IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol. 10, Issue.11, November 2022 E-ISSN: 2347-2693

Energy Efficient Offline Parallel Scheduling in Cloud Computing by Reducing Total Busy Time

Sebagenzi Jason

Department of Information Technology, AUCA University, Kigali 2461, Rwanda

Author's Mail Id: sebagenzij@gmail.com, Tel: +250782386040

DOI: https://doi.org/10.26438/ijcse/v10i11.815 | Available online at: www.ijcseonline.org

Received: 12/Oct/2022, Accepted: 31/Oct/2022, Published: 30/Nov/2022

Abstract - The basic scheduling issue is examined in this chapter. On *n* identical computers with bounded capacity, n deterministic jobs need to be scheduled offline. Each work has a start time, a finish time, a processing time, and a machine capacity requirement. The purpose is to schedule all of the jobs no proactively in their start-time–end-time windows, subject to machine capacity limits so that the overall busy time of the machines is minimized. Minimizing the overall busy time for the scheduling of several identical machines is the name we give to this issue (MinTBT). Power-aware scheduling for Cloud computing, optical network design, customer service systems, and other relevant fields can all benefit from solving this issue. In the particular case where all jobs have the same process time and can be scheduled in a set time interval, scheduling to reduce busy time is already NP-hard. The 5-approximation approach for exceptional situations utilizing the first-fit-decreasing (FFD) algorithm is one of the best-known solutions to this problem. In this chapter, we suggest and demonstrate a modified first-fit-decreasing-earliest 3-approximation technique for the general case and gain further results for particular situations. Then, we demonstrate how our findings might be used in cloud computing to increase energy efficiency.

Keywords - Energy-efficient scheduling; offline algorithm; online algorithm; MFFDE algorithm; BFF algorithm; GRID algorithm; approximation ratio; competitive ratio.

I. INTRODUCTION

In order to represent the task scheduling issue in machines, we use a three-field notation system. This format is suggested in Ref. [1] as, which, respectively, specifies the task characteristics, processor environment, and goal function. For instance, the multiprocessor problem of reducing the completion time (makespan) when each job has a set release date and deadline is referred to as $P|r_j,e_j|Cmax$. When each task has a release date and deadline defined and a m number of processors is supplied as part of the problem type, the multiprocessor problem of minimizing the total completion time is known as $Pm\ |r_j,e_j|\ C_j$.

The notation used in this chapter to represent several machines (each with capacity g) is Pg|rj,ej|ibji. The goal is to reduce the total busy time of all employed machines, and each work has a start time and end time that are set throughout the intervals in which it should be processed. The input is formally a collection of n jobs, $J=J_1, J_2... J_n$. Each job Jj has a processing interval [sj, ej] that it must be completed within; the job J_j process time is defined as $P_j = e_j-s_j+1$. The capacity parameter g1, which represents the maximum capacity offered by a single machine, is also provided. The working time interval length bi of a machine I represents its busy time. The objective is to assign tasks to machines in a way that minimizes the total busy time of all machines, as determined by B=ibi. It should be noted that the output of the method includes the number of machines (m>1) to be employed and accepts an integral value. As far as we are aware, Khandekar et al. [2] are among the first to address this issue, and Brucker [3] discusses the subject and relevant references therein. Unless otherwise stated, indices are written in lower case, whereas a list of jobs, time periods, and machines are written in upper case.

With cloud computing, software, computational, and storage network resources can be shared, allocated, and aggregated as needed. The concealment and abstraction of complexity, the efficient utilization of remote resources, and virtualized resources are a few of the major advantages of cloud computing. A big problem is increasing the energy efficiency of cloud data centers. While Jing et al. [5] undertake a state-of-the-art research study for green Cloud computing and identify three hot research areas, Beloglazov et al. [4] offer a taxonomy and assessment of energy-efficient data centers for Cloud computing.

Virtual machine (VM) resources are made available by Cloud Infrastructure as a Service provider with specific compute units, as Amazon EC2 [6]. A customer makes a time-limited request for certain computing units of resources and pays for them according to the total amount of time they have been used. The overall power-on (busy) time of all computing resources and the total energy cost of those resources for a provider are strongly correlated. Therefore, in order to reduce energy expenses, a provider

seeks to reduce the overall busy time. As a result, we propose and demonstrate a 3-approximation technique in this chapter called modified first-fit-decreasing-earliest (MFFDE), which may be used to schedule virtual machines in cloud data centers in a way that uses the least amount of energy possible.

II. RELATED WORK

On parallel machines, job scheduling has been thoroughly studied. Traditional interval scheduling [7-9] involves real-time interval job delivery, job processing on a single machine that can only handle one job at a time.
Numerous studies have been done on scheduling with fixed intervals, where each job must be processed on a machine during the time interval between its release time and due date, or each job must be processed during the fixed interval between its start time and end time, assuming a machine can process one job at a time. Studies of real-time scheduling with capacity demands, where each machine has a capacity, are also available; However, Khandekar et al. [2] are among the first to articulate the goal of minimizing the overall busy time, to the best of our knowledge. There has also been prior research on the issue of allocating tasks to a group of machines in order to reduce overall costs [10], but in these studies, the cost of allocating each task is fixed. In contrast, the cost of scheduling each work in our scenario depends on the other operations that are planned on the same machine in the corresponding time interval; as a result, it might vary over time and across various machines. Our scheduling issue differs from batch scheduling of incompatible jobs [3], as was mentioned in [2].

The scheduling problem is NP-hard in the broad sense [11]. When the jobs are intervals on the line, like in Chapter 6, it is demonstrated that the problem is NP-hard for g=2. The scheduling problem, when jobs are provided as intervals on the line with unit demand, is taken into account by Flammini et al. [12]. Flammini et al. provide a 4-approximation approach and improved bounds for various subclasses of inputs for this variant of the problem. When no interval is adequately contained in another interval (i.e., the input forms a suitable interval graph) or when any two intervals intersect (i.e., the input forms a clique; see also Reference [2]), Flammini et al. specifically provide a 2-approximation technique. Additionally, Flammini et al. offer a 2-approximation for bounded durations of time, meaning that any job's length (or process time) is constrained by a fixed integer d.

By dividing all jobs into wide and narrow jobs based on their needs when =2, which is a demand parameter of narrow jobs relative to the entire capacity of a machine, Khandekar et al. [2] offer a 5-approximation approach for the scheduling problem. Only in this unique circumstance are the results based on =2 valid. In this chapter, we propose a 3-approximation approach to solve our scheduling problem, expanding and improving the findings of Ref. [2].

In terms of energy efficiency, one of the difficult scheduling issues in cloud data centers is to take into account the allocation and migration of virtual machines with full life cycle limitations, which is sometimes overlooked [13]. The interrelationships between power consumption, resource usage, and performance of aggregated workloads are examined by Srikantaiah et al. [14]. By condensing active tasks, Lee and Zomaya [15] offer two online heuristic methods for resource-efficient utilization of Cloud computing systems. In a Xen virtualized system, Liu et alinvestigation.'s of performance and energy modeling for live VM migration and evaluation of the models using five sample workloads. By reducing the total number of machines employed and the total number of migrations using updated best-fit bin packing heuristics, Beloglazov et al. [10] investigate the offline allocation of VMs. Real-time services are modeled by Kim et al. [17] as real-time VM requests, and they employ dynamic voltage frequency scaling techniques. Mathew et al[18] .'s proposal for an ideal offline algorithm and an online algorithm for content delivery networks combines load balancing with energy economy. Constrained mixed-integer programming is used by Rao et al. [19] to model the issue and suggest a rough resolution. In order to reduce overall costs, Lin et al. [20] suggest online and offline algorithms for data centers that turn off unused servers. However, research on VM scheduling that takes into account set processing intervals is still lacking. As a result, we show how our suggested 3-approximation approach can be used for VM scheduling in cloud computing in this chapter. Similar problem models are taken into consideration by Mertzios et al. [21], but only in relation to specific exceptional circumstances. While we concentrate on energy economy in Cloud data centers, they primarily offer constant factor approximation algorithms for both total busy time minimization and throughput maximization concerns.

III. METHODOLOGY

Based on the following assumptions, the objective of energy-efficient scheduling is to satisfy all criteria with the least number of machines and their combined busy times:

- Unless otherwise stated, the time is formatted in slotted windows, and all data are deterministic. The total number of slots is k=T/I0 since we discretely divide the complete time period [0, T] into slots of equal length (always making it a positive integer). The system's start-time is set to s0=0. The interval of a request, j, can then be expressed in slot format as [StartTime, EndTime, RequestedCapacity] = [s_i, e_i, di], where s_i and e_i are nonnegative integers for both the start-time and end-time.
- Each task in a job is distinct. Except for the ones that the start-time and end-time imply, there are no further precedence restrictions. This chapter does not take preemption into account either.
- Each request's necessary capacity is a positive integer in the range [l, g].

- Interrupting a request and continuing it on another machine is not permitted, unless otherwise specifically indicated, presuming that each request is assigned to a single machine when processed.

We can derive the following significant definitions and observations from the aforementioned presumptions:

Definition 1: The length of Ii is |Ii|=t-s+1 for a time interval Ii = [s, t] where s and t are the start-time and end-time, respectively. Len(I)=|I|= (I =1) k |Ii|, or the length of a set of pairwise intervals, is defined as the sum of the lengths of each interval in the set, where I= (I = 1) k |Ii|.

Definition 2: The length of the union of all intervals taken into account is known as span(I), and it is defined as span(I) = |I |.

Example 1:

I = [1.4], [2, 4], [5, 6], in which case span(I) = [1, 4] + [4, 6] = (4 - 1) + 1 + (6 - 5) + 1 = 6, and Len(I) = [1, 4] + [2, 4] + [5, 6] = 9. Keep in mind that equality and span(I) Len(I) only apply if and only if I is a collection of pairwise no overlapping intervals.

Definition 3: Let OPT(I) stand for the minimized overall busy time across all machines for any instance I and capacity parameter g1. In this context, busy time refers only to when all machines are turned on. To reduce the overall busy time is to minimize the sum of makespan on all machines, according to Definition 2 of span(I). It should be noted that a machine's total power-on time is the total of all of its power-on times. Similar to Example 1, a machine is active (powered on) during the times [1, 5] and [5, 6]. The total busy time of this machine, based on Definition 1 of the interval for each job, is (51) +(65) =5 time units (or slots). The machine's overall busy time does not encompass the range [0, 1].

Definition 4: If the total busy time is at most C times that of an optimal solution, an offline deterministic method is said to be a C-approximation for the goal of minimizing the total busy time.

Definition 5: The needed capacity of each job di is a natural number between 1 and g, i.e., 1 di g, assuming that the start-time and end-time of all jobs are nonnegative integers.

Definition 6: The necessary workload for any job j is w(j), which is equal to the capacity demand times the process time, or w(j)=djpj. Then, W(J)= jnw(j) represents the total workload of all tasks J.

In Ref. [2], the following observations are listed.

The following bounds hold for any instance J and capacity parameter g1:

i. Capacity bound: OPT(J)≥W(J)/g;

ii. Span bound: OPT(J)≥span(J).

Because g is the most capacity that any solution can attain, the capacity bound is valid. The span bound is valid since, for g=1, only one machine is required.

Observation 2: OPT(J)len is the upper bound for the ideal total busy time (J). When g=1 or when g is greater than 1, the equality is maintained and no periods overlap.

IV. RESULTS AND DISCUSSION

The machines are designated as M_1, M_2, and are used to analyze any scheduler S. where J_i is the group of tasks that the scheduler S has given to the machine Mi. Bi=span(Ji) for all I 1, where span(Ji) is the span of the set of job intervals scheduled on M_i, is the length of a machine's overall busy period.

We get the following outcomes for the goal of minimizing the total busy time of several similar machines without preemption subject to fixed interval and capacity limitations (referred to as MinTBT):

- In the general situation, scheduling without preemption and with capacity constraints (MinTBT) is an NP-complete problem for minimizing the total busy time of numerous identical machines (Theorem 1).
- When the demand is one unit and the combined capacity of each machine is also one unit, there are algorithms that can discover the MinTBT problem's optimal solution in polynomial time, therefore in this situation, MFFDE(I)=OPT(I)=len(I) (Theorem 2). This demonstrates the outcome in a unique case that can be used with cloud data centers that use less energy.
- Our suggested MFFDE algorithm's approximation ratio for the MinTBT issue has an upper constraint of 3. (Theorem 3). One of our key findings that directs our approximation of the algorithm design is this.
- There is a specific case of 1 di g for the unit demand scenario, which has di=1, as illustrated in Ref. [12]. (let us call it a general demand case). The unit demand case provides the worst-case scenario for first-fit-decreasing (FFD) and MFFDE algorithms in terms of minimizing the overall busy time (Observation 3).
- There are techniques to identify the optimal minimum number of machines for the MinTBT problem in polynomial time for scenarios when the capabilities of all requests form a highly divisible sequence (Theorem 4). This makes it possible to create algorithms that are roughly and nearly optimal.
- There are algorithms to determine the MinTBT problem's ideal resolution in polynomial time for the cases where the capacity parameter g=. (Theorem 5).
- The overall busy time of all physical machines (PMs) dominates the total energy consumption of all PMs for a linear power model and a given set of VM requests in cloud computing, i.e., a longer total busy time of all PMs for a scheduler result in a higher total energy consumption (Theorem 6).
- This paper's remaining material is organized as follows: Our suggested approximation algorithm and its approximation bounds are presented in Section 7.2. Its applicability to VM scheduling in cloud computing

is covered in Section 7.3. The performance of MFFDE, FFD, and the theoretical best solution are compared in Section 7.4. The conclusion and future directions for this field of study are described in Section 7.5.

Approximation Algorithm and its approximation ratio bound.

The longest processing time (LPT) is one of the best approximation techniques for offline non-real-time scheduling. It is well known that LPT has the best upper bound for minimizing the maximum makespan in the scenario where a conventional multiprocessor system is used [4]. The general example is covered in this chapter, and the start and finish times of jobs are fixed. When allocating work, we must take into account the defined start and end times of each task as well as the machine capacity limitation. If two jobs have the same process time, our MFFDE algorithm, as displayed in Algorithm 1, schedules them in the reverse order of their process times and gives preference to the earlier start time. If two jobs have the exact same start time, end time, and process time, however, it arbitrarily breaks ties. Each task is assigned to the first available machine (so as to use as few machines as possible to minimize the total busy time).

Input: (J, g) where J is set of jobs and g is maximum capacity of a machine

Output: Scheduled jobs, total busy time of all machines, and total number of machines used

Sort all jobs in non-increasing order of their process times, such that $p_1 \geq p_2 \ldots \geq p_n$ (Considers earlier start-time first if two jobs have the same process time. Breaks ties arbitrarily when two jobs have exactly the same start-time,end-time, and process time)

for $j = 1$ to n do

Find first machine i with available capacity;

Allocate job j to machine i and update its load;

Compute workload and busy time of all machines;

ALGORITHM 1 MFFDE Algorithm.

To determine how challenging the overall issue is:

Theorem 1: In the general situation, the MinTBT problem—which seeks to minimize the total busy time of multiple identical machines in offline scheduling without preemption and with a capacity constraint—is NP-complete.

Proof: This is demonstrated via polynomial-time reduction of the well-known NP-complete set partitioning issue to the MinTBT problem as follows:

For a set S of positive numbers and an integer k, the K-Partition problem is NP-complete [22]; divide S into k ranges such that the sums of all the ranges are near to one another.

The MinTBT problem can be converted from the K-Partition problem as follows:

The allocation of K ranges of jobs with the capacity constraint g is equivalent to partitioning J by capacity for a collection of jobs J where each job has a capacity demand di (specified as a positive number) (i.e., the sum of each range is at most g). On the other hand, if K-Partition can be solved for a given collection of intervals, a schedule can be created for that set of intervals. Our issue is NP-hard because K-Partition is NP-hard in the strict sense. In this manner, we have established the NP-completeness of the MinTBT issue. In the particular scenario where all jobs have the same (unit) process time and can be scheduled in one fixed time period, Khandekar et al. [2] have demonstrated that it is already NP-hard to approximate our problem.

Bounds for Approximation ratio when g is one unit and d_i is one unit.

The typical interval scheduling problem with start-time and end-time constraints, where each job requires a one-unit capacity and the total capacity of a machine is one unit, is what our problem simplifies to when g is one unit and di is one unit.

Theorem 2: When the demand is one unit and the combined capacity of each machine is also one unit, there are algorithms to discover the MinTBT problem's optimal solution in polynomial time, notably in the situation of MFFDE(I) = OPT(I) = len (I).

Proof: Let's set the capacity parameter g to 1 since it is a unit-based parameter. Each machine can only handle one job at a time due to the capacity 1 requirement for each job. In this situation, regardless of whether there are jobs that overlap or not, OPT(I) = len(I) is true using Definition 1 of interval length and Definition 2 of span. MFFDE(I) is also the total of lengths of all intervals by assigning each interval to distinct machines for continuous working intervals.

Bounds for Approximation ratio in general case when g>1.

Observation 3: The unit demand case, or $d_i = 1$, as illustrated in Ref. [12], is a specific case of 1 di g. (let us call it a general demand case). The unit demand example depicts the worst-case scenario for FFD and MFFDE algorithms in terms of minimizing the overall busy time.

Proof: Think about the general demand scenario, in which 1 di g The enemy is created in the following way: All g groups of requests have the same start time at si=0, demand di (for 1 I h, _(i=1)hdi=g), and each has an end time at ei=T/kg-1, where T denotes the amount of time being taken into account, k denotes a natural number, and j denotes the modality of the group in question if I mod g 0, otherwise j de The best course of action in this situation is to allocate all of the longest requests to machine (m1) for a busy time of dgT, all of the second-longest requests to machine (m2) for a busy time of dg-1 T/k,..., and finally, all of the shortest requests to machine (mg) with a busy time of diT/kg-I. As a result, the total busy time of the best course of action is dgT.

$$\text{OPT(I)}=T\sum_{i=1}^{g}\frac{di}{gk^{g-i}} = T\sum_{i=1}^{g}\frac{di}{k^{g-i}} \qquad (7.1)$$

We think of the worst scenario (upper bound). The upper bound will make ALGx/OPT the largest while maintaining all other constraints for any offline algorithm, let's call it ALGx. If di has the smallest value, that is, di=1, then Eq. (7.1) will have the least value when k and T are supplied. In other words, the unit demand situation is the worst-case scenario.

Remark 1: For the worst-case FFD scenario depicted in Figure 7.1, we can quickly verify that Observation 3 is accurate. We only take into account the unit demand case for the upper bound because that is the MinTBT problem's worst-case scenario.

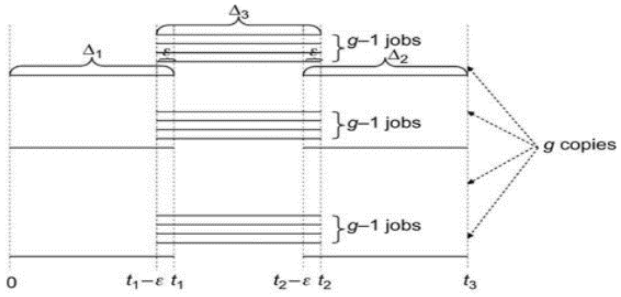

Figure 1 Generalized instance for the proof of the upper bound of FFD.

In References [2,12], the observation described below is made:

Observation 4: In the worst scenario for the FFD algorithm, where m is the total number of machines employed, we have span (Ii + 1) 3w(Ii)/g.

Remark 2 A span (Ii + 1) 3w(Ii)/g result is established and shown for the FFD method in Ref. [12]. The process time for a job on a machine M_i is called pi. Let iL or ig represent the job that finished on machine Mi+1 with the earliest or latest completion timings, respectively, in Ii + 1. We also have span (Ii + 1) 3w(Ii)/g because our suggested approach is also based on the FFD algorithm for process time and takes earlier start-times into account first when ties exist.

Theorem 3 Our suggested MFFDE algorithm for the MinTBT problem has an upper bound on the approximation ratio of 3.

Proof Let's stipulate that the machine $M_i + 1$ is given the assignment of all the jobs in $J_i + 1$. The assignment's total busy time for such a set is exactly its span.

$$\sum_{i+1}^{m} MFFDE(J_i) = MFFDE(J_i) + \sum_{i=2}^{m} MFFDE(J_i) \quad (7.2)$$

$$= MFFDE(J_i) + \sum_{i=1}^{m-1} MFFDE(J_{i+1}) \quad (7.3)$$

$$\leq MFFDE(J_i) + \frac{3}{g} \sum_{i=1}^{m-1} w(J_i) \quad (7.4)$$

$$= MFFDE(J_i) + \frac{3}{g} \sum_{i=1}^{m} w(J_i) - \frac{3}{g} w(J_m) \quad (7.5)$$

$$= MFFDE(J_i) + \frac{3}{g} W(J) - \frac{3}{g} w(J_m) \quad (7.6)$$

$$\leq 3\ \mathrm{OPT(J)} + MFFDE(J_i) - \frac{3}{g} w(J_m) \quad (7.7)$$

$$\leq \mathrm{OPT(J)} \quad (7.8)$$

Eq. (7.6) should theoretically have the upper bound when MFFDE (J i) has the biggest value and (3/g) w (J m) has the smallest value at the same time, however this is typically not the case. This is how the analysis is presented:

1. If all long jobs are allocated to machine M 1 and MFFDE (J 1) =span (J 1) has the upper bound OPT(J), then allocations on other machines have little impact on OPT(J), and 3/(g) w (J m) is very small (which can be ignored in comparison to span (J 1); otherwise, MFFDE (J 1) = span (J 1) cannot reach the upper bound OPT (J). In this instance, span (J 1), which is very close to or equal to OPT, dominates _(I=1) mMFFDE (J i) (J).
2. If MFFDE (J 1) =span (J 1) is less than OPT, then (J). We take the worst-case scenario into consideration since it is for the upper bound (i.e., OPT(J) is not dominated by MFFDE (J 1)). As shown in Figures 7.1 and 7.2, in the worst situation, span (I (i+1))3w (I i/g), making it simple to verify that MFFDE (J 1) (3/g) w (J m). Set Δ 0 = Δ (1 =) Δ (2 =) Δ 3, When jobs have the identical process timings, MFFDE actually takes into account the earlier start-time first, therefore MFFDE (J 1) =span (J 1) = _ (0)-2, (3/g) w (J m) = (3/g) w (J g) = (3/g) (g (0) + _0) = 3 (0) + (3 _ (0)/ g). OPT(J) in this instance equals g (0) + g 0. Therefore, when g is big, MFFDE (J 1) -(3/g) w (J m) = -2 0-(3 0/ g) is much smaller than OPT(J). We may deduce MFFDE (J 1)- (3/g) w (J m) + (3/g) w(J) from Eq. (7.7) as 3 OPT (J). Using Figure 7.2 as the worst scenario, a tight upper bound is demonstrated in this situation (which is shown in the next proof).
3. We may readily discover that MFFDE(J) is very near to or equal to OPT for certain circumstances, such as one-sided clique and clique cases [2,12]. (J).

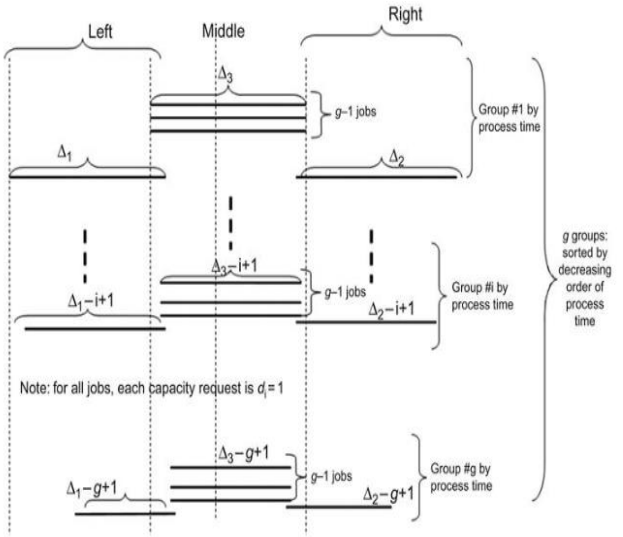

Figure 2 Generalized instance for the proof of the upper bound of MFFDE.

We have demonstrated Theorem 3 by integrating the aforementioned three analyses.

Because we are aiming for the upper bound, a different, simpler proof just takes the worst case into account. The

worst-case scenario for the FFD method is depicted in Figure 7.1, as was mentioned in References [2,12]. Since the MFFDE method takes into account the earliest start-time first (ESTF) when two requests have the same amount of processing time, we can simply verify that MFFDE(J)= OPT(J). We further develop the MFFDE algorithm's worst-case scenario and offer the following evidence.

Proof

Table 1 Eight VM types in Amazon EC2

VM Type	Compute units	Memory (GB)	Storage (GB)
1-1(1)	**1**	**1.875**	**211.25**
1-2(2)	**4**	**7.5**	**845**
1-3(3)	**8**	**15**	**1690**
2-1(4)	**6.5**	**17.1**	**422.5**
2-2(5)	**13**	**34.2**	**845**
2-3(6)	**26**	**68.4**	**1690**
3-1(7)	**5**	**1.7**	**422.5**
3-2(8)	**20**	**6.8**	**1690**

Table 2 Three PM types for divisible configuration

PM Type	CPU	Memory (GB)	Storage (GB)	Pmin (W)	Pmax (W)
1	**16**	**30**	**3380**	**210**	**300**
2	**52**	**136.8**	**3380**	**420**	**600**
3	**40**	**14**	**3380**	**350**	**500**

Algorithm

We contrast the following three algorithms:

1. (FFD [2,12] assigns all VM requests to the first available PM after first sorting them according to their non-increasing process times;
2. The theoretical lower bound, known as the optimal solution (OPT), is calculated by multiplying the total of the minimum number of machines required over all time slots by the duration of each slot. We consider that all VMs completely utilize the requested capacity (the worst case). The simulations are executed ten times for each set of VM queries. The average of the 10 runs is used to calculate all the findings.

Simulation using real traces

We use the easily accessible Lawrence Livermore National Lab Thunder log from the Parallel Workloads Archive [27] to model incoming VM requests because there is a lack of information from actual Cloud data centers regarding the energy use of computing resources. The Lawrence Livermore National Laboratory has a sizable Linux cluster dubbed Thunder that is responsible for gathering this log. We can extract pertinent information from the log that is compatible with our problem model, such as the request number, start-time, desired time, and requested processor count. Because our simulation's time slots are set at one minute, we convert seconds—the time unit used in the log—to minutes. Additionally, we adjust the quantity of processors required to fit the eight categories of VM demands indicated in Table 7.2. We perform the simulations with an adequate number of PMs to allow for the successful allocation of all VM requests without any rejections.

Figures 7.4–7.6 display, for the growing number of VM requests, the total busy time (in minutes), total energy usage (in kilowatt hours), and total simulation time (in milliseconds) (from 1000 to 7000).

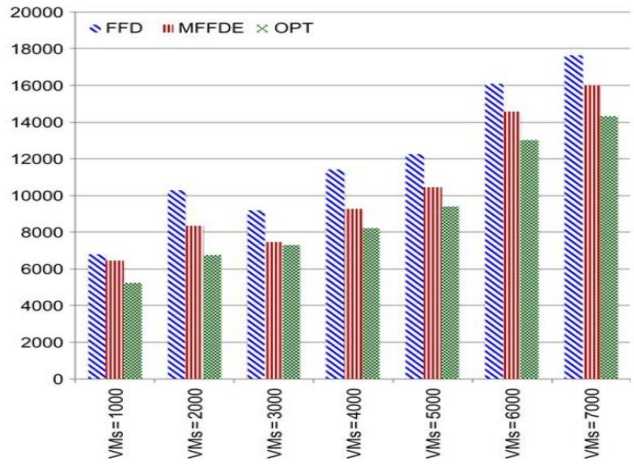

Figure 4 Total busy time (min) for increasing number of VM requests.

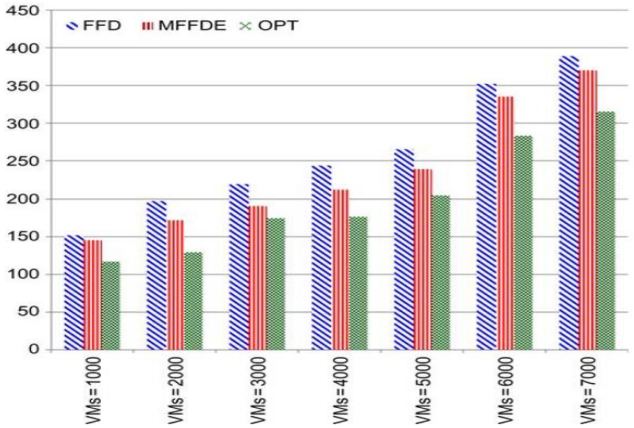

Figure 5 Total energy consumption (kWh) for increasing number of VM requests.

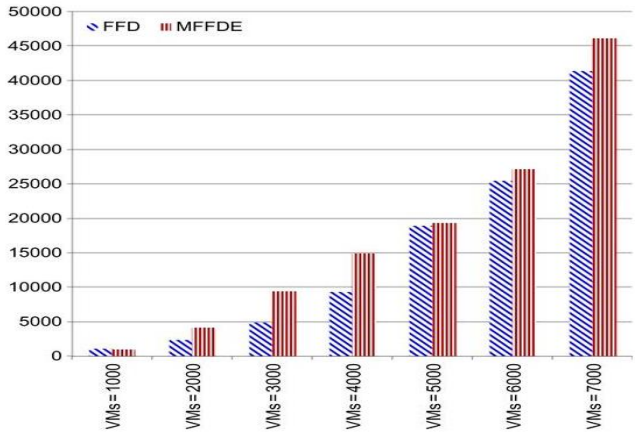

Figure 6 Total simulation time (ms) for increasing number of VM requests.

Simulation with artificial data

Data center energy consumption evaluation: the total busy time (in minutes) and total energy use (in kilowatt hours), respectively, for the rising maximum duration (in slots) of VM requests are shown in Figures 7.7 and 7.8. (From 50 to 800). Results from the MFFDE are not more than three times those from the best solution (OPT). This supports our theoretical conclusions and findings regarding overall energy use. All queries have exponential service times and follow the Poisson arrival pattern. The maximum intermediate interval between two arrivals is set at 50 slots, the mean interarrival interval is set at 5, and the maximum duration of requests is set at 50, 100, 200, 400, and 800 slots, respectively. The total arrivals (VM requests) are 1000, there are 125 requests for each sort of VM, and there are 60 PMs (20 for each PM type). Each slot lasts for 5 minutes. For instance, if a virtual machine (VM) has 20 slots of requested time (service time), its real time is 20 x 5 = 100 minutes.

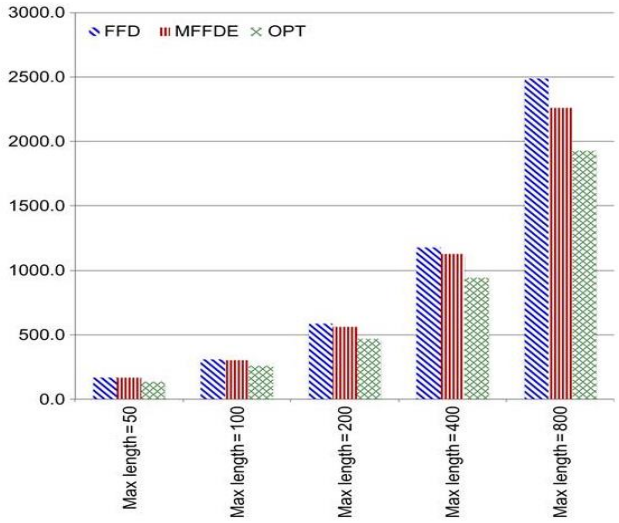

Figure 7. Total busy time (min) for increasing maximum duration (slots) of VM requests.

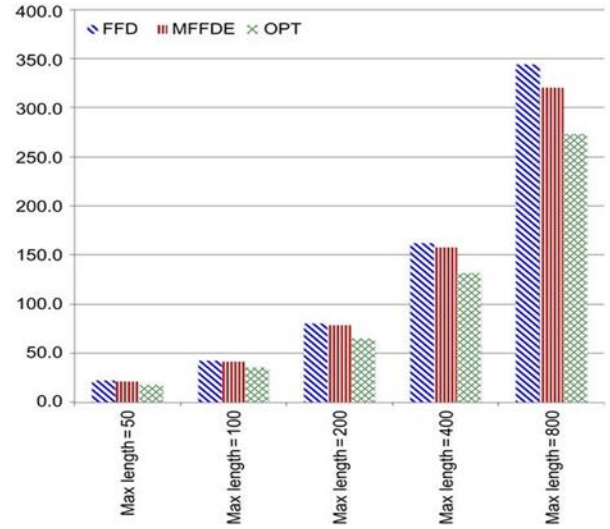

Figure 8 Total energy consumption (kWh) for increasing maximum duration (slots) of VM requests.

V. CONCLUSION AND FUTURE SCOPE

The best-known bounds for multiple machine scheduling are improved in this chapter. According to [2,12], there is currently no polynomial time solution for the problem of scheduling all jobs nonpreemptively inside their start-time-end-time windows while reducing the overall busy time of all machines under the constraints of machine capacity. We put forth an approximation algorithm, the MFFDE, which in the general case is a 3-approximation and is nearly optimal in the special and average instances. The MFFDE algorithm can be used to increase energy efficiency in Cloud computing and other related fields. The MFFDE algorithm is a solid approximation bound for minimizing the maximum makespan while minimizing the overall busy time because it combines aspects of the FFD strategy (biggest process time first) and the ESTF algorithm. The makespan will not be sacrificed in order to reduce the overall busy time Approximation bound for minimizing the maximum makespan while minimizing the overall busy time because it combines aspects of the FFD strategy (biggest process time first) and the ESTF algorithm. The makespan will not be sacrificed in order to reduce the overall busy time.

ACKNOWLEDGMENT

My acknowledgment is addressed to anyone who contributes and gave the input and constructive explanations on the improvements of the paper.

AUTHOR CONTRIBUTION

Sebagenzi Jason proposed and designed Energy Efficient Offline Parallel Scheduling in Cloud Computing by Reducing Total Busy Time added to the existing techniques for power management in cloud computing.

REFERENCES

[1]. Hoogeveen JA, van de Velde SL, Veltman B. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Appl Math*. Vol.**55,** Issue.**3,** pp.**259–272, 1994.**

[2]. Khandekar R, Schieber B, Shachnai H, Tamir T. *Minimizing busy time in multiple machine real-time scheduling*. *Proceedings of IARCS annual conference on foundations of software technology and theoretical computer science (FSTTCS 2010)*. Chennai, India: LIPIcs, Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik; Vol.**8**, pp.**169–180, 2010.**

[3]. Brucker P. *Scheduling algorithms* 5th ed. Berlin, Heidelberg, New York: Springer; **2007.**

[4]. Beloglazov A, Buyya R, Lee YC, Zomaya AY. A taxonomy and survey of energy-efficient data centers and Cloud computing systems. *Adv Comput*. Vol.**82**, pp.**47**–**111**, **2011.**

[5]. Jing SY, Ali S, She K, Zhong Y. State-of-the-art research study for green Cloud computing. *J Supercomput*, pp.**1–24**, **2011**.

[6]. Graham RL. Bounds on multiprocessing timing anomalies. *SIAM J Appl Math*., Vol.**17,** Issue.**2**, pp.**416–429, 1969.**

[7]. Kleinberg JM, Tardos E. *Algorithm design* Boston, MA: Addison-Wesley: Pearson Education, Inc.; **2006.**

[8]. Kovalyov MY, Ng CT, Cheng TCE. Fixed intervals scheduling: models, applications, computational complexity and

algorithms. *Eur J Oper Res*., Vol.**178,** Issue.**2,** pp.**331–342, 2007.**

[9]. Beloglazov A, Abawajy JH, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Gen Comput Syst*. Vol.**28**, Issue.**5**, pp.**755–768, 2012.**

[10]. Winkler P, Zhang L. *Wavelength assignment generalized interval graph coloring*. *Proceedings of fourteenth annual ACM-SIAM symposium on discrete algorithms (SODA 2003)* Baltimore, MD: ACM/SIAM; pp.**830–831, 2003.**

[11]. Flammini M, Monaco G, Moscardelli L, et al. Minimizing total busy time in parallel scheduling with application to optical networks. *Theor Comput Sci*., Vol.**411**, Issue.**40–42**, pp.**3553–3562, 2010.**

[12]. Kolen AW, Lenstra JK, Papadimitriou CH, Spieksma FC. Interval scheduling: a survey. *Nav Res Log*., Vol.**54**, Issue.**5**, pp.**530–543, 2007.**

[13]. Srikantaiah S, Kansal A, Zhao F. *Energy aware consolidation for Cloud computing*. *Proceedings of USENIX workshop on power aware computing and systems (HotPower 2008)* San Diego, CA: USENIX; pp.**10–14, 2008.**

[14]. Lee YC, Zomaya AY. Energy efficient utilization of resources in Cloud computing systems. *J Supercomput*. Vol.**60**, Issue.**2**, pp.**268–280, 2012.**

[15]. Liu H, Xu C, Jin H, Gong J, Liao X. Performance and energy modeling for live migration of virtual machines, the 20th international symposium on high performance distributed computing. pp.**171–182, 2011.**

[16]. Kim KH, Beloglazov A, Buyya R. Power-aware provisioning of virtual machines for real-time Cloud services. *Concurr Comput Pract Exp*. Vol.**23**, Issue.**13**, pp.**1491–1505, 2011.**

[17]. Mathew V, Sitaraman RK, Shenoy PJ. *Energy-aware load balancing in content delivery networks*. *Proceedings of IEEE INFOCOM* Orlando, FL: IEEE; pp.**954–962, 2012.**

[18]. Rao L, Liu X, Xie L, Liu W. *Minimizing electricity cost: optimization of distributed internet data centers in a multi-electricity-market environment*. *Proceedings of IEEE INFOCOM 2010* San Diego, CA: IEEE; pp.**1145–1153, 2010.**

[19]. Lin M, Wierman A, Andrew LLH, Thereska E. *Dynamic right-sizing for power-proportional data centers*. *Proceedings of IEEE INFOCOM 2011* Shanghai, China: IEEE; pp.**1098–1106, 2011.**

[20]. Mertzios GB, Shalom M, Voloshin A, Wong PWH, Zaks S. *Optimizing busy time on parallel machines*. *Proceedings of IPDPS* Shanghai, China: IEEE; pp.**238–248, 2012.**

AUTHOR PROFILES

Sebagenzi Jason pursed Bachelor of science from Adventist University of Central Africa (AUCA-Rwanda), and Master of Science in information Technology from Jain University (India) in year 2021. He is currently Ph.D. In computer science major in cloud computing from Jain University (India) and the Dean of Information Technology Faculty.

