IJCSI International Journal of Computer Science Issues, Volume 14, Issue 5, September 2017
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org https://doi.org/10.20943/01201705.6875

Energy-Efficient Virtual Machine Placement Algorithm

Bello Sururah Apinke, Gazali Abdulwakil Adekunle, and Aderounmu Ganiyu Adesola

**Computer Science and Engineering Department, Obafemi Awolowo University,
Ile-Ife. Osun State, Nigeria**

Abstract— **Cloud datacentres are large datacentres with thousands of servers that consume excessive energy and have significant carbon footprints. The increasing cloud users, on the other hand, are demanding more services with better response time. Hence, resources allocation, power management as well as better service delivery to users are challenging tasks for cloud providers. The problem has been formulated as a Bin Packing problem and many algorithms have been proposed with the aim of attaining maximum throughput and minimum computation time in order to achieve an energy efficient datacentre. In this study, a new algorithm called Neighbour-Fit was proposed to address the aforementioned problem. A model based on the algorithm, for Virtual Machine (VM) allocation, was designed. A web-based simulator was also developed using HTML, CSS and PHP to simulate the proposed model. The proposed algorithm was benchmarked with five existing allocation algorithms using throughput and computation time. The Neighbour-Fit algorithm is about 90 percent faster than the Almost Worst-Fit, Best-Fit, First-Fit and Worst-Fit algorithms. Although the Next-Fit algorithm is about 20 percent faster than the Neighbour-Fit algorithm, Neighbour-Fit algorithm utilizes 4 percent less number of servers than the Next-Fit algorithm. This performance infers that the Neighbour-Fit algorithm with a moderate computational time and a high throughput optimizes server utilization. This in turn reduces the power consumed by the servers in cloud datacentres.**

***Index Terms*—Cloud Computing, Datacentre, Energy Efficient, Neighbour-Fit, Placement Algorithm.**

I. Introduction

The growth of cloud computing has led to the setting up of massive datacentres with thousands of servers which in turn has increased the datacentres energy consumption. The resource and power management at this scale becomes an issue because cloud providers are interested in effective utilization of datacentre resources. Effective utilization of computing resources in a cloud datacentres entails reducing the number of running servers which, in effect, will reduce the energy consumption and consequently the operational expenses.

The economy of datacentre, according to [1], depends on three major building blocks. These building blocks are the electricity supply, networking infrastructure and cooling resources. In addition to the major building blocks are the cost of physical space (an estate) that is required to host the building and equipment, the operational expenditures incurred that are related to personnel, software licenses, and equipment depreciation. Thus, the economy of a datacenter can be summarized as follows:

$$Cost_{total} = Cost_{estate} + Cost_{power} + Cost_{networking} + Cost_{cooling} + Cost_{operation} \quad (1)$$

Reference [2] further decomposed the cost of power in (1) into the cost of power consumed by servers plus the cost of power consumed by switches plus the cost of power consumed by storage. This is summarized as follows:

$$Cost_{power\_hardware} = Cost_{power\_servers} + Cost_{power\_switches} + Cost_{power\_storage} \quad (2)$$

The cost of power consumed by servers is further decomposed into the addition of the cost of power consumed by the CPU, memory, disk, mainboard, and Network Interface Card (NIC). Equation 3 summarized the cost of power consumption by a server.

$$Cost_{power\_servers} = Cost_{power\_CPU} + Cost_{power\_memory} + Cost_{power\_disk} + Cost_{power\_mainboard} + Cost_{power\_NIC} \quad (3)$$

Reference [3] concluded that the CPU of a server consumes the most important amount of power and the relationship between power and CPU utilization is linear. The mathematical model is as shown in (4).

$$P = \left[\frac{c}{C}(1-\propto)+\propto\right] P_p \quad (4)$$

P is total power consumption of a Physical Machine (PM) at time t, P_p is peak power consumption, c is the total number of cores required by the resident VMs, C is the total number of

cores of a PM and $\propto$ is the percentage of idle power versus the peak power (this is usually 50 percent in a typical PM).
Hence, the economy of a datacentre depends largely on CPU utilization. Therefore, effective CPU utilization will cut down the running cost of a datacenter.

The Server and Energy Efficiency report states that more than 15% of servers in cloud datacentres are running without being used actively on a daily basis [4]. The United State (U.S.) Environmental Protection Agency (EPA) reported that the energy consumed by datacentres has doubled in the period of 2000 and 2006 and estimated another two fold increase over the next few years if the servers are not used in an improved operational scenario [5]. According to the Greenpeace International, some datacentres (including Akamai, Amazon, Apple, Facebook, Google, HP, IBM, Microsoft, Twitter, and Yahoo) use as much electricity as 250,000 European homes. Figure 1 shows the electricity consumption of various countries and the datacentres.

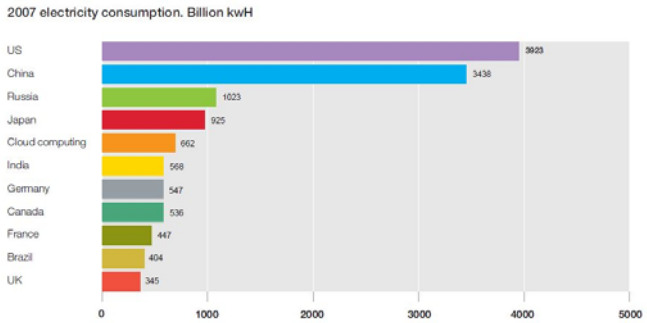

Fig. 1. Electricity consumption statistics of various countries in the year 2007. Source: [6].

It was found that if the cloud datacentres were a country its electricity demand would be more than the total electricity consumed by a big country like India, ranked 5th in the world, and is expected to triple by 2020 [6].

Cloud users, on the other hand, are interested in minimizing service response time and optimizing overall application throughput [7]. It becomes a very challenging task in cloud environment to allocate the resources with minimum operating time along with effective utilization of available resources.

Scheduling is responsible for allocating servers in cloud datacentre to users' resource requests and it is at the heart of resource management. The basic unit of scheduling in cloud datacentres is Virtual Machine (VM), a software-defined computer. Users' resource requests are submitted in the form of VMs and the resources in cloud datacentre are allocated using a number of VM scheduling techniques. VM scheduling techniques is an on-going research. Its aim is to minimize running servers as well as time taken in carrying out their operations. However, existing models take long time in scheduling servers for VM requests in a cloud datacentre. Hence, there is the need to allocate VMs in the cloud environment within a minimum time and using a minimum number of servers.

II. REVIEW OF LITERATURE

Reference [8] explained that more than half of the electrical power in a datacenter is consumed by the IT loads (see Fig. 2). Servers consume 80% of the total IT load and 40% of total data center power consumption. The rest of power is consumed by other devices like transformers, distribution wiring, air conditioners, pumps, and lighting. Therefore the easiest and most obvious way to save energy is to run fewer PMs [9].

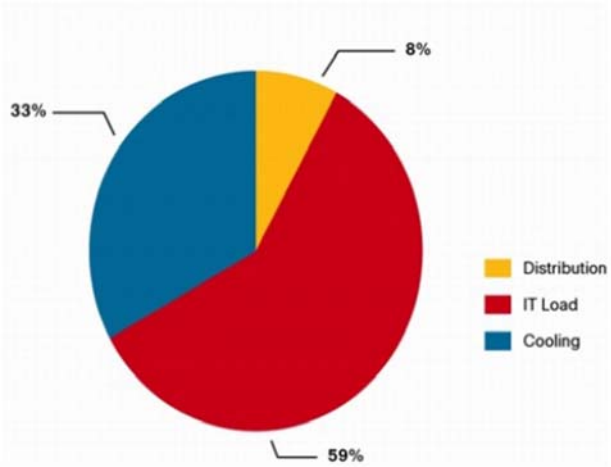

Fig. 2. Energy Consumption in a Datacentre.
Source:[8].

Cloud schedulers are implemented by placement algorithms. In recent years, several techniques like Constraint Programming, Integer Linear Programming, Genetics, Fuzzy, and Bin Packing techniques have been proposed for VMs placement in cloud datacentres [10]. Bin Packing technique is studied further in this work because of its usefulness in dynamic VM placement, especially where the demand is highly sporadic. Bin Packing is a heuristic based technique. It always generates a good solution in considerable amount of time. A Bin Packing technique is really useful when all PMs have the same amount of memory and processing capabilities.

Placement of VMs in a datacentre can be viewed as a Bin Packing problem. The PMs can be considered as bins having *n*-dimensions (CPU, RAM, and Disk). Similarly, the VMs can be considered as objects (having *n*- dimensions) to be packed into these bins. These dimensions correspond to resource requirements and capacities of the VMs and PMs respectively. In Bin Packing problem, there is always the need to find a mapping between these objects and bins such that the total number of bins required is minimized. By applying this technique in datacentre, it is possible to minimize the cost of running datacentre by packing the VMs required to be running at a time onto the least number of PMs. Some attempts of heuristic Bin Packing algorithms proposed by several authors are explained below:

Next-Fit algorithm, according to [11], places VMs in the order in which they arrive. It places the next VM in the request queue into the current PM if it fits. If it does not, leaves that PM, starts a new PM and then places the VM in it. For instance, assuming *S = {4, 8, 5, 1, 7, 6, 1, 4, 2, 2}* is a given set of VMs' CPU requests and CPU of datacentre's PMs are of size 10, placing the VMs into the PMs using Next Fit algorithm is shown in Fig. 3.

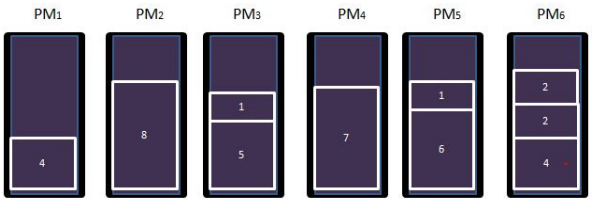

Fig. 3. VMs Placement under Next-Fit Algorithm.

The first request, four (*4*), was placed on PM_1 leaving six as the remaining capacity of the PM. Eight (*8*) was checked against the remaining capacity of PM_1 but the placement failed. Therefore, PM_2 was started and became the current PM. The request, eight (*8*), was placed in this PM leaving two (*2*) as the remaining capacity. The new request, five (*5*), was placed in PM_3 because PM_2 can not accommodate it. The next request, one (*1*), was placed in PM_3 because PM_3 which was the current PM can accommodate it. Seven (*7*) is the next request. It was placed on PM_4 because the remaining capacity on PM_3 was four and is less than the request (*7*). PM_5 becomes the current PM after six (*6*), the new request, has been placed on it. One (*1*) is placed on PM_5 and PM_5 remains the current PM. The next request, four (*4*), was placed on PM_6 because the remaining capacity on PM_5 was three (*3*). The next request two (*2*) was placed on PM_6 and the final request, two (*2*), in the queue was placed on PM_6. At this point, PM_6 remains the current PM because it still has two (*2*) unused capacities.
A total of six PMs are required to pack the VMs under Next Fit. This algorithm wastes PMs because some PMs may not be fully utilized (such as the first PM in Fig. 3). However, computation time for the Next-Fit algorithm is less because it does not perform any search during VM placement.

Reference [12] proposed an efficient VM assignment algorithm, the Resource-based First Fit Algorithm (RFFA), to assign VMs to PMs. Also dynamic placement of VMs to minimize SLA violations is studied in [13]. The authors modeled the problems as a Bin Packing problem and use the well-known First Fit algorithm to place the VMs to datacentre's PMs periodically.

First-Fit algorithm potentially assigns a VM to one of PMs with smaller identifiers. Therefore, the PMs with larger identifiers could be shut down and then the number of running PMs can be minimized. In another words, the algorithm places VMs in the order in which they arrive. It places the next item into the lowest numbered PM in which it fits. If it does not fit into any opened PM, it then starts a new PM. Figure 4 shows how First-Fit algorithm allocates VMs to PMs using the same set of VM requests and the same capacity of PMs as above.

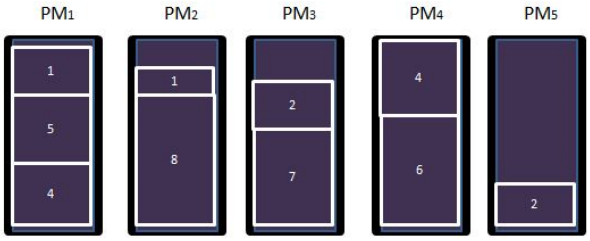

Fig. 4. VMs Placement under First-Fit Algorithm.

The first request, four (*4*), was placed on PM_1 leaving six as the remaining capacity of the PM. Eight (*8*) was checked against the remaining capacity of PM_1 but the placement failed. Therefore, PM_2 was started and the request, eight (*8*), was placed in this PM leaving two (*2*) as the remaining capacity. The new request, five (*5*), was checked against the remaining capacity of PM_1. The request, five (*5*) was placed on PM_1 leaving one (*1*) as the remaining capacity. The next request, one (*1*), was checked against the remaining capacity of PM_1. This request (*1*) was placed on PM_1 leaving zero as the remaining capacity of PM_1. The next request, seven (*7*), was checked against the remaining capacity of PM_1 but the placement failed. Then, it was checked against the remaining capacity of PM_2 but the placement failed. PM_3 was started and the request was then checked against the remaining capacity. Seven (*7*) was then placed on PM_3 leaving three (*3*) as the remaining capacity of PM_3. The next request, six (*6*) was checked against the remaining capacities of PM_1, PM_2 and PM_3 consecutively. The placement failed but PM_4 was started and six (*6*) was placed on it. Four (*4*), the next request, was checked against the remaining capacities on PM_1, PM_2, PM_3 and PM_4 consecutively.
The placement of four failed on PM_1, PM_2 and PM_3 but was successful on PM_4 leaving no remaining capacity on PM_4. The next request, one (*1*) was checked against the remaining capacities of PM_1. The placement failed but when it was checked against the remaining capacity on PM_2, One (*1*) was placed on PM_2 leaving no remaining capacity on PM_2. The next request, four (*4*), was checked against the remaining capacities on PM_1, PM_2, PM_3 and PM_4 consecutively. The placement of the request, four (*4*) failed on PM_1, PM_2 and PM_3 but was successful on PM_4 leaving no remaining capacity on PM_4. Two (*2*), was then picked and checked against the remaining capacities on PM_1, PM_2 and PM_3 consecutively. The placement failed on PM_1 and PM_2 but was successful on PM_3 leaving one (*1*) as the remaining capacity on PM_3. The last request, two (*2*), was checked against the remaining capacities on PM_1, PM_2, PM_3 and PM_4 consecutively. The placement failed, then a new PM was started and the request, two (*2*), was placed on it (PM_5). This algorithm uses less number of PMs compared to the Next Fit algorithm but its computation time is far more than the Next Fit algorithm.

Reference [14] proposed Modified Best Fit Decreasing (MBFD) algorithms for resource allocation. This algorithm sorts all VMs in decreasing order of utilization and allocates each VM to a PM that provides the least increase of utilization due to this allocation. If a VM does not fit in any of the running PMs, a new PM is started. Figure 5 shows how Best-Fit Decreasing algorithm allocates VMs to PMs using the same set of VM requests and the same capacity of PMs as above.

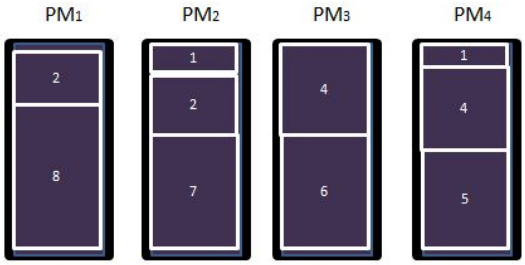

Fig. 5. Packing under Best-Fit Decreasing Algorithm.

The VM requests became $S = \{8, 7, 6, 5, 4, 4, 2, 2, 1, 1\}$ after sorting. The first request, eight (*8*) was placed in PM_1 leaving two (*2*) as the remaining capacity of the PM. The next request, seven (*7*) was checked against the remaining capacity on PM_1 but the placement failed. PM_2 was started the request was placed on it. The next request, six (*6*), was checked against the remaining capacities on PM_1 and PM_2. None of the PMs could accommodate the request. Therefore, PM_3 was started and the request, six (*6*) was placed on it. The next request, five (*5*), was checked against the remaining capacities on PM_1, PM_2 and PM_3. None of the PMs could accommodate the request. Therefore, PM_4 was started and the request, five (*5*), was placed on it. The next request, four (*4*), was checked against the remaining capacities on PM_1, PM_2, PM_3 and PM_4. Only PM_3 and PM_4 could accommodate the request but PM_3 would have the least remaining capacity after placement of the request. Therefore, the request, four (*4*), was placed on PM_3 leaving no remaining capacity on it. The next request, four (*4*), was checked against the remaining capacities on PM_1, PM_2, PM_3 and PM_4 and only PM_4 could accommodate the placement. Therefore, the request, four (*4*), was placed on PM_4 leaving one (*1*) as the remaining capacity on it. The next request, two (*2*), was checked against the remaining capacities on PM_1, PM_2, PM_3 and PM_4. Only PM_1 and PM_2 could accommodate the request but PM_1 would have the least remaining capacity after placement of the request. Therefore, the request, two (*2*), was placed on PM_1 leaving no remaining capacity on it. The next request, two (*2*), was checked against the remaining capacities on PM_1, PM_2, PM_3 and PM_4 and only PM_2 could accommodate the placement. Therefore, the request, two (*2*), was placed on PM_2 leaving one (*1*) as the remaining capacity on it. The next request, one (*1*), was checked against the remaining capacities on PM_1, PM_2, PM_3 and PM_4. Only PM_2 and PM_4 could accommodate the request and would have the least remaining capacities after placement of the request. Therefore, the placement was done on PM_2, being the PM with smaller identifier. The last request, one (*1*), was checked against the remaining capacities on PM_1, PM_2, PM_3 and PM_4 and only PM_4 could accommodate the placement. Therefore, the request, one (*1*), was placed on PM_4 leaving no remaining capacity on it.

Reference [15] has studied round robin algorithm to schedule and consolidate VMs. They have proposed a new strategy for VMs placement and migration that is called Dynamic Round-Robin (DRR). DDR as the extension of the Round-Robin method tries to reduce the number of active physical machines using two rules. In the first rule, if the running of a VM on a server has finished and there are still other VMs on the same physical machine, this physical machine will not accept new VMs.
In the second rule, if a physical machine remains in the first rule for a sufficiently long period of time, instead of waiting for the VMs to finish, the physical machine will be forced to migrate the rest of its VMs to other physical machines which in turn leads to shut down of physical machine after the migration completion.

Reference [16] proposed Most Full First algorithm. With this algorithm, PMs are sorted from most full to least full. Once sorting has been done, the VMs are allocated using First Fit.

Many of these placement/scheduling algorithms, apart from the Next-Fit and Round Robin algorithms, spend a considerable amount of time in sorting and searching before allocation of PMs to VMs is done. Also, the exempted algorithms allocate more PMs to VMs than other studied algorithms. Hence, this work intends producing a new heuristic (Neighbour-Fit) algorithm to strike the balance by allocating minimum number of PMs to VM requests and using minimum time.

III. Methodology

The activity diagram of the Virtual Machine allocation model as shown in Fig. 6 describes the movement of users' VM requests from the control node of the Cloud Providers to the pool of their servers (PMs).

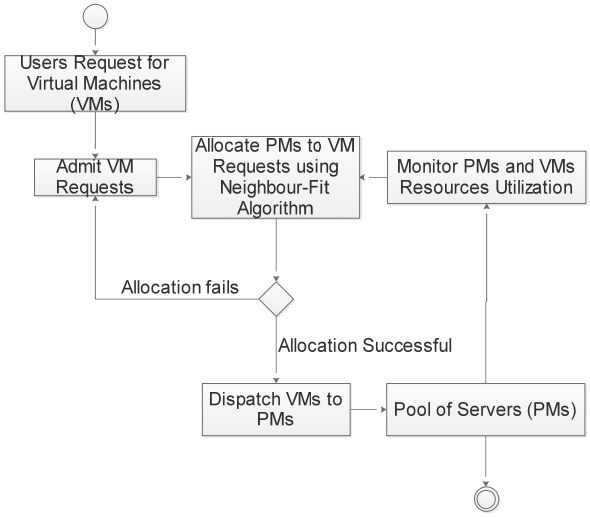

Fig. 6. On-demand VM Allocation Model.

Generally, VM requests arrive at the cloud scheduler and admitted into a queue of requests. The requests are then checked for placement on the pool of PMs in the datacentre by the placement algorithm (in this study, the proposed Neighbour-Fit algorithm) using the information gathered at the monitoring module. If allocation is successful, the VM dispatcher creates the VMs on the PMs. If otherwise, the VM request is rejected and the algorithm proceeds to the next request in the queue.

Neighbour-Fit algorithm is a heuristic algorithm that can be used to address Bin Packing problems. It was inspired through the behaviour of the well known Next-Fit algorithms. Unlike the Next-Fit algorithm which places objects in the current bin if it fits or open a new (next) bin if otherwise, Neighbour-Fit algorithm places objects in the previous bin if it fit, and behaves like Next-Fit algorithm if otherwise. This behaviour is shown in Fig. 7.

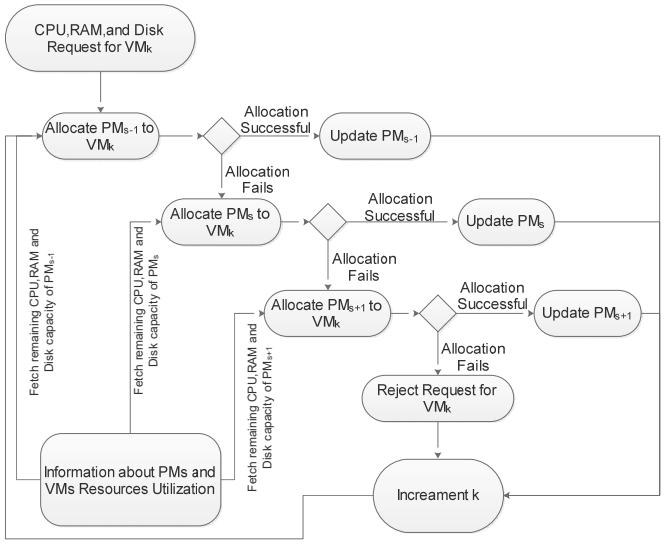

Fig. 7. Behaviour of Neighbour-Fit Algorithm in a Single Iteration.

Subscript *k* and *s* represent the current VM's request and current server (PM) respectively. The algorithm tries to allocate PM_{s-1} to VM_k. If the allocation fails, it tries to allocate PM_s to VM_k. If the allocation fails, it tries to allocate PM_{s+1} to VM_k. If the allocation fails, it rejects the request (VM_k) and picks another request by incrementing *k* by one. Wherever the allocation is successful, the PM (PM_{s-1}, PM_s, or PM_{s+1}) is updated, made the current server and *k* is incremented by one. The Neighbour-Fit algorithm and its sub function (bin filling function) is provided in Fig. 8 and Fig. 9 respectively.

Assuming $S = \{4, 8, 5, 1, 7, 6, 1, 4, 2, 2\}$ is a given set of VMs' CPU requests and CPU of datacentre's PMs are of size 10, placing the VMs into the PMs using Neighbour-Fit algorithm is as shown in Fig. 10.

```
Neighbour-Fit Algorithm (Main)
Set request index = 0
Set server index = 0
Loop while request index < Total Number of Requests
        if server index = Total Number of Servers – 1 then
                exit loop
        end if
        if server index < Total Number of Servers – 1 then
//Previous PM Checking Code
                if server index >= 1 then
                        if previous server capacity >= request requirement
                                call fill_bin(request requirement)
                                server index = server index - 1
                                request index = request index + 1
                                if request index >= Total Number of Requests
                                        exit loop
                                end if
                        end if

                end if
//Current PM Checking Code

                        if current server capacity >= request requirement
                                call fill_bin(request requirement)
                                request index = request index + 1
                                if request index >= Total Number of Requests
                                        exit loop
                                end if
                        end if

//Next PM Checking Code

                        if next server capacity >= request requirement
                                call fill_bin(request requirement)
                                request index = request index + 1
                                if request index >= Total Number of Requests
                                        exit loop
                                end if
                        end if
                        server index = server index + 1
        end if
end while loop
```

Fig. 8. Neighbour-Fit Algorithm.

```
Bin Filling Function
Function fill_bin(N)

    Set result = empty
    Set counter = 0
    Loop while counter < N
            result = result . "#";
            compute counter = counter + 1
    end while loop
    return result
```

Fig. 9. Sub Function of Neighbour-Fit Algorithm.

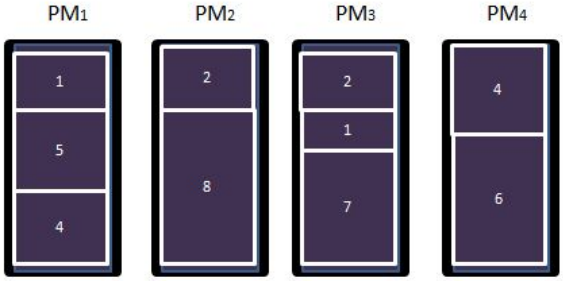

Fig. 10. Allocation Scenario According to Neighbour-Fit Algorithm.

The first item in the request queue (*4*) is placed in PM_1 leaving six (*6*) as the remaining capacity of PM_1. The second item in the request queue (*8*) is higher than the remaining capacity on PM_1 therefore, it is placed in PM_2. PM_2 became the current server. When the third item (*5*) is picked by the algorithm, it checked PM_1 if it can accommodate the request. This item (*5*) is placed in PM_1 leaving one (*1*) as the remaining capacity of PM_1. The fourth item (*1*) is placed on PM_1. The fifth item (*7*) is placed on PM_3 because the remaining capacity on PM_1 and PM_2 are less than the current request. The sixth item (*6*) is placed on PM_4 because the remaining capacity on PM_3 is less than the current request. The seventh item (*1*) is placed on PM_3 because the remaining capacity on PM_3 was three (*3*) leaving two as the remaining capacity of PM_3. PM_3 becomes the current server. When the eight item (*4*) is picked by the algorithm, it checked PM_2 if it can accommodate the request. The placement failed and the algorithm checked PM_3. The placement also failed but successful when PM_4 was checked. PM4 became the current server. When the ninth item (*2*) is picked by the algorithm, it checked PM_3 if it can accommodate the request. The placement is successful and PM_3 became the current server. When the tenth item (*2*) is picked by the algorithm, it checked PM_2 if it can accommodate the request. The placement is successful and PM_2 became the current server.

A. Asymptotic Analysis of the Neighbour-Fit Algorithm

Allocation is the mapping of PMs to VMs. It can be represented as shown in (5).

$$\text{Allocation} = Map(\text{PMs},\text{VMs}) \tag{5}$$

Each PM has three parameters (*C*, *R* and *D*). The *C*, *R*, and *D* are the capacities (in percentage) of CPU, RAM, and Disk for each PM. Also, each VM has three parameters (*c*, *r*, and *d*). The *c*, *r*, *d* are the requirements (in percentage) of CPU, RAM, and Disk for each VM request. The summation of *c* must not equal or greater than the available *C*. The summation of *r* must not equal or greater than the available *R*. Likewise, the summation of *d* must not equal or greater than the available *D*. In addition to the previous conditions, the remaining capacities ($C_s^{remaining}$, $R_s^{remaining}$, and $D_s^{remaining}$) of each PM can not be negative (that is, less than zero). These conditions are stated in (6) and (7).

$$\left\{ \begin{array}{l} \sum_k^K c_k < \sum_s^S C_s \\ \sum_k^K r_k < \sum_s^S R_s \\ \sum_k^K d_k < \sum_s^S D_s \end{array} \right\} \tag{6}$$

$$\left\{ \begin{array}{l} C_s^{remaining} \geq 0 \\ R_s^{remaining} \geq 0 \\ D_s^{remaining} \geq 0 \end{array} \right\} \tag{7}$$

Assuming C, R, and D are initially 100 percent empty and are switched off except the first server (PM) in the array.

In the main Neighbour-Fit algorithm, c_k, r_k and d_k are compared with the $C_s^{remaining}$, $R_s^{remaining}$, and $D_s^{remaining}$ respectively. Whenever $C_s^{remaining}$, $R_s^{remaining}$, and $D_s^{remaining}$ are greater than the c_k, r_k and d_k ; c_k, r_k and d_k values are then passed as argument into the function fill_bin. Function fill_bin has a loop that can iterate for the value of the argument (any of c_k, r_k and d_k). Therefore, the maximum iteration of function fill_bin is the maximum of the VM requirements (c_k, r_k and d_k) and is presented in (8).

$$N = max\,(c, r, d) \tag{8}$$

The number of iterations of Neighbour-Fit algorithm depends on the number of VM requests (*K*) and the number of PMs (*S*) in the datacenter. A loop that can iterate from zero to K-1 was constructed. Within the loop, statements that check for the attainments of *S* were provided. If any of the statements returns *true*, the loop will terminate. Therefore, the number of iterations of Neighbour-Fit is the minimum of *K* and *S*. This is presented in (9).

$$M = min(K,S) \tag{9}$$

Combining the two time complexities together, the asymptotic time complexity of Neighbour-Fit algorithm is in the order of *N*M* or **O(*N*M*).**

IV. RESULT AND DISCUSSION

The allocation algorithms under study were simulated 10 times with 2500, 5000, and 10000 VM requests respectrively. In a single run of the simulation, for 2500 VM requests, Fig. 11 shows the number of allocated PMs and Fig. 12 shows the computational time of each algorithm.

On the average, for 2500 VM requests, the number of allocated PMs by each algorithm and the computational time of each algorithm were recorded as shown in Table 1 and Table 2 respectively. Neighbour-Fit, Almost Worst-Fit, Best-Fit and First-Fit algorithm allocated 111 PMs to the requests. Next-Fit algorithm allocated 115 PMs and Worst-Fit algorithm allocated 110 PMs to the requests. Neighbour-Fit spent 0.33 second to allocate PMs to the requests. Almost Worst-Fit and Worst-Fit spent 3.58 second. Best-Fit and First-Fit spent 3.51 second and 2.38 second respectively. The Next-Fit algorithm spent 0.2 second to allocate PMs to the requests. Apart from the Next-Fit algorithm which was a little bit faster than the proposed algorithm (Neighbour-Fit), Neighbour-Fit proved to be significantly faster than other allocation algorithms.

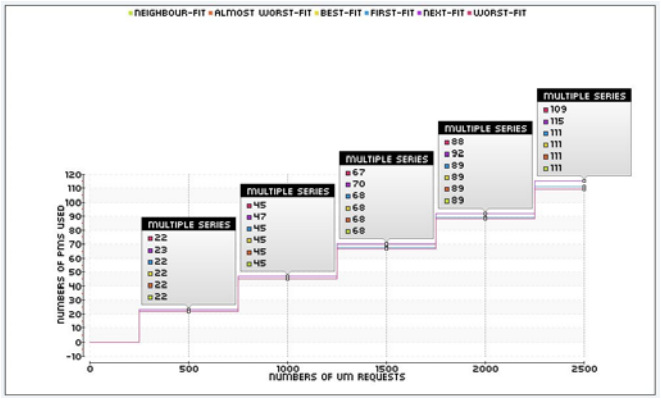

Fig. 11. Number of Allocated Servers for 2500 VM Requests at Intervals.

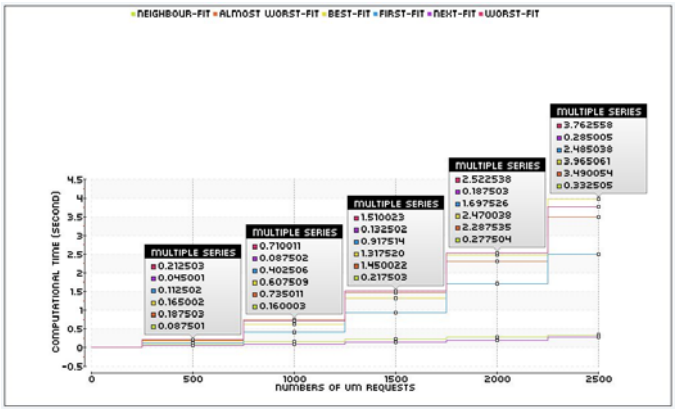

Fig. 12. Computation Time for 2500 VM Requests by Each Algorithm at Intervals.

Table 1. Number of Allocated PMs for 2500 VM Requests.

Algorithms	Neighbour-Fit	Almost Worst-Fit	Best-Fit	First-Fit	Next-Fit	Worst-Fit
Number of Servers (PMs) Used	110	111	111	111	115	108
	111	111	111	111	117	110
	111	111	111	111	116	110
	111	111	111	111	116	110
	111	111	111	111	116	110
	110	111	111	111	115	109
	111	111	111	111	115	110
	111	111	111	111	115	110
	110	110	110	110	115	110
	111	111	111	111	114	109
Mean	**110.7**	**110.9**	**110.9**	**110.9**	**115.4**	**109.6**
Standard Deviation	**0.483045892**	**0.316227766**	**0.31622777**	**0.31622777**	**0.843274043**	**0.699205899**

Table 2. Computational Time and Throughput of Each Algorithm for Placing 2500 VM Requests.

Algorithms	Neighbour-Fit	Almost Worst-Fit	Best-Fit	First-Fit	Next-Fit	Worst-Fit
Computational Time (seconds)	0.307018	3.749214	3.569204	2.357135	0.249014	3.548203
	0.328019	3.613207	3.753215	2.359135	0.278016	3.857221
	0.334019	3.999229	3.612206	2.392137	0.223012	3.472198
	0.320018	3.711212	3.398194	2.389136	0.225013	3.568204
	0.370021	3.402195	3.388194	2.482142	0.243014	3.4962
	0.294017	3.365192	3.403194	2.329134	0.36402	3.510201
	0.343019	3.368192	3.4992	2.43414	0.319019	3.948226
	0.34402	3.557204	3.4952	2.312132	0.224012	3.522201
	0.306017	3.553204	3.617206	2.303132	0.326018	3.438197
	0.34002	3.5082	3.367193	2.417139	0.219013	3.440197
Mean	**0.3286188**	**3.5827049**	**3.5103006**	**2.3775362**	**0.2670151**	**3.5801048**
Standard Deviation	**0.022526302**	**0.197665573**	**0.1265224**	**0.05673894**	**0.052143742**	**0.176415312**
Throughput (VM/s)	**7607.598835**	**697.7967959**	**712.18972**	**1051.5087**	**9362.766375**	**698.3035804**

Throughput according to [17] is the number of VMs that are successfully allocated per time unit. Therefore, for 2500 VM requests, the throughput of each allocation algorithm was calculated using (4) below and the results were also presented in Table 2.

$$Throughput = \frac{Number\ of\ allocated\ VMs}{Computational\ Time} \quad (4)$$

Neighbour-Fit had a throughput of 7607 VM/s which is significantly higher than other allocation algorithms under study except the Next-Fit. Although the Next-Fit algorithm is about 20 percent faster than the Neighbour-Fit algorithm, Neighbour-Fit algorithm utilizes 4 percent less number of servers than the Next-Fit algorithm. Figure 12 shows the throughput of each allocation algorithm.

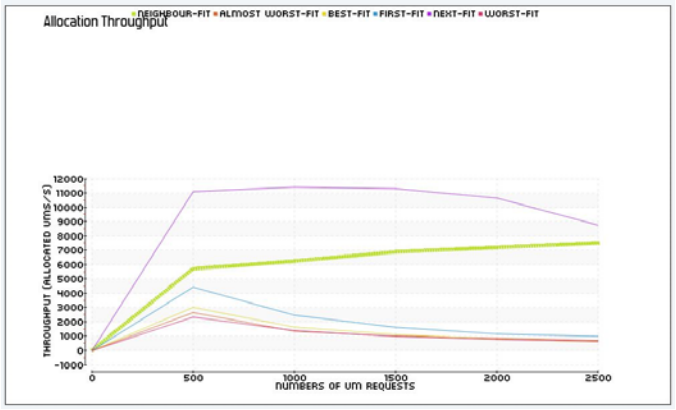

Fig. 13. Throughput of Each Allocation Algorithm.

V. CONCLUSION

In this study, a new VM placement algorithm (Neighbour-Fit) has been proposed. It is a step towards the implementation of a full-fledged cloud scheduler. It covers only the initial placement of VMs on a datacentre's PMs.

The results of the simulation have shown that the Neighbour-Fit algorithm is about 90 percent faster than the Almost Worst-Fit, Best-Fit, First-Fit and Worst-Fit algorithms. Although the Next-Fit algorithm is about 20 percent faster than the Neighbour-Fit algorithm, Neighbour-Fit algorithm utilizes 4 percent less number of servers than the Next-Fit algorithm.

Therefore, the proposed algorithm is better than the other under-studied algorithms in terms of computational time and server utilization which in turn reduces power consumption of cloud datacentres. In the future, the algorithm could be implemented in an open-source cloud manager like Opennebula. Consolidation of PMs in cloud datacentres using the proposed algorithm will also be carried out.

REFERENCES

[1] D. P. Chandrakant, and J. S. Amip. (2005). 'Cost model for planning, development and operation of a data center'. Hewlett-Packard (HP), HP Laboratories, Palo Alto, CA, USA, Tech. Rep.

[2] L. Luo, W. Wu, D. Di, F. Zhang, Y. Yan, and Y. Mao. (2012). 'A resource scheduling algorithm of cloud computing based on energy efficient optimization methods'. In Green Computing Conference (IGCC), 2012 International (pp. 1-6). IEEE.

[3] X. Fan, W. D. Weber, and L. A. Barroso. (2007). 'Power provisioning for a warehouse-sized computer'. In ACM SIGARCH Computer Architecture News (Vol. 35, No. 2, pp. 13-23). ACM.

[4] S. Karayi. (2009). 'Server Energy and Efficiency Report'. Technical report, 1E, 2009.

[5] R. Brown, E. Masanet, B. Nordman, B. Tschudi, A. Shehabi, J. Stanley, J. Koomey, D. Sartor, and P. Chan. (2008). 'Report to Congress on Server and Data Center Energy Efficiency: Public Law 109–431'. Lawrence Berkeley National Laboratory, Berkeley, CA, USA, 2008. Pp14-15.

[6] G. Cook, and J. V. Horn. (2012). 'How dirty is your data? A Look at the Energy Choices That Power Cloud Computing'. Technical report, Green Peace International, April 20, 2012.

[7] V. S. Rathor, R. K. Pateriya, and R. K. Gupta. (2014). 'Survey on Load Balancing through Virtual Machine Scheduling in cloud computing Environment'. International Journal of Cloud Computing and Services Science (IJ-CLOSER). 3, 37-43.

[8] C. Ghribi. (2014). 'Energy efficient resource allocation in cloud computing environments'. Networking and Internet Architecture [cs.NI]. Institut National des T´el´ecommunications, 2014.

[9] C. Aschberger, and F. Halbrainer. (2013). 'Energy Efficiency in Cloud Computing'.<http://www.uni-salzburg.at/fileadmin/multimedia/SRC/docs/teaching/SS13/SaI/Paper_Aschberger_Halbrainer.pdf> December 29, 2015.

[10] R. K. Gupta, and R. K. Pateriya. (2014). 'Survey on Virtual Machine Placement Techniques in Cloud Computing Environment'. International Journal on Cloud Computing: Services and Architecture (IJCCSA) , 4.

[11] K. Mills, J. Filliben, and C. Dabrowski. (2011). Comparing vm-placement algorithms for on-demand clouds. In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on (pp. 91-98). IEEE.

[12] C. F. Kuo, T. H. Yeh, Y. F. Lu, and B. R. Chang. (2015). 'Efficient Allocation Algorithm for Virtual Machines in Cloud Computing Systems'. In *Proceedings of the ASE BigData and SocialInformatics 2015* (p. 48). ACM.

[13] N. Bobroff, A. Kochut, and K. Beaty. (2007). 'Dynamic Placement of Virtual Machines for Managing SLA Violations,' in *Proc. of the IFIP/IEEE International Symposium on Integrated Network Management (IM'07)*, 2007.

[14] R. Buyya, A. Beloglazov, and J. Abawajy. (2010). 'Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges'. *arXiv preprint arXiv:1006.0308*.

[15] C. C. Lin, P. Liu, and J. J. Wu. (2011). 'Energy-efficient virtual machine provision algorithms for cloud systems'. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on* (pp. 81-88). IEEE.

[16] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder. (2011). '*Validating Heuristics for Virtual Machines Consolidation. Microsoft Research* MSR-TR-2011-9. pp. 1-14.

[17] S. Subramanian, N. G. Krishna, K. M. Kumar, P. Sreesh, and G. R. Karpagam. (2012). 'An adaptive algorithm for dynamic priority based virtual machine scheduling in cloud'. *International Journal of Computer Science Issues(IJCSI)*, *9*(6).

SURURAH Bello obtained B.Sc. Computer Engineering in 1995, M.Sc. Computer Science in 2002 and Ph.D Computer Science in 2011 from the Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria. She is currently a Senior Lecturer in the same department. Her research interests include Distributed Computing, Artificial Intelligence and Social Computing.

GAZALI Abdulwakil. Adekunle**.** was born in Lagos State, Nigeria, in 1981. He received the B.Sc. in Computer Engineering and M.Sc Computer Science from Obafemi Awolowo University, Ile-Ife, Osun State, Nigeria in 2012 and 2016 respectively.

Prof. ADEROUNMU Ganiyu Adesola is a Professor of Computer Science and Engineering in Obafemi Awolowo University, Ile-Ife. He is the current Director, Information Technology and Communications Unit, Obafemi Awolwo University, Ile-Ife, Nigeria. Professor Aderounmu received his B.Sc. Degree in Computer Engineering and M.Sc. Degree in Computer Science from Obafemi Awolowo University, Ile-Ife in 1991 and 1997 respectively. He is the current President, Nigeria Computer Society. Professor Aderounmu is a visiting research fellow, University of Zululand, Republic of South Africa.