 3 newly proposed schemes: the Load Balancing Modified Particle Swarm Optimization (LBMPSO) [42], the Honey bee foraging (HBF) with VM pre-selection [47], and the community-based Particle Swarm Optimization (CPSO) scheme introduced in [54]. These schemes are multi-objective; the first aims at enhancing the makespan and the resource utilization, the second aims at enhancing the makespan, the degree of LB, and the response time, while the third aims at reducing the processing costs and the response time. Also, recall that the third strategy has been selected among a many community-based dynamic task allocation strategies, because of its large-scale capabilities. For each experiment we ran 10 simulations and averaged the results. Table 9 shows the basic simulation parameters.

The simulation system works in a manner quite similar to what we have already described in our motivating examples. However, we added the rule that, if the expected utilization of a VM exceeds 0.9, it cannot accept any task requests, that is, its u_k value becomes 0. This VM can accept further requests once it completes the execution of its tasks, so its utilization drops below 0.9. Fig. 2 presents a flowchart of how our simulator works.

In all the experiments, we initially (in the first time slot) generated an imbalance situation by equally assigning a small number of tasks among the heterogeneous VMs. In the remaining time slots, we iteratively assigned tasks to the available VMs using our strategy, to remove the imbalance. For the first experiment, we have 2 VMs of 250 MIPS and 3 VMs of 300 VMs. For the second experiment, we have 5 VMs of 500 MIPS, 5 VMs of 1000 MIPS, . . . , and 5 VMs of 4000 MIPS. Finally, the tasks were generated randomly, but their sizes were uniformly distributed.

A. MAKESPAN

The makespan is perhaps the most widely used and important metric to evaluate a task scheduling algorithm. In Fig.3 we compare the makespan of our scheme to the makespan of the LBMPSO scheme. We have proved that our scheme can balance the load under different task numbers and sizes that may be produced during a time slot. This fact eliminates the need for task migrations, which are necessary only in cases where VMs are unavailable due to failures. The LBMPSO scheme utilizes the task migration approach whenever the VMs are overloaded, thus its makespan was found to be about 40% higher compared to our scheme. Fig. 4 compares the makespan of our scheme to the efficiency of the HBF with VM pre-selection. The HBF is highly based on assigning the tasks to a group of underloaded, thus suitable, VMs. For

$$\delta(t_0) = \frac{(0.399-0.14)^2+(0.399-0.16)^2+(0.399-0.29)^2+(0.399-1)^2}{4} = 0.12.$$

$$\delta(t_1) = \frac{(0.348-0.465)^2+(0.348-0.465)^2+(0.348-0.465)^2+(0.348)^2}{4} = 0.04.$$

TABLE 9. Simulation parameters and scenaria.

Experiment	# of tasks	Task sizes	# of VMs	MIPS of VMs	Comparable schemes	Parameters compared
1	10 - 50	1000-6000	3-5	250-300	[44]	Makespan, Average utilization
2	20-3000	up to 10000	40	500-4000	[49]	Makespan, degree of imbalance, response time

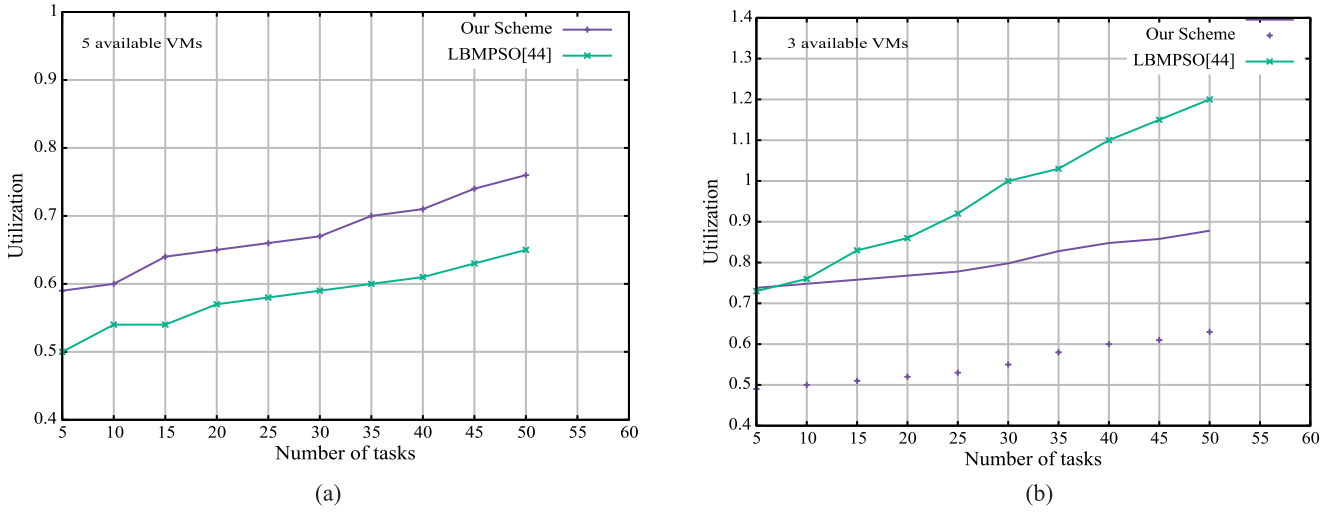

FIGURE 5. Comparison of utilization between our scheme and LBMPSO [42]. (a) for larger number of VMs, (b) for smaller number of VMs.

small number of tasks, this strategy approaches ours as the VMs are not heavily loaded and the makespan differences are only about 2-3%. When the number of tasks increases, the makespan difference rises to about 12%. Our scheme loads all the VMs proportionally in every time slot. As a result, the execution times of the VMs are approximate. On the other hand, a policy like the one implemented in HBF can increase the average response time and thus the makespan. This justifies the makespan difference of 12%, when the task numbers increase.

B. AVERAGE UTILIZATION

One of the objectives of the LBMPSO scheme is to enhance the VMs' utilization. Specifically, it uses the idea of distributing the tasks in such a way that, for each VM_k that is assigned N' tasks, the ratio

$$(\sum_{i=1}^{N'} ET_k)/Makespan$$

is the maximum possible. We experimented on two different scenarios to compare our work to the LBMPSO strategy: in the first scenario, the available VMs are 5. The average utilization for LBMPSO approaches 0.65 (see Fig.5(a)). When the number of VMs decreases, the LBMPSO increases the utilization, which reaches values > 1 (see Fig. 5(b)). This means that further task assignment can lead to long execution times, large makespan values and imbalances. our scheme maintains equal utilization values among the VMs for both scenarios and the utilization values increased smoothly as we kept adding new tasks. For 3 VMs, the utilization approached 0.9, but no over-utilization incurred because our scheme temporarily excludes over-utilized VMs.

C. DEGREE OF IMBALANCE

The degree of imbalance is given by

$$DI = \frac{T_{\max} - T_{\min}}{T_{avg}} \tag{29}$$

where $T_{\max}$ and $T_{\min}$ are the maximum and minimum execution times of all the tasks among all the VMs and T_{avg} is their average value. We have compared our strategy against the HBF scheme with VM pre-selection for 70 tasks. From Fig. 6, it is clear that our scheme achieves very low values for the degree of imbalance. The reason is that $T_{\max}$ and $T_{\min}$ approximate each other, as a result of our load balancing policy. The HBF strategy achieves good balancing, but there are cases where the response times (see the next paragraph) increase the maximum execution times, $T_{\max}$, thus the degree of imbalance.

D. RESPONSE TIME

In the last set of experiments, we compared the response times of our scheme, the HBF and the CPSO. Fig.7 shows the average response time comparisons. The HBF scheme is designed mainly to prevent overloading. As the number of tasks increase, the overloaded or balanced VMs are not assigned extra workload. Therefore, more tasks are waiting in the queue of the underloaded VMs, resulting in higher response times. The CPSO scheme employs a task allocation

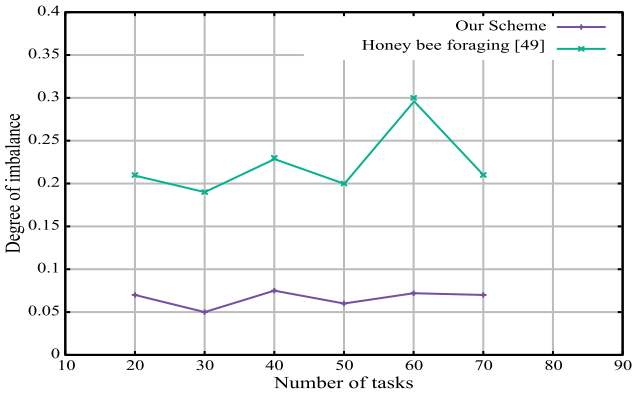

FIGURE 6. Comparison of the degree of imbalance between our scheme and the HBF scheme [47].

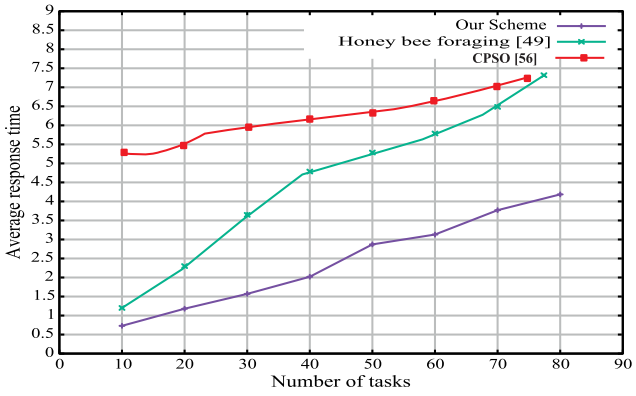

FIGURE 7. Comparison of the response times between our scheme, the HBF [47], and the CPSO [54] strategies.

algorithm which reduces the response time per task. The tasks are assigned between selected co-workers, but it has the drawback that, if there are no available workers to perform a set of tasks, then, these tasks can be queued until the potential workers are released from their current assigned task. This increases the response time. Our work proportionally the tasks to the VMs based on the expected processing capacity. Thus, the waiting times in the queues are lower, compared to the other schemes.

VI. CONCLUSION - FUTURE WORK

In this work we addressed the problem of load balanced task scheduling. Our system model is based on the Markov process model, which is combined with a simple fair task distribution scheme. From the balance state probabilities, we obtain the expected utilizations for the virtual machines (VM). Our fair task allocation policy is implemented on a time slot basis and in such a way, that the expected utilizations of all the VMs is equal. We proved that this scheme always guarantees load balancing. The proposed scheme is multi-objective in the sense that it enhances a number of important metrics: makespan, average utilization, degree of imbalance, and response time. Compared with three new state-of-the art schemes like the Load Balancing Modified Particle Swarm Optimization (LBMPSO), the Honey bee foraging (HBF) with VM pre-selection, and the and the community-based Particle Swarm Optimization (CPSO) scheme, our task allocation strategy was found to achieve better results regarding the aforementioned metrics.

An issue that needs to be further investigated is optimization. Our work was proven to guarantee load balancing, it provides good results for a variety of metrics under different distribution scenarios, but we still have not proved optimality. Also, we need to embed a strategy for VM availability. Currently, when a VM is not functioning properly, its tasks are transferred to the remaining VMs, using the fair distribution scheme. However, such situations cause larger total completion times, thus larger makespans and degrade the overall performance.

REFERENCES

[1] C. Liu, K. Li, and K. Li, "A game approach to multi-servers load balancing with load-dependent server availability consideration," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 1–13, Jan. 2021.
[2] S. Afzal and G. Kavitha, "Load balancing in cloud computing—A hierarchical taxonomical classification," *J. Cloud Comput.*, vol. 8, no. 1, pp. 1–24, Dec. 2019.
[3] P. S. Pillai and S. Rao, "Resource allocation in cloud computing using the uncertainty principle of game theory," *IEEE Syst. J.*, vol. 10, no. 2, pp. 637–648, Jun. 2016.
[4] M. Vanitha and P. Marikkannu, "Effective resource utilization in cloud environment through a dynamic well-organized load balancing algorithm for virtual machines," *Comput. Electr. Eng.*, vol. 57, pp. 199–208, Jan. 2017.
[5] K.-M. Cho, P.-W. Tsai, C.-W. Tsai, and C.-S. Yang, "A hybrid meta-heuristic algorithm for VM scheduling with load balancing in cloud computing," *Neural Comput. Appl.*, vol. 26, no. 6, pp. 1297–1309, 2015.
[6] M. Kumar, K. Dubey, and S. C. Sharma, "Elastic and flexible deadline constraint load balancing algorithm for cloud computing," in *Proc. 6th Int. Conf. Smart Comp. (ISCC)*, 2017, pp. 717–724.
[7] M. Ashouraei, S. N. Khezr, R. Benlamri, and N. J. Navimipour, "A new SLA-aware load balancing method in the cloud using an improved parallel task scheduling algorithm," in *Proc. IEEE 6th int. con. fut. internet things cloud (FiCloud)*, Aug. 2018, pp. 71–76.
[8] F. Tang, L. T. Yang, C. Tang, J. Li, and M. Guo, "A dynamical and load-balanced flow scheduling approach for big data centers in clouds," *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 915–928, Oct. 2018.
[9] C.-W. Ang and C.-K. Tham, "Analysis and optimization of service availability in a HA cluster with load-dependent machine availability," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 9, pp. 1307–1319, Sep. 2007.
[10] A. N. Singh and S. Prakash, "WAMLB: Weighted active monitoring load balancing in cloud computing," in *Big Data Analytics* (Advances in Intelligent Systems and Computing), vol. 654, V. Aggarwal, V. Bhatnagar, and D. Mishra, Eds. Singapore: Springer, 2018, pp. 677–685.
[11] S. G. Fatima, S. K. Fatima, S. A. Sattar, N. A. Khan, and S. Adil, "Cloud computing and load balancing," *Int. J. Adv. Res. Eng. Technol.*, vol. 10, no. 2, pp. 189–209, Mar. 2019.
[12] A. Rashid and A. Chaturvedi, "Cloud computing characteristics and services a brief review," *Int. J. Comput. Sci. Eng.*, vol. 7, no. 2, pp. 421–426, Feb. 2019.
[13] B. Mallikarjuna and D. A. K. Reddy, "The role of load balancing algorithms in next generation of cloud computing," *Control Syst.*, vol. 11, p. 20, Jul. 2019.
[14] U. Patel and M. H. Gupta, "A review of load balancing technique in cloud computing," *Int. J. Res. Anal. Rev.*, vol. 6, no. 2, p. 8, 2019.
[15] V. Arulkumar and N. Bhalaji, "Performance analysis of nature inspired load balancing algorithm in cloud environment," *J. Ambient Intell. Hum. Comput.*, vol. 2020, pp. 1–8, Jan. 2020.
[16] A. A. Prakash, V. Arul, and A. Jagannathan, "A look at of efficient and more suitable load balancing algorithms in cloud computing," *Int. J. Eng. Res. Comput. Sci. Eng.*, vol. 5, no. 4, p. 7, 2018.
[17] P. Kumar and R. Kumar, "Issues and challenges of load balancing techniques in cloud computing: A survey," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–35, Feb. 2019.

[18] K. Lakhwani, "An extensive survey on load balancing techniques in cloud computing," *J. Gujarat Res. Soc.*, vol. 21, no. 10, pp. 309–319, 2019.
[19] M. A. Shahid, N. Islam, M. M. Alam, M. M. Su'ud, and S. Musa, "A comprehensive study of load balancing approaches in the cloud computing environment and a novel fault tolerance approach," *IEEE Access*, vol. 8, pp. 130500–130526, 2020.
[20] M. Elrotub and A. Gherbi, "Virtual machine classification-based approach to enhanced workload balancing for cloud computing applications," *Proc. Comput. Sci.*, vol. 130, pp. 683–688, Jan. 2018.
[21] X. Sui, D. Liu, L. Li, H. Wang, and H. Yang, "Virtual machine scheduling strategy based on machine learning algorithms for load balancing," *EURASIP J. Wireless Commun. Netw.*, vol. 2019, no. 1, p. 160, Dec. 2019.
[22] U. K. Lilhore, S. Simaiya, K. Guleria, and D. Prasad, "An efficient load balancing method by using machine learning-based VM distribution and dynamic resource mapping," *J. Comput. Theor. Nanosci.*, vol. 17, no. 6, pp. 2545–2551, Jun. 2020.
[23] A. Ghasemi and A. T. Haghighat, "A multi-objective load balancing algorithm for virtual machine placement in cloud data centers based on machine learning," *Computing*, vol. 102, no. 9, pp. 2049–2072, Sep. 2020.
[24] S. Garg, D. D. V. Gupta, and R. K. Dwivedi, "Enhanced active monitoring load balancing algorithm for virtual machines in cloud computing," in *Proc. Int. Conf. Syst. Model. Advancement Res. Trends (SMART)*, 2016, pp. 339–344.
[25] S. Sotiriadis, N. Bessis, C. Amza, and R. Buyya, "Elastic load balancing for dynamic virtual machine reconfiguration based on vertical and horizontal scaling," *IEEE Trans. Services Comput.*, vol. 12, no. 2, pp. 319–334, Mar./Apr. 2019.
[26] M. Jodayree, M. Abaza, and Q. Tan, "A predictive workload balancing algorithm in cloud services," *Proc. Comput. Sci.*, vol. 159, pp. 902–912, Jan. 2019.
[27] K. Li, C. Liu, K. Li, and A. Y. Zomaya, "A framework of price bidding configurations for resource usage in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2168–2181, Aug. 2016.
[28] C. Liu, K. Li, C. Xu, and K. Li, "Strategy configurations of multiple users competition for cloud service reservation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 508–520, Feb. 2016.
[29] X. Xu and H. Yu, "A game theory approach to fair and efficient resource allocation in cloud computing," *Math. Problems Eng.*, vol. 2014, pp. 1–14, Apr. 2014.
[30] F. Zegrari, A. Idrissi, and H. Rehioui, "Resource allocation with efficient load balancing in cloud environment," in *Proc. BDAW*, Nov. 2016, pp. 1–7.
[31] J. Chen, T. Du, and G. Xiao, "A multi-objective optimization for resource allocation of emergent demands in cloud computing," *J. Cloud Comput.*, vol. 10, no. 1, p. 20, Dec. 2021.
[32] J. Kumar, A. K. Singh, and A. Mohan, "Resource-efficient load-balancing framework for cloud data center networks," *ETRI J.*, vol. 43, no. 1, pp. 53–63, Feb. 2021.
[33] A. Singh, D. Juneja, and M. Malhotra, "Autonomous agent based load balancing algorithm in cloud computing," *Proc. Comput. Sci.*, vol. 45, pp. 832–841, Jan. 2015.
[34] Z. Xiao, Z. Tong, K. Li, and K. Li, "Learning non-cooperative game for load balancing under self-interested distributed environment," *Appl. Soft Comput.*, vol. 52, pp. 376–386, Mar. 2017.
[35] M. Lavanya and V. Vaithiyanathan, "Load prediction algorithm for dynamic resource allocation," *Indian J. Sci. Technol.*, vol. 8, no. 35, pp. 1–4, Dec. 2015.
[36] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang, "Cloud task scheduling based on load balancing ant colony optimization," in *Proc. 6th Annu. Chinagrid Conf.*, Aug. 2011, pp. 3–9.
[37] D. B. L. D. and P. Venkata Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Appl. Soft Comput.*, vol. 13, no. 5, pp. 2292–2303, May 2013.
[38] V. Polepally and K. S. Chatrapati, "Dragonfly optimization and constraint measure-based load balancing in cloud computing," *Cluster Comput.*, vol. 22, pp. 1–13, Jan. 2017.
[39] S. K. Vasudevan, S. Anandaram, A. J. Menon, and A. Aravinth, "A novel improved honey bee based load balancing technique in cloud computing environment," *Asian J. Inf. Technol.*, vol. 15, no. 9, pp. 1425–1430, 2016.
[40] A. Gupta, H. S. Bhadauria, and A. Singh, "Load balancing based hyper heuristic algorithm for cloud task scheduling," *J. Ambient Intell. Hum. Comput.*, vol. 12, no. 6, pp. 5845–5852, Jun. 2021.
[41] B. Mondal and A. Choudhury, "Simulated annealing (SA) based load balancing strategy for cloud computing," *Int. J. Comp. Sci. Inf. Technol.*, vol. 6, no. 4, pp. 3307–3312, 2015.
[42] A. Pradhan and S. K. Bisoy, "A novel load balancing technique for cloud computing platform based on PSO," *J. King Saud Univ. Comput. Inf. Sci.*, pp. 677–681, Oct. 2020, doi: 10.1016/j.jksuci.2020.10.016.
[43] M. Adhikari and T. Amgoth, "Heuristic-based load-balancing algorithm for IaaS cloud," *Future Gener. Comput. Syst.*, vol. 81, pp. 156–165, Apr. 2017.
[44] S. S. Rajput and V. S. Kushwah, "A genetic based improved load balanced min-min task scheduling algorithm for load balancing in cloud computing," in *Proc. 8th Int. Conf. Comput. Intell. Commun. Netw. (CICN)*, Dec. 2016, pp. 677–681.
[45] D. A. Shafiq, N. Z. Jhanjhi, A. Abdullah, and M. A. Alzain, "A load balancing algorithm for the data centres to optimize cloud computing applications," *IEEE Access*, vol. 9, pp. 41731–41744, 2021, doi: 10.1109/ACCESS.2021.3065308.
[46] A. Asghari, M. K. Sohrabi, and F. Yaghmaee, "Task scheduling, resource provisioning, and load balancing on scientific workflows using parallel SARSA reinforcement learning agents and genetic algorithm," *J. Supercomput.*, vol. 77, no. 3, pp. 2800–2828, Mar. 2021.
[47] F. Ebadifard, S. M. Babamir, and S. Barani, "A dynamic task scheduling algorithm improved by load balancing in cloud computing," in *Proc. 6th Int. Conf. Web Res. (ICWR)*, Apr. 2020, pp. 177–183.
[48] P. Neelima and A. R. M. Reddy, "An efficient load balancing system using adaptive dragonfly algorithm in cloud computing," *Cluster Comput.*, vol. 23, no. 4, pp. 2891–2899, Dec. 2020.
[49] W. Tan, L. Zhao, B. Li, L. D. Xu, and Y. Yang, "Multiple cooperative task allocation in group-oriented social mobile crowdsensing," *IEEE Trans. Services Comput.*, early access, Jun. 2, 2021, doi: 10.1109/TSC.2021.3086097.
[50] M. M. Rad, A. M. Rahmani, A. Sahafi, and N. N. Qader, "Social Internet of Things: Vision, challenges, and trends," *Human-centric Comput. Inf. Sci.*, vol. 10, no. 1, pp. 1–40, Dec. 2020.
[51] A. Khanfor, R. Hamadi, H. Ghazzai, Y. Yang, M. R. Haider, and Y. Massoud, "Computational resource allocation for edge computing in social Internet-of-Things," in *Proc. IEEE 63rd Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2020, pp. 233–236, doi: 10.1109/MWSCAS48704.2020.9184663.
[52] P. Wu, L. Pan, and C. Zheng, "Mining set of interested communities with limited exemplar nodes for network based services," *IEEE Trans. Services Comput.*, vol. 14, no. 4, pp. 1138–1151, Jul. 2021, doi: 10.1109/TSC.2018.2864147.
[53] L. Zhao, W. Tan, N. Xie, and L. Huang, "An optimal service selection approach for service-oriented business collaboration using crowd-based cooperative computing," *Appl. Soft Comput.*, vol. 92, Jul. 2020, Art. no. 106270.
[54] R. Estrada, R. Mizouni, H. Otrok, A. Ouali, and J. Bentahar, "A crowdsensing framework for allocation of time-constrained and location-based tasks," *IEEE Trans. Services Comput.*, vol. 13, no. 5, pp. 769–785, Sep. 2020, doi: 10.1109/TSC.2017.2725835.
[55] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "Optimal load distribution for the detection of VM-based DDoS attacks in the cloud," *IEEE Trans. Services Comput.*, vol. 13, no. 1, pp. 114–129, Jan. 2020, doi: 10.1109/TSC.2017.2694426.
[56] A. Khanfor, H. Ghazzai, Y. Yang, M. R. Haider, and Y. Massoud, "Automated service discovery for social Internet-of-Things systems," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Seville, Spain, Oct. 2020, pp. 1–5.
[57] W. J. Gordon and G. F. Newell, "Closed queuing systems with exponential servers," *Oper. Res.*, vol. 15, no. 2, pp. 254–265, Apr. 1967.

STAVROS SOURAVLAS (Member, IEEE) is currently an Assistant Professor in computer architecture with the Department of Applied Informatics, School of Information Sciences, University of Macedonia. His research interests include scheduling of parallel and distributed computing systems, big data stream scheduling, cloud computing, and systems modeling and simulation. He has published more than 40 papers in peer reviewed journals and conferences, including six papers in IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS, and IEEE ACCESS.

SOFIA D. ANASTASIADOU is currently a Professor in statistics and research methodology with the Department of Midwafery, School of Health Sciences, University of Western Macedonia. Her research interests include specialization is qualitative and quantitative methods in social sciences, applied statistics, implicative statistic analysis, multivariate statistical analysis, biostatistics, meta-analysis, structural equation models, analyse des donnees, and big data.

NICOLETA TANTALAKI received the M.Sc. degree in business informatics, the M.Sc. degree in information and communication technology in education, and the Ph.D. degree in parallel and distributed processing of big data streams from the Department of Applied Informatics, School of Information Sciences, University of Macedonia. She is currently a Postdoctoral Researcher with the Department of Informatics, Aristotle University of Thessaloniki. Her research interests include the field of parallel and distributed computing systems, with a special emphasis on big data systems and scheduling algorithms for big data stream processing. She has received over ten research and education grants from various agencies, such as the State Scholarship Foundation, the Onassis Foundation, and Google.

STEFANOS KATSAVOUNIS is currently an Associate Professor with the Department of Production Engineering & Management, Democritus University of Thrace, Greece. His research interests include revolve around scheduling, RCMPSP, project management, graph theory & modeling, heuristics for NP-hard problems in social networks, transportation & supply chain management, grey analysis, and data processing in material science.