Simulation Modelling Practice and Theory 98 (2020) 101982

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

A Scheduling Algorithm for a Fog Computing System with Bag-of-Tasks Jobs: Simulation and Performance Evaluation

Dimitrios Tychalas*, Helen Karatza

Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece

ARTICLE INFO

Keywords:
Cloud
Cluster
Containers
Smartphones
Raspberry
Bag-of-Tasks

ABSTRACT

Cloud computing is evolving in such a way that it is apparent that the future of High Performance Computing (HPC) lies in interconnected heterogeneous systems. Moreover, many of today's needs prefer resources that are diverse in geographic distribution and are close to their location. Hence, Fog Computing was “born” in order to better meet all the demands of today's computational needs. Compared to Cloud Computing, Fog Computing extends it by bringing computing nearer to users, enhancing location-based services and utilizing every available resource.

The development and simulation of a Fog Computing System based on Grids, Smartphones, Raspberries and Cloud (Virtual Machines or Containers) is presented in this paper.

This work aims at studying the possibilities of using every resource that is available in order to reduce total expenses under a Bag-of-Tasks workload model. Simulation results show that by combining every available resource we can reduce the costs, while the mean response time is not increasing drastically.

1. Introduction

The number of areas using cloud computing is rapidly increasing; from complex biological and aerospace algorithms to a number of fields as authors [1–5,29] present. Furthermore, the majority of companies choose Cloud Computing to run their software to reduce costs and energy.

HPC infrastructure is divided into several categories— clouds, grids, clusters — with a lot of different sub-categories [6]. So, choosing the best solution each time, is quite difficult for a company which its needs are increasing every year. It must also be considered that if the "best" solution is chosen, the cost of scaling it up is exponentially increased. In addition, real - time data processing is required from more applications every year. The authors in [7] have stated that uploading all the data to a remote computational site (e.g. VM), the execution time can be significantly increased. Additionally, if the VM is turned off, it can result in an increased execution time because the jobs have to wait until the VM is switched on which is a huge waste of time. As a result, containers were introduced. Containers share the host OS kernel – binaries and libraries – instead of a virtualized “real” operating system. This makes containers really “light” and it takes just seconds to start. In addition, the number of smart devices is increasing exponentially so that when they are idle, we can use their computing power to reduce the amount of data that are transferred to a

* Corresponding author.
E-mail addresses: dtychala@csd.auth.gr (D. Tychalas), karatza@csd.auth.gr (H. Karatza).

https://doi.org/10.1016/j.simpat.2019.101982
Received 15 July 2019; Received in revised form 25 August 2019; Accepted 3 September 2019
Available online 04 September 2019
1569-190X/

remote resource and decrease the execution time of an application.

The are many types of available infrastructures, and each type has its advantages and disadvantages. Moreover, the options that exist today cannot combine all the available resources of different available computational platforms for simultaneously running applications - experiments, thus scientists have to use each one separately [7,8,29,30].

There are many types of workloads submitted to HPC but the one that best reflects a real case scenario is the Bag of Tasks [27]. BoT is a set of tasks that can run in any order and they do not need to communicate with each other [9–12]. BoT applications are used in a wide range of situations despite their simplicity, including data mining, heavily searches (such as logging analytics), sweeping parameters, simulations, computer imaging, bioinformatics and fractal calculations. In addition, BoT applications run effectively over extensively distributed computing grids due to the independence of their tasks, as illustrated by SETI@home [28]. One can claim that BoT applications are the most suitable applications for computer grids, where communication can rapidly turn into a bottleneck.

In this paper, we introduce an algorithm that can use a organization's existing infrastructure and combine low - cost computational resources, VMs and mobile phones to reduce costs under a BoT workload. The algorithm does not require from the scheduler to have any knowledge about the tasks or their execution time. This strategy is based on the fact that this data can be difficult to obtain, particularly in extremely unstable distributed systems. In order to investigate the significance of using Containers in the algorithm, we run our simulation once using Virtual Machines and once using Containers, so that comparison can be made.

The following is the organization of this article. In Section 2, Fog Computing's basic architectures, requirements and the advantages over Cloud Computing are presented. In Section 3, related work is discussed and the state of the art paradigms are shown. In Section 4, our system is presented, our algorithm is analyzed and our simulation results are discussed. In Section 5, conclusions are presented and guidance for future research is provided.

2. Fog computing

"Fog Computing" is a term that Cisco Systems introduce as a new network model to reduce data transfer within the IoT applications [13,14]. As authors stated in [15] “Fog is nothing but Cloud that is nearer to the ground”. Therefore Cloud Computing performed at the edge of the network of end users is known as Fog Computing. Fog Computing exists between Cloud and end user devices [13]. This leads to higher service quality in terms of delay, power consumption and reduced Internet data traffic. Many applications require mobility, low latency, and location awareness, and they can be supported by Fog Computing as shown in Fig. 1.

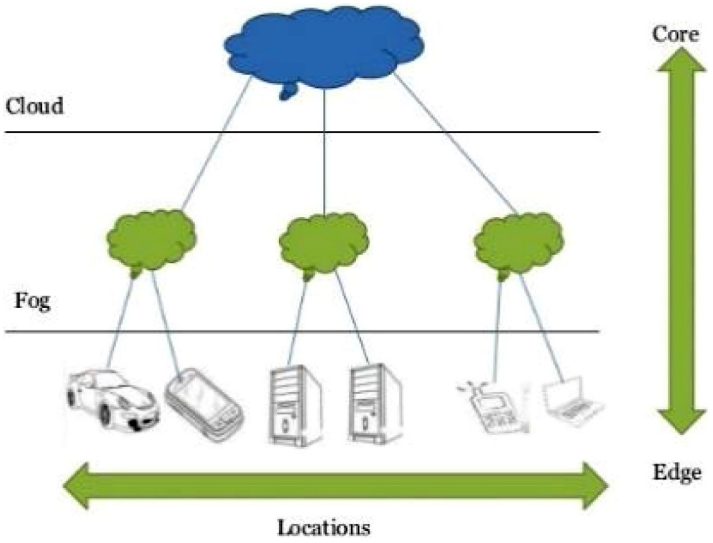


Fig. 1. Fog computing.

Cloud and Fog are created with similar standards; however, Fog Computing is nearer to client. The qualities which make Fog a non-trivial cloud extension, as discussed in [15], are shown below:

1. Geographical distribution
2. Low latency

3. Edge location
4. Location awareness
5. Very large number of nodes due to the wide geo-distribution
6. Support for mobility
7. Real-time interactions
8. Heterogeneity
9. Interoperability and federation
10. Support for on-line analytics and interplay with Cloud

There are many advantages associated with Fog Computing, including the following, as presented by authors in [16]:

1. Low latency requirement
2. Suitable for IoT tasks and queries
3. Scalability
4. Reduction of network traffic

3. Related work

In this section, we discuss the algorithms that are available for resource management and scheduling based on the current literature [9–12]. Many scientists have contributed to efficient scheduling, resource management and load balancing algorithms but there is still room to improve them as these are NP-Complete / NP-Hard problems.

In [17] authors introduced BaTS, a "budget-constrained scheduler for Bag-of-Tasks applications". BaTS does not require information about task execution times, instead it uses statistical methods to find the best solution. Moreover, BaTS monitors the progress of the tasks and dynamically reconfigures the set of the machines as required. They ran several tests and each test used two clouds with different price-performance ratio. Each test was run twice, firstly with the Round Robin algorithm and secondly with the BaTS. Their method produced schedules for a guaranteed budget, albeit slower than Round Robin.

Authors in [18] presented a new heuristic algorithm, Qsufferage, which considered the influence of input data repositories' location and enhances the response ratio of each task, as well as reduces the makespan of the Bag-of-Tasks application. They tested it among other well-known heuristic algorithms and the results indicated that the size of input data of each task influences the performance over-all, but Qsufferage constantly outperformed all the other algorithms.

Authors in [19] proposed a set of knowledge-free scheduling algorithms that could deal with BoTs on Desktop Grids. Specifically, they tested the following algorithms: i) *First Come First Served - Exclusive* (FCFS-Excl), ii) *First Come First Served - Shared* (FCFS-Share), iii) *Round Robin* (RR), iv) *Round Robin - No Replica First* (RR-NRF) & v) *Longest Idle* (LongIdle). The results indicate that among the proposed strategies there is no clear winner, but the RR algorithms were slightly better for medium and heavy availability configurations. As a result, in this paper we use the Round Robin algorithm as base comparison.

Authors in [20] showed that using a cloud-based Fog system could lead to an enhanced experience. They developed a simulation based on the Discrete Event System specification. However, they did not use any algorithm for load balancing which could minimize the use of Cloud and subsequent costs.

We simulated a Fog computing system in a previous work of ours [26] that is different from the current research system model. Bag-of-Tasks workload was not used in that work. Additionally, containers were not used as an alternative to the VMs. Thus, the algorithm that used for simulating this work is completely different from the previous one.

As compared to the existing related research, we simulate and propose an algorithm that both meet Fog's Computing requirements and minimize costs and running times. More specific:

- We examine extremely heterogeneous resources,
- Load balancing is first viewed from the cost perspective and then from the performance perspective.
- The proposed algorithm can adapt to the resources in a rapidly changing environment,
- Each node is characterized by its costs and computational capabilities,

To our knowledge, this is the only research that uses this system and workload models to study Fog Computing.

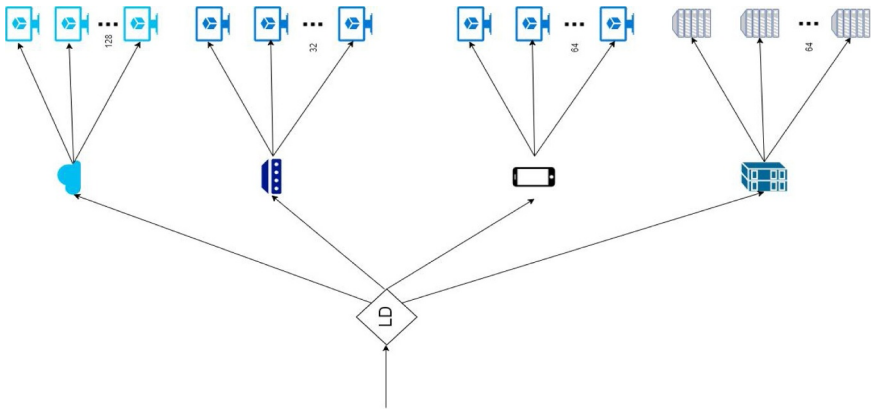


Fig. 2. The system.

4. The model

Our system simulates a Fog Computing System with Bag-of-Tasks Jobs. There is a total of 128 Vms/Containers, 32 low-cost devices (e.g. Raspberries), 64 Smartphones and 64 Desktop PCs (Fig. 2). A load balancer selects each system (virtual machine or container, raspberry, smartphone or computer) for each sub-system (Cloud, Raspberries, Smartphones, Desktop PC) and each resource has its own queue. Both systems and queues are empty at the start of the simulation. Since multiple users use the system, we are studying, to run their applications, the application's jobs dynamically arrive at the Local Dispatcher. Each job is a BoT, consisting of a set $J = \{t_1, t_2, \ldots, t_m\}$ of m independent parallel tasks, without precedence constraints or inter-task communication. Each task is assigned to a sub-system queue according to the routing policy employed. If the sub-system is busy then the task is stored in its queue. Upon completion of execution, a task awaits completion of execution at the join point for sibling tasks of the same job. We assume that in our model:

- Jobs are not mutually dependent
- Every sub-system can execute all jobs and their tasks
- Ten percent of the jobs are served from the Desktop PCs
- If a resource is switched off (Raspberry or VM/Container) then it waits 5 times its service time to switch on and be ready for use. In the case of Containers, it has to wait only 1 time its service time.

Independent and identically distributed random variables, which follow the exponential distribution, are used as interarrival times. The same distribution is followed by jobs service demands.

4.1. Job allocation policies

Job allocation refers to the system selection where the job is to be routed. Local Dispatcher (LD) is effectively distributing jobs in heterogeneous sub-systems [22]. We are using the following policies:

- All jobs are executed on the local cluster, if its utilization is under a threshold θ1.
- If the utilization of smartphones is below a threshold θ2, then the jobs are executed on them.
- The system which has the lowest load is selected.
- The sub-system which has the shortest queue is selected
- Firstly, we use one low-energy machine and one Virtual machine / Container
  - In case of exceeding the utilization of the above systems by a threshold θ3, one more resource is used, if is available.
  - If the utilization drops under a threshold θ4, one resource is scheduled to turn off after executing of all its tasks.

Algorithm 1
The proposed algorithm in pseudo code.

```
Job = new_job_arrives();
    foreach task In Job.Tasks{
      Local_cluster_utilization = Calculate_Utilization('local_cluster')
      Smartphone_utilization = Calculate_Utilization('smartphone')
      Cloud_utilization = Calculate_Utilization('cloud')
      Raspberries_utilization = Calculate_Utilization('raspberries')
      Cloud_work_load = Calculate_work_load('cloud')
      Raspberries_work_load = Calculate_work_load('raspberries')
      VMs_turned_on = Calculate_turned_on('cloud')
      Raspberries_turned_on = Calculate_turned_on('cloud')
      if(Local_cluster_utilization < theta1){
        DesktopPC = Select_resource('local_cluster', 'shortest_queue');
        Execute_task(DesktopPC, task);
      }
      elseif(Smartphone_utilization < theta2){
        Smartphone = Select_resource('smartphone', 'shortest_queue');
        Execute_task(Smartphone, task);
      }
        else{
          if(Cloud_work_load < Raspberries_work_load){
            if(Cloud_utilization > theta3){
              if(VMs_turned_on < 128){
                Turn_on('cloud')
              }
            }
            else if(Cloud_utilization < theta4){
              if(VMs_turned_on > 1){
                Turn_off('cloud');
              }
            }
            VM = Select_resource('cloud', 'shortest_queue');
            Execute_task(VM, task);
        }
        else{
          if(Raspberries_utilization > theta3){
            if(Raspberries_turned_on< 32){
              Turn_on('raspberries')
            }
          }
          else if(Raspberries_utilization < theta4){
            if(Raspberries_turned_on > 1){
              Turn_off('raspberries');
            }
          }
          Raspberry = Select_resource('raspberries', 'shortest_queue');
          Execute_task(Raspberry, task);
        }
      }
    }
```

4.2. Performance metrics

Tables 1A and 1b presents the performance metrics and parameters that were used in this simulation. We define a random Bag-of-Task response time, r_i, as the period from the arrival of this job in the system to the completion of all the tasks of this job. To put it another way, a job's response time is equal to a server queue's delay plus service time. The mean response time for RT is as follows [27]:

Table 1A
Performance parameters.

Performance parameter	Description
RT	Mean response time of jobs
U	Mean system utilization

Table 1B
Description of inputs.

Input	Description
θ1	Utilization threshold of Cluster
θ2	Utilization threshold of Smartphone
θ3	Max utilization threshold
θ4	Min utilization threshold
μ	Mean service rate of BoT tasks
1/μ	Mean service time of BoT tasks
λ	Mean arrival rate
1/λ	Mean inter-arrival time of job

$$RT = \frac{1}{m} \times \sum_{i=1}^{m} r_i$$

where m refers to the total number of jobs processed and r_i to the response time of the i_{th} job.

The value of θ1 was set at 0.7 to use the local cluster as our primary resource and reduce costs and execution time. The value θ2 was set to 0.3 so the smartphones not overused, even if they are idle. To avoid overloading the system, the value of θ3 was set to 0.7. This is crucial because the mean response time will be increased if a system is overloaded. Furthermore, θ4 has been set to 0.5 to reduce costs when a Raspberry or Virtual Machine is not used.

4.3. Input parameters

The model is implemented using C programming language with discrete event simulation [21,23]. Based on this technique, the state of the system only changes when an event takes place. An event may be a job arrival to the system, or a task assignment to a queue or a task arrival. There are no changes in the system's state between events. When the 60000th job is completed, each simulation experiment ends. This simulation length was considered long enough to derive results, as experiments show that longer runs do not significantly affect the results. Table 2 shows the mean service rates per subsystem and thresholds that are used for this simulation.

We consider that the service time of a cloud or cluster job also includes the needed uploading time. Therefore, the service time for smartphones is smaller. The number of tasks that can be served in a single time unit in each subsystem depends on the processing unit's mean service rate.

Cloud has a total of 128 servers (VMs or containers). Therefore, at full load, 128 (128/1) tasks can be executed in one-time unit.

Table 2
Input parameters of sub-systems.

System	Number	Mean service time (1/μ)	Parallel Jobs [n/((n+1)/2)]	Thresholds
Cloud	128	1	1.77 (n=8)	θ3 = 0.7, θ4 = 0.5
Raspberry PIs	32	2	1.77 (n=8)	θ3 = 0.7, θ4 = 0.5
Cluster	64	1	1.77 (n=8)	θ1 = 0.7
Smartphones	64	0.5	1.77 (n=8)	θ2 = 0.3

Table 3
Load cases.

L = Low with Load Balancing	LC = Low with Load Balancing on containers	LR = Low Round Robin
LRC = Low Round Robin on containers	LRL = Low with Load Balancing and Round Robin	LRLC = Low with Load Balancing and Round Robin on Containers
M = Medium with Load Balancing	MC = Medium with Load Balancing on containers	MR = Medium Round Robin
MRC = Medium Round Robin on containers	MRL = Medium with Load Balancing and Round Robin	MRLC = Medium with Load Balancing and Round Robin on Containers
H = High with Load Balancing	HC = High with Load Balancing on containers	HR = High Round Robin
HRC = High Round Robin on containers	HRL = High with Load Balancing and Round Robin	HRLC = High with Load Balancing and Round Robin on Containers

Table 4
Input parameters of cases.

	L	LC	LR	LRC	LRL	LRLC
Load	31,25%	31,25%	31,25%	31,25%	31,25%	31,25%
	M	MC	MR	MRC	MRL	MRLC
Load	44,64%	44,64%	44,64%	44,64%	44,64%	44,64%
	H	HC	HR	HRL	HRL	HRLC
Load	78,12%	78,12%	78,12%	78,12%	78,12%	78,12%

Table 5A
Results.

Light Load	L	LC	LR	LRC	LRL	LRLC
Average jobs in system	65,51	47,50	34,39	34,39	55,13	40,00
System utilization	0,26	0,26	0,25	0,25	0,25	0,26
Cloud utilization	0,01	0,01	0,04	0,04	0,03	0,03
Raspberries utilization	0,27	0,28	0,07	0,07	0,17	0,20
Local PCs utilization	0,70	0,70	0,70	0,70	0,70	0,70
Smartphones utilization	0,30	0,30	0,30	0,30	0,30	0,30
Mean response time	7,28	5,24	4,17	4,17	7,73	5,64
Mean task average delay in queue	0,46	0,47	0,48	0,48	0,59	0,58
Throughput	19,92	19,93	19,89	19,89	19,93	19,97
Cloud cost	6,80	5,53	128,00	128,00	11,43	9,52
Raspberry PIs cost	14,82	14,51	32,00	32,00	10,49	11,77
Cloud cost %	5,31	4,32	100,00	100,00	8,93	7,43
Raspberry PIs cost %	46,30	45,36	100,00	100,00	32,77	36,77

Table 5B
Results.

Medium Load	M	MC	MR	MRC	MRL	MRLC
Average jobs in system	83,76	83,76	77,63	77,63	132,55	85,81
System utilization	0,42	0,42	0,40	0,40	0,39	0,40
Cloud utilization	0,27	0,27	0,31	0,31	0,33	0,29
Raspberries utilization	0,71	0,71	0,36	0,36	0,25	0,39
Local PCs utilization	0,70	0,70	0,70	0,70	0,70	0,70
Smartphones utilization	0,30	0,30	0,30	0,30	0,30	0,30
Mean response time	3,71	3,71	3,49	3,49	6,69	4,37
Mean task average delay in queue	0,55	0,55	0,70	0,70	0,84	0,82
Throughput	28,51	28,51	28,44	28,44	28,45	28,44
Cloud cost	56,32	56,32	128,00	128,00	80,42	73,05
Raspberry PIs cost	28,82	31,78	32,00	32,00	21,46	28,56
Cloud cost %	43,99	44,00	100,00	100,00	62,83	57,07
Raspberry PIs cost %	90,04	99,33	100,00	100,00	67,05	89,26

There are 32 Raspberries in total, so 16 (32/2) tasks can be completed at full load in one-time unit. There are 64 Desktop PCs, resulting in a simultaneous service of 64 (64/1) tasks. There are 64 smartphones, resulting in simultaneous serving of 128 (64/0.5) tasks. This means we should choose a $\lambda < 336$ to preserve the system's stability. In addition, each BoT job is composed of n parallel tasks. For $n = 8$, the average number of tasks per parallel job is 4.5 ((n + 1)/2). As a result, it is necessary to select a $\lambda <$ (336/4.5), $\lambda < 74$.

As Table 3 shows, six cases were studied for the mean job interarrival time:

Table 5C
Results.

Heavy Load	H	HC	HR	HRC	HRL	HRLC
Average jobs in system	103,52	103,64	77,91	77,91	95,91	75,87
System utilization	0,77	0,77	0,76	0,76	0,76	0,76
Cloud utilization	0,99	0,99	1,00	1,00	0,96	0,99
Raspberries utilization	0,97	0,98	0,89	0,89	1,00	0,91
Local PCs utilization	0,70	0,70	0,70	0,70	0,70	0,70
Smartphones utilization	0,30	0,30	0,30	0,30	0,30	0,30
Mean response time	4,66	4,59	4,71	4,71	6,43	4,95
Mean task average delay in queue	1,16	1,15	1,29	1,29	1,30	1,30
Throughput	49,74	49,74	49,71	49,71	48,93	49,68
Cloud cost	125,73	125,71	128,00	128,00	122,51	127,14
Raspberry PIs cost	31,34	31,48	32,00	32,00	31,93	29,33
Cloud cost %	98,23	98,21	100,00	100,00	95,71	99,33
Raspberry PIs cost %	97,93	98,38	100,00	100,00	99,78	91,64

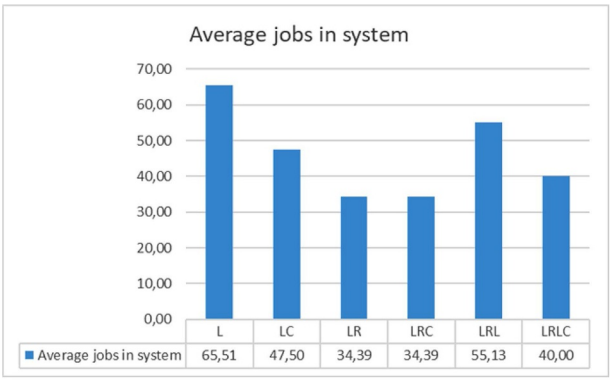
Firstly, we evaluated the performance of the model, which includes containers to minimize costs and reduce the mean response time. Then, we simulated the model, which consists of VMs instead of containers. Finally, the communication protocol is the same in all cases as our purpose is to investigate how the available resources can be used together to increase overall performance. We consider that in all cases the best protocol is used. Table 4 presents all the cases.

Algorithm 2
Weighted Round Robin Algorithm ins pseudo code.

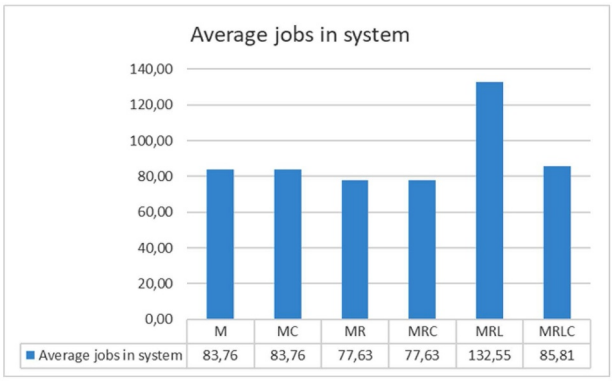
```
#128 servers μ=1, 128 / 16 = 8 (8)
    #32 Raspberries μ=2, 16 / 16 = 1 (9)
    #64 PCs μ=1, 64/16 = 4 (13)
    #64 Smartphones μ=0.5, 128/16 = 8 (21)
    # progs_served = 0;
    Job = new_job_arrives();
    foreach Task In Job.Tasks{
      round_robin = progs_served%21 + 1;
      if(round_robin <= 8){
        VM = Select_resource('cloud', 'shortest_queue');
        Execute_task(VM, task);
      }
      else if(round_robin == 9){
        Raspberry = Select_resource('raspberry', 'shortest_queue');
        Execute_task(Raspberry, task);
      }
      else if(round_robin <= 13){
        DesktopPC = Select_resource('raspberry', 'shortest_queue');
        Execute_task(DesktopPC, task);
      }
      else{/* 14-21 */
        Smartphone = Select_resource('raspberry', 'shortest_queue');
        Execute_task(Smartphone, task);
      }
      progs_served++;
    }
```

4.4. Simulation results

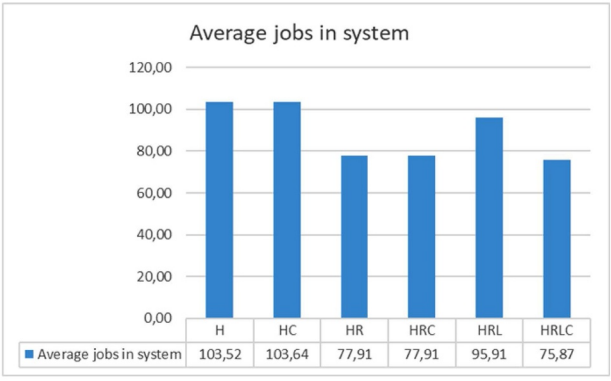
The simulation shows the performance of scheduling a Bag - of - Tasks workload in a Fog computing system. There is a Load Balancer that monitors the utilization and chooses when to switch on or off a VM/Container of Raspberry in order to reduce costs such as: the cost of energy consumption and the cost of using a cloud system. The Desktop PCs are turned on at all times and as a result we cannot reduce power consumption, but when the system is not loaded we can use its computational power to reduce the usage of the other resources. The following figures show comparisons of different cases. All experiments were conducted with 3 different average arrival rates: 23, 32 & 58, representing load of approximately 0.31 (31 %), 0.44 (44 %) and 0.78 (78 %) respectively, therefore the figures were split to group of threes and the explanation is followed. Tables 5A, 5B, 5C presents, as described above, an average of several runs of all results for each case.



A



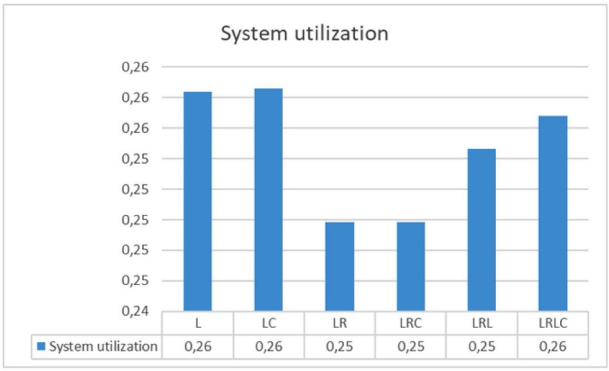
B



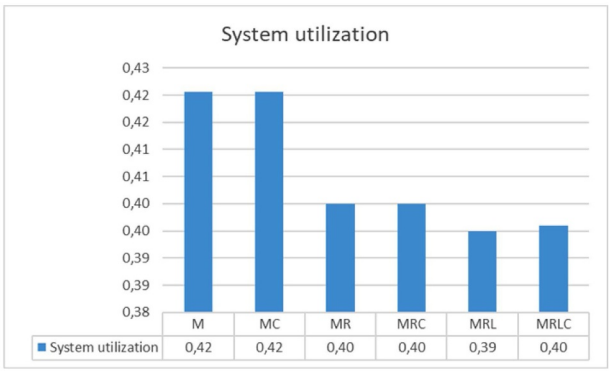
C

Fig. 3. (A) Average number of jobs in System (Low Load). (B) Average number of jobs in System (Medium Load). (C) Average number of jobs in System (High Load).

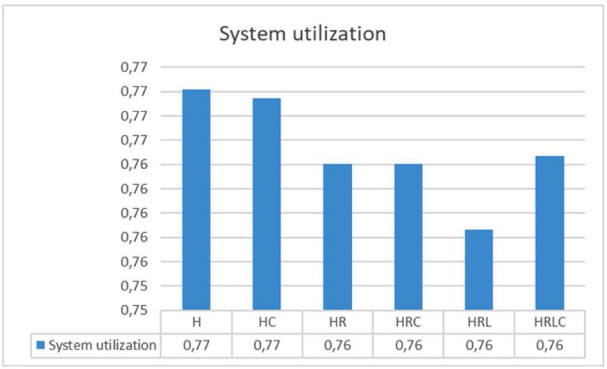
Fig. 3 depicts the system's average number of jobs. It can be concluded that it is slightly increased by using our load balancer algorithm, particularly in the High Load case as we start with one VM/Container and one Raspberry turned on. As a result, jobs must wait until the resource is switched on as stated in our hypotheses. In the Light and Medium load cases, the number is higher because the algorithm switches off Raspberries and VMs/Containers, when utilization is not high, to reduce costs. Moreover, it is shown that by using containers the average number of jobs in the system decreases. That is something expected, since the containers can start up more quickly than the VMs [24,25].



A



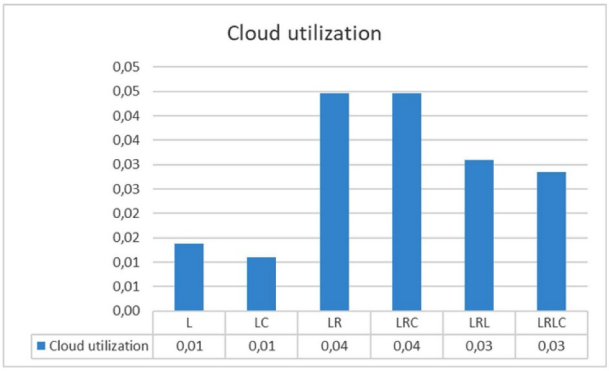
B



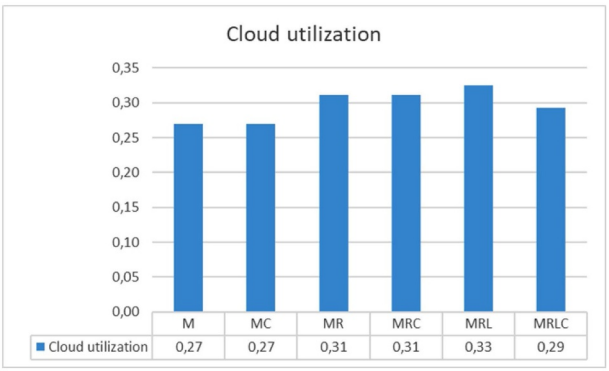
C

Fig. 4. (A) System Utilization (Low Load). (B) System Utilization (Medium Load). (C) System Utilization (High Load).

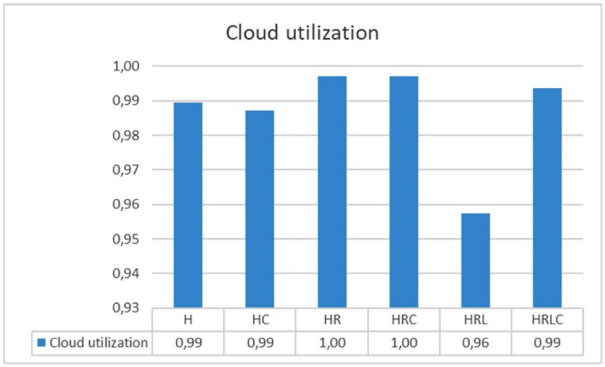
Fig. 4 shows how the proposed algorithm affects the utilization of the entire system. There are no differences regarding the utilization in the cases described above, something that was expected.



A



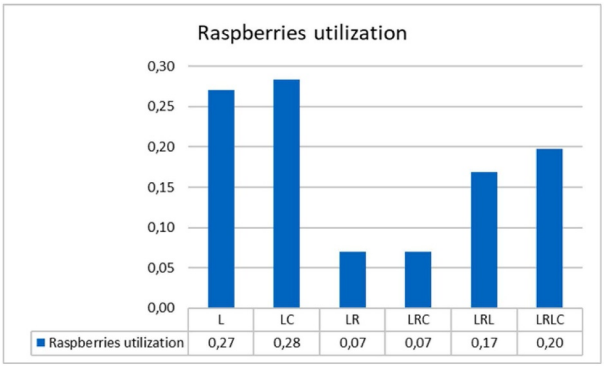
B



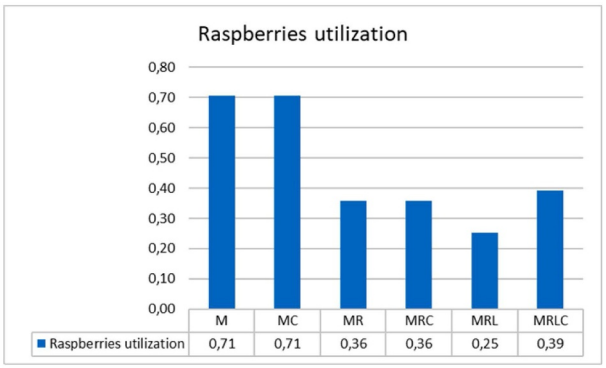
C

Fig. 5. (A) Cloud Utilization (Low Load). (B) Cloud Utilization (Medium Load). (C) Cloud Utilization (High Load).

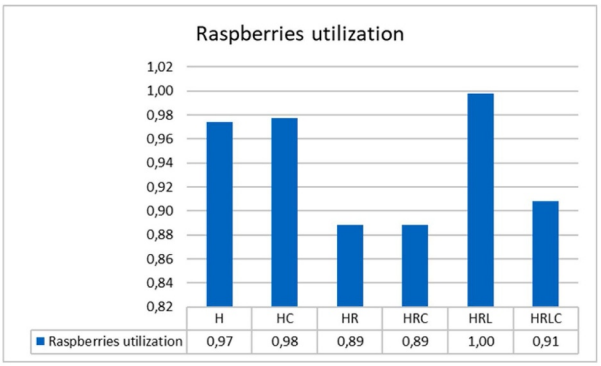
Fig. 5 shows the utilization of the cloud system in each case. It can be concluded that the cases do not differ significantly, although our algorithm is slightly better in Low and Medium Load cases. This is expected as the algorithm is programmed to choose the Raspberries over the VMs/Containers since they are not as expensive as the latters.



A



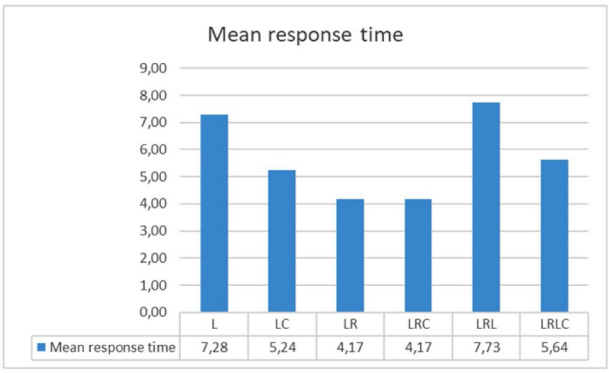
B



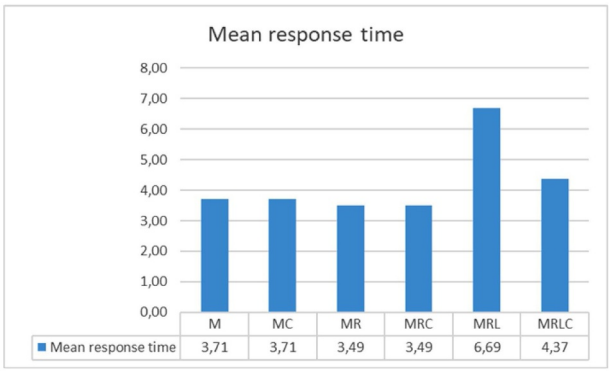
C

Fig. 6. (A) Raspberries utilization (Low Load). (B) Raspberries utilization (Medium Load). (C) Raspberries utilization (High Load).

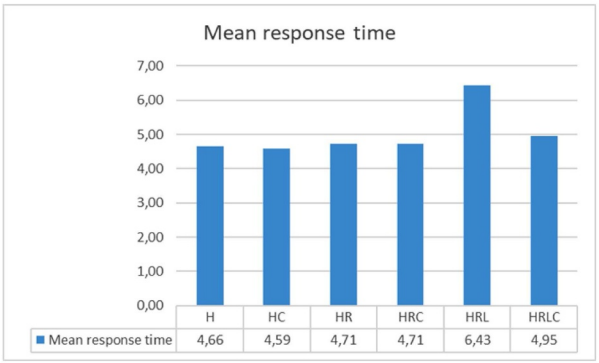
Fig. 6 shows the differences in each case between the utilization of raspberries. A slightly increased utilization can be observed with the use of the proposed algorithm, since it chooses Raspberries over Vms/Containers. The Weighted Round Robin algorithm, on the other hand, selects a system without considering either the costs nor the utilization of a system, resulting in same utilization for both systems (VMs - Raspberries).



A



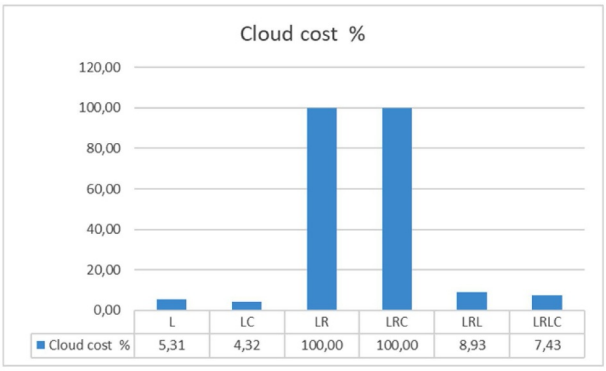
B



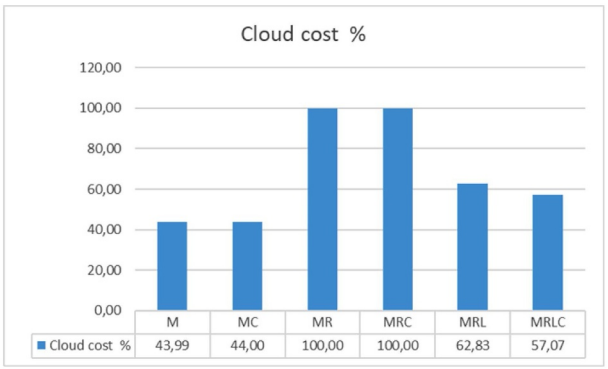
C

Fig. 7. (A) Mean response time (Low Load). (B) Mean response time (Medium Load). (C) Mean response time (High Load).

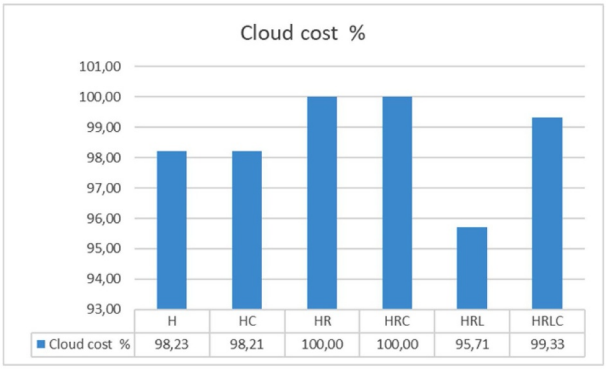
As shown in Fig. 7, when containers are used, the mean response time is significantly reduced, especially in the cases that the load balancer is used. Moreover, it can be concluded that the 'best' algorithm is the Weighted Round Robin, but it has to be mentioned that in this case all the VMs and Raspberries are continuously turned on which results in the increase of expenses. As a result, our algorithm may slightly increase the mean response time, but considering the reduction in expenses it is obvious that it is the preferred approach.



A

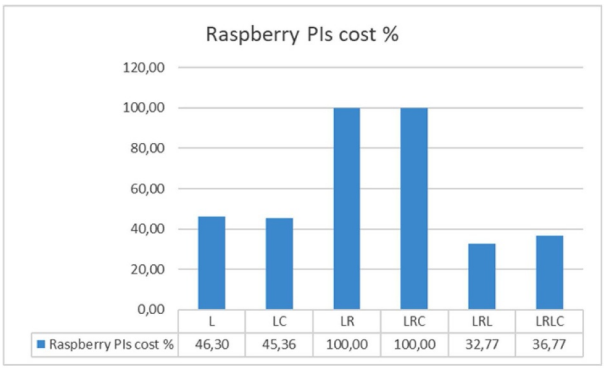


B

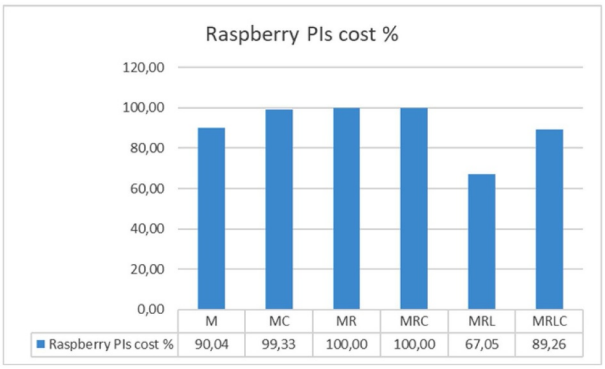


C

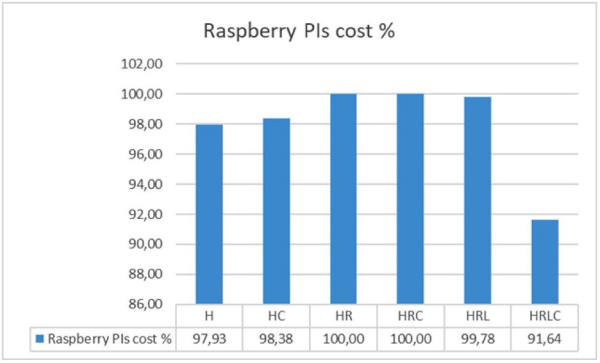
Fig. 8. (A) Cloud cost (energy + expenses for VMs, Low Load). (B) Cloud cost (energy + expenses for VMs, Medium Load). (C) Cloud cost (energy + expenses for VMs, High Load).



A



B



C

Fig. 9. (A) Raspberry PIs – energy consumption (Low Load). (B) Raspberry PIs – energy consumption (Medium Load). (C) Raspberry PIs – energy consumption (High Load).

The total cost of the cloud system and Raspberries cluster energy consumption are presented in Figs. 8 and 9 respectively. Either the monthly fees that someone pays for using it and/or the energy consumption needed for its use can be defined as "cost of the cloud system". We evaluate total cost as the percentage of the time when all the Raspberries and VMs were switched on to the time needed to complete the simulation. In any case, it can be safely concluded that the total cost of using the cloud system decreases with our algorithm. This is because only 1 VM/Container is powered on and more VMs/Containers are switched on when more computational power is needed. In the case of the low energy cluster, the same applies. In addition, when the proposed algorithm is used, particularly in the Low Load case, the difference is even greater since the algorithm prefers Raspberries over Vms/Containers to run the jobs. Figs. 8 and 9 show that when the proposed algorithm is used the total cost of the cloud system and the power consumption of the low-energy cluster, in the Low Load case scenario, can be reduced to ~7% and ~15%, respectively.

4.5. Summary of simulation

Summarizing our findings, we can conclude that the mean response time is slightly increased by using the proposed algorithm. This is more likely to happen specifically in the low-load scenario, as the algorithm chooses not to use the cloud or low-energy machines to save power and minimize costs. On the other hand, the total cost of either the low-energy cluster power consumption or the cost of VMs/Containers is reduced. This is because the proposed algorithm selects each time the solution that is less expensive. Therefore, a slight increase in the mean response time can lead to a significant cost reduction.

In addition, our algorithm has nearly the same results as the Weighted Round Robin Algorithm when there is high load but also minimizes costs.

Finally, in the Low Load case scenario, the proposed algorithm can reduce the cloud system's cost to **~7%** and the low-cost cluster's energy consumption to **~15%**. It should also be noted that the cost of using VMs/Containers is higher than the cost of energy consumption of the Raspberries. This is expected because the algorithm is structured so that it first selects the Desktop PCs, which are powered on at all times, and then the VMs/Containers or the Raspberries.

It can be concluded that if a user takes advantage of all available resources then he can reduce costs and increase the performance. The computational power of a Desktop PCs, Grids, VMs/Containers, Raspberries and Smartphones can be used simultaneously for this purpose.

5. Conclusion and future work

We argued in this paper that while the need for speed and scalability is increasing, the available resources were more diverse than those in a single cloud, cluster or grid. As a result, Fog Computing Systems were proposed in order to combine all the available computational power. It is a very difficult task to design a Fog computing system and it must meet a minimum set of requirements [15]. As a result, we proposed a Fog computing system algorithm that could reduce costs (e.g. monthly fees for VMs) by utilizing all available resources. In addition, we expanded our research by investigating the use of Containers rather than Virtual Machines to further reduce costs and decrease the execution time.

We simulated an algorithm that, by slightly increasing the mean response time, could minimize total expenses. Our system composed of 128 Virtual Machines/Containers, 32 Raspberries, 64 PCs and 64 Smartphones. We proposed a way to select the best resource available and minimize the overall cost. From the simulation results, we concluded that our algorithm could reduce the Cloud's cost to ~7% and reduce the Raspberries' energy consumption by ~85%. Mean response time was only slightly increased, and therefore load-balancing policies did not affect overall performance. The algorithm that we proposed also considered the costs of every available sub-system. We are going to examine the overall energy consumption of the system in future work and test our policies in many different ways than those used in this paper.

Furthermore, we believe it is necessary to use Fog Computing Systems to run applications on any available resources that somebody can access while minimizing the costs of either energy consumption or monthly fees. We demonstrated the viability of our approach and how someone could take advantage of the existing computational power and expand it with little effort by using other available resources, such as smartphones.

Finally, in a real case scenario, we will test the above algorithm using the JPPF, “Java Parallel Processing Framework”, a framework that can be deployed at any system with Java and also Android mobile phones.

References

[1] W. Zhu, C. Luo, J. Wang, S. Li, Multimedia cloud computing, IEEE Signal Process. Mag. 28 (3) (2011) 59–69.
[2] Y. Liu, W. Guo, W.S. Jiang, J.Y. Gong, Research of remote sensing service based on cloud computing mode, Appl. Res. Comput. 26 (9) (2009) 3428–3431.
[3] C.T. Yang, L.T. Chen, W.L. Chou, K.C. Wang, Implementation of a medical image file accessing system on cloud computing, Proceedings of the 13th International Conference on Computational Science and Engineering (CSE)), IEEE, 2010, pp. 321–326.
[4] P. Mika, G. Tummarello, Web semantics in the clouds, IEEE Intell. Syst. 23 (5) (2008) 82–87.
[5] D. Tychalas, H. Karatza, A cloud system for health care, Proceedings of the 19th Panhellenic Conference on Informatics, ACM, 2015, pp. 169–170.
[6] D. Kahanwal, D.T. Singh, The distributed computing paradigms: P2P, grid, cluster, cloud, and jungle, International Journal of Latest Research in Science and Technology 1 (2) (2013) 183–187 2012.
[7] D. Tychalas, H. Karatza, High performance system based on cloud and beyond: jungle computing, J. Comput. Sci. 22 (2017) 131–147.
[8] J. Maassen, N. Drost, H.E. Bal, F.J. Seinstra, Towards jungle computing with Ibis/Constellation, Proceedings of the Workshop on Dynamic Distributed Data-Intensive Applications, Programming Abstractions, and Systems, ACM, 2011, pp. 7–18.
[9] H.D. Karatza, Simulation study of multitasking in distributed server systems with variable workload, Simul. Model. Pract. Theory 12 (7-8) (2004) 591–608.
[10] K.H. Kim, R. Buyya, J. Kim, Power aware scheduling of Bag-of-Tasks applications with deadline constraints on DVS-enabled clusters, Proceedings of the CCGrid, 7 2007, pp. 541–548.
[11] I.A. Moschakis, H.D. Karatza, A meta-heuristic optimization approach to the scheduling of bag-of-tasks applications on heterogeneous clouds with multi-level arrivals and critical jobs, Simul. Model. Pract. Theory 57 (2015) 1–25.
[12] Z.C. Papazachos, H.D. Karatza, Scheduling bags of tasks and gangs in a distributed system, Proceedings of the International Conference on Computer, Information and Telecommunication Systems (CITS), IEEE, 2015, pp. 1–5.
[13] …, P.O. Östberg, J. Byrne, P. Casari, P. Eardley, A.F. Anta, J. Forsman, M.A.L. Pena, Reliable capacity provisioning for distributed cloud/edge/fog computing applications, Proceedings of the European Conference on Networks and Communications (EuCNC), IEEE, 2017, pp. 1–6.
[14] M. Firdhous, O. Ghazali, S. Hassan, Fog computing: Will it be the future of cloud computing? Proceedings of the European Third International Conference on Informatics & Applications (ICIA2014), 2014.
[15] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, ACM, 2012, pp. 13–16.
[16] A.V. Dastjerdi, H. Gupta, R.N. Calheiros, S.K. Ghosh, R. Buyya, Fog computing: Principles, architectures, and applications, Internet of Things, Morgan Kaufmann, 2016, pp. 61–75.
[17] A.M. Oprescu, T. Kielmann, Bag-of-tasks scheduling under budget constraints, Proceedings of the *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, 2010, pp. 351–359.
[18] C. Weng, X. Lu, Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid, Future Gener. Comput. Syst. 21 (2) (2005) 271–280.
[19] C. Anglano, M. Canonico, Scheduling algorithms for multiple bag-of-task applications on desktop grids: A knowledge-free approach, Proceedings of the *IEEE International Symposium on Parallel and Distributed Processing*, IEEE, 2008, pp. 1–8.
[20] M. Etemad, M. Aazam, M. St-Hilaire, Using DEVS for modeling and simulating a Fog Computing environment, Proceedings of the *International Conference on Computing, Networking and Communications (ICNC)*, IEEE, 2017, January, pp. 849–854.
[21] A.M. Law, W.D. Kelton, W.D. Kelton, Simulation Modeling and Analysis 2 McGraw-Hill, New York, 1991.
[22] I.A. Moschakis, H.D. Karatza, Multi-criteria scheduling of Bag-of-Tasks applications on heterogeneous interlinked clouds with simulated annealing, J. Syst. Softw. 101 (2015) 1–14.
[23] Richard M. Fujimoto, Parallel discrete event simulation, Commun. ACM 33.10 (1990) 30–53.
[24] I. Mavridis, H. Karatza, Performance and overhead study of containers running on top of virtual machines, Proceedings of the IEEE 19th Conference on Business Informatics (CBI), 2 IEEE, 2017, pp. 32–38.
[25] I. Mavridis, H. Karatza, Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing, Future Gener. Comput. Syst. 94 (2019) 674–696.
[26] D. Tychalas, H. Karatza, Simulation and performance evaluation of a fog system, Proceedings of the Third International Conference on Fog and Mobile Edge Computing (FMEC), IEEE, 2018, pp. 26–33.
[27] G.L. Stavrinides, H.D. Karatza, Task Group Scheduling in Distributed Systems, *Proceedings of the Third International Conference on* International Conference on Computer, Information and Telecommunication Systems (CITS), IEEE, 2018, pp. 1–5.
[28] B. Javadi, D. Kondo, J.M. Vincent, D.P. Anderson, Discovering statistical models of availability in large distributed systems: An empirical study of seti@home, IEEE Trans. Parallel Distrib. Syst. 22 (11) (2011) 1896–1903.
[29] Yinong Chen, Service-Oriented Computing and System Integration: Software, IoT, Big Data, and AI as Services, 6th ed., Kendall Hunt Publishing, 2018.
[30] Yinong Chen, Hualiang Hu, Internet of intelligent things and robot as a service, Simul. Model. Pract. Theory 34 (2013) 159–171.