I.J. Information Technology and Computer Science, **2019, 1, 1-13**
Published Online January 2019 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2019.01.01

A Task Scheduling Model for Multi-CPU and Multi-Hard Disk Drive in Soft Real-time Systems

Zeynab Mohseni[1,¤], **Vahdaneh Kiani**[1,¤]
[1]Department of Computer Engineering, Science and Research Branch, Azad University, Iran
[¤]These authors have contributed equally to this article as co-first authors
E-mail: amohseni.vkiani@gmail.com

Amir Masoud Rahmani
Department of Computer Engineering, Science and Research Branch, Azad University, Iran
Computer Science, University of Human Development, Sulaimanyah, Iraq
E-mail: rahmani@srbiau.ac.ir

Received: 16 August 2018; Accepted: 22 September 2018; Published: 08 January 2019

***Abstract*—**In recent years, by increasing CPU and I/O devices demands, running multiple tasks simultaneously becomes a crucial issue. This paper presents a new task scheduling algorithm for multi-CPU and multi-Hard Disk Drive (HDD) in soft Real-Time (RT) systems, which reduces the number of missed tasks. The aim of this paper is to execute more parallel tasks by considering an efficient trade-off between energy consumption and total execution time. For study purposes, we analyzed the proposed scheduling algorithm, named HCS (Hard disk drive and CPU Scheduling) in terms of the task set utilization, the total execution time, the average waiting time and the number of missed tasks from their deadlines. The results show that HCS algorithm improves the above mentioned criteria compared to the HCS_UE (Hard disk drive and CPU Scheduling _Unchanged Execution time) algorithm.

***Index Terms*—**Non-preemptive task scheduling, soft real-time system, Task parallelism, Multi-CPU, Multi-device.

I. Introduction

In recent years, more energy is consumed due to the increasing demands and the development of embedded systems used to complicate computing devices such as laptop and smartphone. Therefore, reducing the energy consumption while the task set can still meet the deadlines is an important issue. Different technologies are proposed to reduce the CPU energy consumption, such as Dynamic Frequency Scaling (DFS), Dynamic Voltage Frequency Scaling (DVFS), which determine the operating frequency of the processors [1-12]. Another major technique for energy conservation is Dynamic Power Management (DPM) that is used to reduce power consumption of off-chip devices such as HDD by switching a device from the active to the energy sleep state [13]. In this work, the number of missed tasks is reduced without considering energy saving. In order to achieve energy saving, a strategy is proposed to minimize the idle times of CPUs and hard disk drives by reducing the frequency of CPUs and/or Revolutions Per Minute (RPM) level of hard disk drive.

In real-time systems, a service request is responded within a certain amount of time. A timing constraint includes a hard real-time [3, 12-15] or a soft real-time [9] based on the importance of the deadline in the missed tasks. The hard real-time system offers guaranteed services because the missed task is completely unacceptable, whereas in a soft real-time system a request is completed within a known finite time.

Due to the improvement of technology in the processor design, the processor now consists of innumerable cores, called as multi-core processor [16, 11]. The multi-core processor can reduce the total execution time and the number of missed tasks. Since the system demands for a CPU and I/O devices are increased, the parallelism on systems should be considered to design a scheduling strategy. It is performed by composing of a CPU and multi-device [17].

In this paper, a task scheduling model is proposed to compose of multi-core processors and multi-hard disk drive in soft real-time systems. This model reduces the number of missed tasks and the execution time of task set for CPU and hard disk drive requests.

The remainder of the paper is organized as follows. Section II presents related works and in section III the proposed algorithms are described. Performance evaluation and conclusion are presented in Sections IV and V, respectively.

II. Related Work

In recent years, a significant number of studies have been proposed in the field of special-purpose systems for scheduling real-time tasks. DVFS technique was used to improve energy or power consumption in these studies. A

mechanism was proposed to emulate a precise CPU frequency by using the DVFS management in virtualized environments [1]. In [14] the focus was on a scheduling approach towards the sporadic task set in the uni-processor system according to DVFS technique. In [5], A DVFS-based algorithm was presented to reduce energy consumption of processors through efficient use of the generated tasks' slack times by an independent scheduler. In order to reduce the power consumption, a scheduling algorithm for DVFS- enabled clusters for executing multiple virtual machines was proposed in [6]. Moreover, in order to save energy, several mechanisms for scheduling of real-time tasks on Dynamic Voltage Scaling (DVS) processor were introduced in [8, 18 and 12]. Additionally, a mechanism for scheduling of tasks on a non-ideal DVS processor with shared resources in order to obtain better energy efficiency was presented in [8]. In [12] a pre-runtime scheduling for hard RT systems based on time Petri nets in order to find a feasible schedule that satisfies the timing and energy constraints was presented. These above articles considered uni-processor in their proposed scheduling algorithms. In few other studies such as [16, 11, 19, 20], they presented a CPU scheduling on multi-core and multi-processor. In [11], the focus was on the energy-efficient scheduling of periodic RT tasks on multi-core processors, which cores are partitioned into multiple blocks. These blocks are known as voltage islands. The voltage of cores in each island is the same and it can be adjusted by DVFS technique. Two new partitioned approaches for scheduling real-time sporadic tasks on platform under RMS, in order to improve the performance were presented in [16].

Many case studies have been done on the reduction of the energy consumption of the hard disks. A RT scheduling was presented in [15], which was extended for intra-task devices with multiple sleep states to further minimize the overall device energy consumption of the system. Here, the energy saving was achieved by switching from active to sleep mode. In [21-23], the mechanisms were proposed in order to reduce the power consumption and the energy consumption of devices without considering the scheduling algorithms for a task set of real-time systems regardless of missed tasks. Additionally, a scheduling algorithm for a set of real-time tasks with I/O requests based on DPM technique in RT systems was presented in [13].

In [17], they focused on single processor and multiple off-chip devices. A frame-based RT task model for minimizing the energy consumption by combining the DVS and DPM techniques was presented. It was done based on changing the CPU frequency and transitioning the devices to sleep state when they are not in use [18]. All explained works exclusively focus on reducing the energy consumption, whereas improving the performance (e.g. utilization and total execution time) is also a desirable outcome. Furthermore, none of the papers considered multi-CPU and multi-device scheduling in combination. On the other hand, they minimized the energy consumption of hard disk drives just by using the DPM technique.

A memory access control framework called BWLOCK, was designed in [24] to protect MPCSs (Memory-Performance Critical code Sections) of soft real-time applications. The focus of the paper was on protecting real-time performance of the evaluated applications in the existence of co-running memory intensive non-real-time applications.

In this paper, a novel scheduling algorithm is presented by using multi-CPU and multi-device to improve the performance of system considering the energy saving.

III. Proposed Algorithms

In this section, a summary of our previous work is presented [25] in sub-section A. In sub-section B, the new algorithms is extended. The system consists of soft real-time task set, represented as $T_L = \{T_1, T_2, .., T_N\}$. A task $T \in T_L$ has 3-tuple (R_t, D_t, $E_{t_{task}}$) where R_t is the ready time of the task, D_t is the deadline of the task and $E_{t_{task}}$ is the execution time of the task. In addition, all tasks are assumed to be non-preemptive. Fig. 1 shows the process of running algorithms.

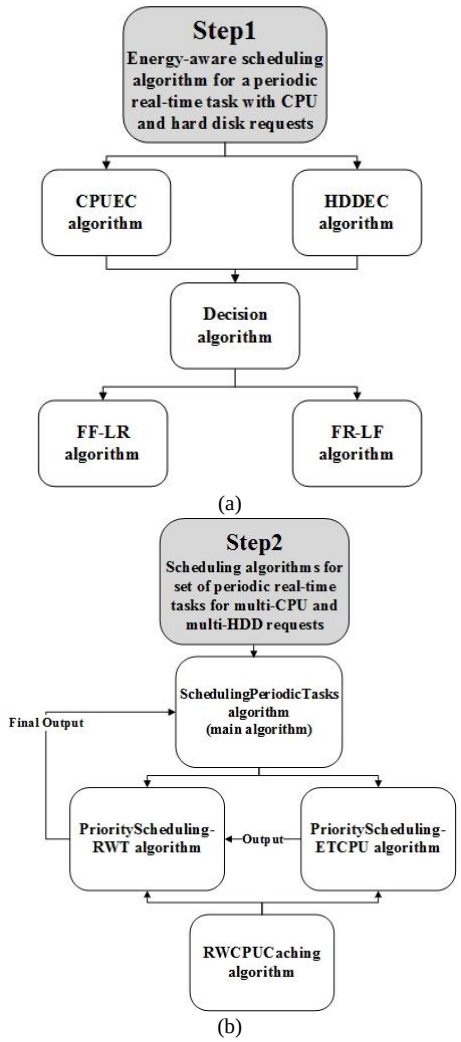


Fig.1. The process of running the HCS algorithm (a) previous algorithms [25] (b) new algorithms

A. Energy-aware scheduling algorithm for a periodic real-time task with CPU and hard disk requests

The main goal of our previous work was to present a scheduling mechanism for a real-time periodic task that can save more energy. This mechanism was based on increasing the execution time of the CPU and/or the Read/Write(R/W) time of the hard disk drive, as much as possible, without passing the task deadline. Five algorithms including the CPU Energy Consumption (CPUEC), HDD Energy Consumption (HDDEC), Decision, FF-LR (First Frequency-Last RPM) and FR-LF (First RPM-Last Frequency) was presented in [25].

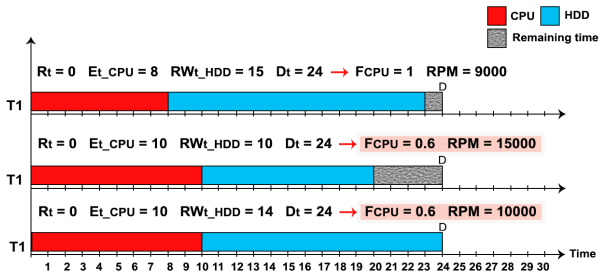
In Decision algorithm, according to obtained energy consumption of CPU and HDD, FF-LR or FR-LF algorithm was executed [25] (Fig.1.a). The execution time of modified tasks (output) from these two algorithms is considered as a set of input ideal tasks in SchedulingPeriodicTasks algorithm (Fig.1.b).

The Eqs. (1) to (11) used in [25] are shown in Table 1. The major parameters of the algorithms are described in Table 2.

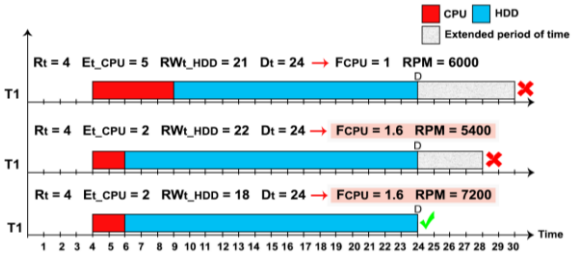
Table 1. The equations

Num.	Equations
1	$P = \lambda \times F_{CPU} \times V^2$
2	$E_{CPU} = P \times E_{t_{cpu}}$
3	$P = \text{Number of platters} \times (RPM)^{2.8} \times (\text{Diameter})^{4.6}$ [34]
4	$E_{HDD} = P \times T_t$
5	$RE_t = D_t - (E_{t_{cpu}} + RW_{t_{HDD}})$
6	$RW_{t_{HDD}} = RD_t + AS_t + T_t$
7	$RPM = (\frac{1}{2} \times 60)/ RD_t$
8	$EX_t = (E_{t_{cpu}} + RW_{t_{HDD}}) - D_t$
9	$E_{t_{cpu}} = \frac{\sum_{i=1}^{N} CPI_i}{F_{CPU}}$
10	$F_{ideal} = \frac{F_{min\text{-}cpu} \times E_{t_{cpu}}}{E_{t_{cpu}} - EX_t}$
11	$F_{ideal} = \frac{F_{max\text{-}cpu} \times E_{t_{cpu}}}{E_{t_{cpu}} + RE_t}$

The basic idea of FF-LR and FR-LF algorithms was to extend the execution time of CPU and/or the R/W time of the hard disk drive up until the deadline. This was performed by dynamically changing the CPU frequency and/or the RPM level of HDD [25]. Fig. 2 shows a sample of the previous work [25].



(a)



(b)

Fig.2. Samples of (a) FF-LR algorithm (b) FR-LF algorithm

Table 2. The major parameters

Parameter	Definition
E_{CPU}	CPU Energy consumption(J)
P	Power consumption
E_{HDD}	Hard Disk Drive Energy consumption(J)
R_t	Ready time(ms)
D_t	Deadline time(ms)
RE_t	Remaining time up until the deadline(ms)
EX_t	Extended period of time from the deadline(ms)
F_{CPU}	Frequency of CPU(GHz)
F_{ideal}	New frequency of CPU(GHz)
λ	Effective capacitance
V	Voltage of CPU(V)
$F_{min\text{-}cpu}$	Minimum frequency of CPU(GHz)
$F_{max\text{-}cpu}$	Maximum frequency of CPU(GHz)
$E_{t_{cpu}}$	CPU Execution time(ms)
$RW_{t_{HDD}}$	R/W time for hard disk drive(ms)
T_t	Transfer time(ms)
AS_t	Average Seek time in hard disk(ms)
RPM	Revolutions Per Minute in hard disk
IT_I	Set of input ideal tasks ($E_{t_{task}}$)
RD	Rotational delay(ms)
$E_{t_{task}}$	The execution time of modified task(ms)
IT_O	Set of output ideal tasks ($E_{t_{task}}$)
T_{cache}	R/W time in cache of CPU
FT_{CPU}	CPU finishing time
WI_t	Waiting time or Idle time
CR	Clock rate
D	Data path internal bus (bit)
N_{core}	Number of cores
N	Number of tasks

B. Scheduling algorithms for set of periodic real-time tasks for multi-core processors and multi-HDD requests

SchedulingPeriodicTasks algorithm is based on real-time scheduling for multi-CPU and multi-HDD requirements-based periodic multi-tasking with the aim of improving the performance (e.g. task set utilization). Note that each CPU has several cores, which is known as multi-core processor. The SchedulingPeriodicTasks algorithm includes the PriorityScheduling-ETCPU and PriorityScheduling-RWT algorithms which is described in section B.a. At the end of the SchedulingPeriodicTasks algorithm, the task set is scheduled. The RWCPUCaching algorithm is explained in section B.b. In this algorithm the cache time is calculated. Section B.c presents the CPU and HDD scheduling.

a. SchedulingPeriodicTasks algorithm

As explained in section A, the ideal execution time of each task ($E_{t_{task}}$) is calculated. The SchedulingPeriodicTasks algorithm is shown in Fig.3. As can be seen, the execution time of each task are considered as the input tasks of this algorithm and they are put in IT_I. In order to run the PriorityScheduling-ETCPU algorithm, the tasks sort in an ascending order based on their ready time. If some tasks have the same ready time, they sort in ascending order based on the execution time of CPU. Then PriorityScheduling-ETCPU algorithm executes based on the sorted tasks. In the next stage of the algorithm, the scheduled tasks sort in ascending order based on FT_{CPU}. If some tasks have the same FT_{CPU}, they sort in ascending order by R/W time of HDD. Finally, PriorityScheduling-RWT algorithm executes.

1. **Input**: R_t, $E_{t_{cpu}}$, $E_{t_{task}}$, $RW_{t_{HDD}}$
2. **Output:** IT_O
3. Get each $E_{t_{task}}$ from either FF-LR algorithm or FR-LF algorithm and put it in IT_I
4. Sort the set of $E_{t_{task}} \epsilon\ IT_I$ in ascending order by the R_t
5. **If** R_t for some of tasks are equal **then**
6. Sort the set of $E_{t_{task}}$ in ascending order by the $E_{t_{cpu}}$
7. **end if**
8. Run PriorityScheduling-ETCPU algorithm
9. Get IT_O and FT_{CPU} from PriorityScheduling-ETCPU algorithm
10. Sort the set of $E_{t_{task}}$ in ascending order by the FT_{CPU}
11. **If** FT_{CPU} for some of tasks are equal **then**
12. sort the set of $E_{t_{task}}$according to the $RW_{t_{HDD}}$
13. **end if**
14. Run PriorityScheduling-RWT algorithm

Fig.3. SchedulingPeriodicTasks

b. RWCPUCaching algorithm

In order to reduce the execution time of tasks, multi-core CPU with their own cache memory is considered. All cache memories are located on the CPU or into a chip on the system board. Cache stores data to accelerate the access time of requests in the future. Cache memory has two states, which are cache hit and cache miss. If the requested data is found in cache, the cache hit occurs, otherwise cache miss occurs [26]. RWCPUCaching algorithm, which is presented in Fig. 4 is considered in Figs. 5 and 6. Since the write-through cache is assumed, the cache writes data to both cache and storage. The advantage to this approach is that newly written data is always cached thereby allowing the data to be read quickly. Also, the values of transfer rates of CPU and HDD are considered according to [27]. The transfer rate of CPU is calculated from Eq. (12) where D, CR and N_{core} represent the data path internal bus, clock rate of CPU, and the number of CPU cores, respectively. Also, the transfer rate of HDD is 0.75 GB/s (Table 4).

$$\text{Transfer Rate of CPU} = \text{CR} \times \text{D}/8 \times \text{N}_{\text{core}} \quad (12)$$

In Eq. (13), X is the fraction of transfer rate of CPU to transfer rate of HDD and $RW_{t_{HDD}}$ is the R/W time of HDD. In other words, X represents how much CPU is faster than the hard disk drive.

$$\text{X} = \frac{\text{Transfer Rate of CPU}}{\text{Transfer Rate of HDD}} \quad (13)$$

According to Eqs. (12) and (13) the cache time (T_{cache}) is calculated (Eq. (14)).

$$\text{T}_{\text{cache}} = \text{RW}_{\text{t}_{\text{HDD}}}/\text{X} \quad (14)$$

In this algorithm, according to R/W mode and the cache states, one of the following modes will occur.

- The execution time of task is in write mode
- The execution time of task is in read mode and cache miss occurs
- The execution time of task is in read mode and cache hit occur

In all cases, the cache time of CPU is calculated. In the last case, the R/W time of HDD sets to zero and the CPU cache reads data, whereas in the two first cases the R/W time of HDD do not change (HDD reads and writes the data when the execution time of task is in read mode and write mode, respectively). This approach leads to the reduction in the execution time of tasks remarkably.

1. **Input:** $RW_{t_{HDD}}$, write_mode, read_mode, $E_{t_{task}}$
2. **Output:** $RW_{t_{HDD}}$, T_{cache}
3. Set T_{cache} to zero
4. Use Eq.(12) calculate X
5. Use Eq.(13) calculate T_{cache}
6. **If** $E_{t_{task}}$ is in write_mode **or** $E_{t_{task}}$ is read_mode and a cache miss occur **then**
7. $RW_{t_{HDD}}$ is not changed
8. **else if** $E_{t_{task}}$ is in read_mode and cache hit is occurred **then**
9. Set $RW_{t_{HDD}}$to zero
10. **end if**

Fig.4. RWCPUCaching

c. CPU and HDD Scheduling

In this section, CPU and HDD scheduling is done to improve the utilization and also reduce the execution time of tasks, the waiting time of tasks and the number of missed tasks as much as possible. Furthermore, the energy consumption is considered by extending the execution time of CPU and R/W time of HDD when the CPU and HDD are in idle mode. So, based on the use of this approach, increasing the energy consumption is not significant.

a) PriorityScheduling-ETCPU algorithm

In Fig.5 first, according to the number of free CPU, RWCPUCaching algorithm is called to calculate the execution time of CPU as follows.

$$E_{t_{cpu}} = E_{t_{cpu}} + T_{cache} \quad (15)$$

Then the free CPUs assigns to the sorted tasks that are in IT_I. If the free CPU is not available to assign the unscheduled task (current task), one of the CPUs that are assigned to the scheduled tasks considers. So, the scheduled task with minimum execution time of CPU (selected task) selects. Afterward the waiting time or idle time of the current task calculates from Eq. (16).

$$WI_t = E_{t_{cpu}} - R_t \quad (16)$$

Where WI_t , $E_{t_{cpu}}$ and R_t are the waiting time or idle time, execution time of selected task and ready time of the current task, respectively. Based on the calculated WI_t, the frequency of selected task changes from Eq. (17). If WI_t is greater than zero, it means that there is waiting time. Thus, the frequency of selected task increases in order to reduce its execution time. This work leads to the reduction in the waiting time of the current task. If WI_t is lower than zero, it means that there is an idle time of CPU. So, the frequency of selected task decreases in order to increase its execution time. This approach leads the reduction in the idle time of CPU that assign to the selected task.

$$F_{ideal} = \frac{F_{CPU} \times E_{t_{cpu}}}{R_t} \quad (17)$$

After the calculation of the new frequency, the new execution time of selected task obtains from Eq. (18), where N and CPI represent the total number of instructions and clock cycle per instruction, respectively. In order to add the cache time to the CPU execution time of current task, the RWCPUCaching algorithm calls and then the execution time of current task calculates from Eq. (15). Then the CPU assigns to the current and selected tasks, according to their execution times. Finally the CPU finishing time of the tasks calculate according to Eq. (19). To calculate FT_{CPU} of current task, in case of waiting time, the WI_t is considered in the Eq. (19), otherwise it is set to zero.

$$E_{t_{cpu\_new}} = \frac{\sum_{i=1}^{N} CPI_i}{F_{ideal}} \quad (18)$$

$$\begin{cases} \text{if selected task} \quad FT_{CPU} = E_{t_{cpu\_new}} + R_t \\ \text{if current task} \quad FT_{CPU} = E_{t_{cpu}} + R_t + WI_t \end{cases} \quad (19)$$

Finally, the CPUs update according to the scheduled tasks and the same strategy is followed for scheduling the next tasks.

1. **Input**: R_t, $E_{t_{cpu}}$, IT_I
2. **Output:** IT_O, FT_{CPU}
3. **While** there is a free CPU **do**
4. Call RWCPUCaching algorithm to calculate T_{cache} // $RW_{t_{HDD}}$ is unused
5. Use Eq. (15) to calculate $E_{t_{cpu}}$
6. Schedule $E_{t_{task}} \epsilon IT_I$ on one of the free CPU
7. **end while**
8. **If** there is not a free CPU **and** there is an unscheduled task **then**
9. Select the scheduled $E_{t_{task}} \epsilon IT_I$ that has the minimum $E_{t_{cpu}}$
10. F_{CPU} changes to F_{ideal} (from Eq. (17))
11. Use Eq.(18) to Calculate $E_{t_{cpu}}$ of selected $E_{t_{task}}$ according to new frequency
12. Call RWCPUCaching algorithm to calculate T_{cache} // $RW_{t_{HDD}}$ is unused
13. Use Eq. (15) to calculate $E_{t_{cpu}}$
14. Schedule selected $E_{t_{task}}$ on CPU according to calculated $E_{t_{cpu}}$
15. Schedule current $E_{t_{task}}$ on CPU according to $E_{t_{cpu}}$
16. Get the last value of $E_{t_{task}}$ and put it in IT_O
17. Update CPU
18. **end if**
19. Use Eq.(16) to calculate WI_t for $E_{t_{task}}$
20. Use Eq.(19) to calculate FT_{CPU} of $E_{t_{task}}$

Fig.5. PriorityScheduling-ETCPU

b) PriorityScheduling-RWT algorithm

In Fig.6, the PrioritySheduling-RWT algorithm is explained. First, according to the number of free HDDs, RWCPUCaching algorithm is called to calculate the R/W time of hard disk drive. After running the RWCPUCaching algorithm, the R/W time sets to zero or it does not change. Then the free hard disk drives assigns to the sorted tasks. If the free HDD is not available to assign the unscheduled task, one of the scheduled tasks with minimum R/W time of hard disk drive is selected. Then the waiting time of the current task or idle time of the hard disk drive is calculated from Eq. (20).

$$WI_t = RW_{t_{HDD}} - FT_{CPU} \quad (20)$$

Where $RW_{t_{HDD}}$ and FT_{CPU} are the R/W time of selected task and the CPU finishing time of the current task, respectively. If there is a waiting time and a RPM level that is greater than the RPM level of selected task, the R/W time of selected task is calculated according to the new selected RPM level from Eq. (6). The calculated R/W time of hard disk drive is compared to the CPU finishing time of the current task. If the R/W time is greater than the CPU finishing time, the RPM level increases and the R/W time is calculated again. If there is an idle time of HDD and the task is not missed, the RPM level is decreased by prolonging the R/W time of selected task. Decreasing RPM level continues until the R/W time

is lower than the CPU finishing time of the current task. In order to decrease the changed RPM level of hard disk drive that is assigned to selected task, the R/W time of current task is calculated according to the RPM level of selected task ($RW_{t_{HDD\text{-}new}}$). Then the calculated R/W time is compared to the R/W time based on its RPM level ($RW_{t_{HDD}}$). According to this comparison and the deadline of current task, one of the if-else clauses executes. The process descriptions is explained in detail from line 27-40 with the aim of not changing the RPM level, as much as possible. Increasing the number of change in the RPM level leads to increasing in the energy consumption.

1. **Input**: $RW_{t_{HDD}}$, D_t,IT_O, FT_{CPU}
2. **Output:** IT_O
3. **While** there is a free HDD **do**
4. Call RWCPUCaching algorithm to calculate $RW_{t_{HDD}}$ // T_{cache} is unused
5. Schedule $E_{t_{task}} \epsilon\ IT_O$ on one of the free HDD
6. **end while**
7. **If** there is not a free HDD **and** there is an unscheduled task **then**
8. Select the scheduled $E_{t_{task}} \epsilon\ IT_O$ that has the minimum $RW_{t_{HDD}}$
9. Use Eq.(20) to calculate WI_t for selected $E_{t_{task}}$
10. **If** WI_t>0 **then**
11. **While** there is RPM level is greater than the RPM of selected $E_{t_{task}}$ **do**
12. Use Eq.(6) to Calculate $RW_{t_{HDD}}$ of selected $E_{t_{task}}$ according to this RPM level
13. **If** $RW_{t_{HDD}}$ <= FT_{CPU} **then**
14. Consider $RW_{t_{HDD}}$ of selected $E_{t_{task}}$ with this RPM level
15. break
16. **end if**
17. **end while**
18. **else if** WI_t< 0 **and** $RW_{t_{HDD}}$ - D_t > 0 **then**
19. **While** there is RPM level is lower than the RPM of selected $E_{t_{task}}$ **do**
20. Use Eq.(6) to calculate $RW_{t_{HDD}}$ of selected $E_{t_{task}}$ according to this RPM level
21. **If** $RW_{t_{HDD}}$ <= FT_{CPU} **then**
22. Consider $RW_{t_{HDD}}$ of selected $E_{t_{task}}$ with this RPM level
23. **end if**
24. **end while**
25. **end if**
26. Use Eq.(6) to calculate $RW_{t_{HDD\text{-}new}}$ according to RPM level of selected $E_{t_{task}}$
27. **If** ($RW_{t_{HDD\text{-}new}}$ < D_t **and** $RW_{t_{HDD}}$ < D_t) **or** ($RW_{t_{HDD\text{-}new}}$ < D_t **and** $RW_{t_{HDD}}$ > D_t) **then**
28. Replace $RW_{t_{HDD}}$ in $E_{t_{task}}$ with $RW_{t_{HDD\text{-}new}}$ and Call RWCPUCaching algorithm
29. **else if** $RW_{t_{HDD\text{-}new}}$ > D_t **and** $RW_{t_{HDD}}$ > D_t **then**
30. **If** the current RPM is the maximum level **then**
31. $RW_{t_{HDD}}$ is missed
32. **else**
33. Select the first RPM level is greater than RPM of current and selected $E_{t_{task}}$ up until the $E_{t_{task}}$ is not missed
34. Use Eq.(6) to calculate $RW_{t_{HDD}}$ with this RPM level and Call RWCPUCaching algorithm
35. **end if**
36. **else if** $RW_{t_{HDD\text{-}new}}$ > D_t **and** $RW_{t_{HDD}}$ < D_t **then**
37. Call RWCPUCaching algorithm
38. **end if**
39. Schedule selected $E_{t_{task}}$ on HDD according to calculated $RW_{t_{HDD}}$
40. Schedule current $E_{t_{task}}$ on HDD according to $RW_{t_{HDD}}$
41. Get the last value of $E_{t_{task}}$ and put it in IT_O
42. Update HDD

end if

Fig.6. PriorityScheduling-RWT

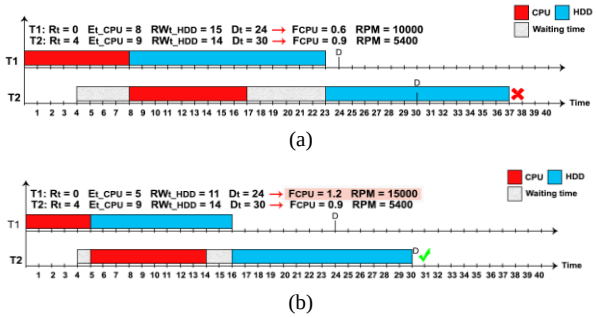


Fig.7. Sample of task scheduling for two tasks (Increasing the frequency of CPU and RPM level of HDD) in (a) HCS_UE algorithm, (b) HCS algorithm

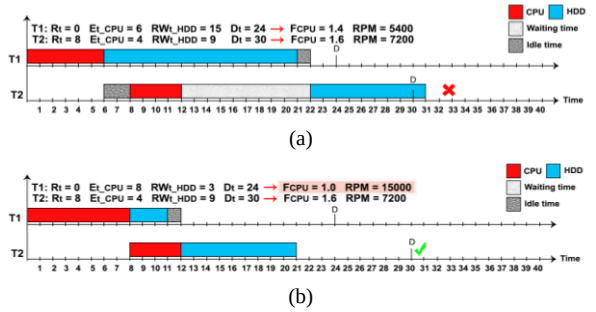


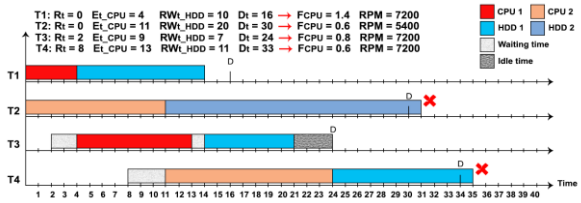
Fig.8. Sample of task scheduling for two tasks (Decreasing the frequency of CPU and increasing the RPM level of HDD) in (a) HCS_UE algorithm, (b) HCS algorithm

In the next stage of the algorithm, the HDD assigns to the current and selected tasks, according to their R/W times. Finally the HDDs updates and the same strategy is followed for the next tasks scheduling.

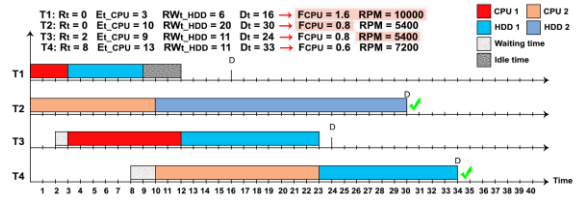
Figs.7 and 8 show two samples of two scheduled tasks by applying the HCS_UE and HCS algorithms. Fig 7. (b) shows that to reduce the waiting time of T2 (current task), the execution time of T1 (selected task) is reduced by increasing the CPU frequency and the RPM level of HHD. As a result, the waiting time and the number of missed tasks in HCS algorithm are reduced compared to the HCS_UE algorithm (Fig.7. (a)). In Fig.8. (b), the CPU execution time of T1 is extended (decrease the CPU

frequency) with the aim of reducing the idle time of CPU and also the R/W time of T1 is reduced by increasing the RPM level of HHD. So, T2 is not missed in HCS algorithm compared to the HCS_UE algorithm (Fig.8. (a)). In Figs.7, 8 and 9, the values are considered for a better understanding of the algorithms.

Fig.9 is a sample of task scheduling in HCS and HCS_UE algorithms for two CPUs, two hard disk drives and four tasks. This figure shows all of the states including the reduction of waiting time of tasks, idle time of CPUs and hard disk drives. As a result, the number of missed tasks is reduced and the performance is improved.



(a)



(b)

Fig.9. Sample of task scheduling for 4 tasks in (a) HCS_UE algorithm, (b) HCS algorithm

IV. Performance Evaluation

In this section, the simulation results are provided to demonstrate the effectiveness of HCS algorithm. Since the focus is on scheduling the real-time multi-task with multi-CPU and multi-HDD requests and previous works were merely focused on task scheduling algorithm to reduce the energy consumption, this work incorporates novel approach in improving the performance of system with both multi-CPU and multi-HDD requests. This improvement is based on reducing the waiting times of tasks that leads to the reduction in average execution time, total execution time, number of missed task and average waiting time. It should be mentioned that increasing of the frequency of CPUs and/or the RPM level of hard disk drives reduces the waiting time. In order to reduce the loss of energy, we try to remove, as much as possible, the idle times of resources by reducing the frequency of CPUs and/or the RPM level of hard disk drives.

As explained in section II, all the papers were focused on multi-CPU scheduling or multi-device scheduling. Whereas, this paper is the combination of both multi-CPU and multi-device scheduling. Therefore, HCS algorithm is compared with HCS_UE algorithm. In HCS_UE algorithm the execution time of tasks is not changed and the tasks are scheduled according to the ready time and the execution time for CPU requests and also the tasks are scheduled by CPU finishing time and R/W time for HDD requests. In the simulation, we assume that the tasks are scheduled by higher priority CPU requests in comparison with HDD requests.

The proposed algorithm is evaluated by using 50, 100 and 200 tasks. Furthermore, multi-CPU and multi-HDD are represented as C and H, respectively. C is the number of CPU and H is the number of HDD that are varied from 4 to 64. Table3 summarizes the specifications of CPU and HDD that are used in the simulation. MATLAB [28-30] is used to develop the simulation using Intel® Xeon® Processor E5-2670 (30M Cache, 2.30 GHz) [31-32] and HP 300GB-6G-SAS 15k RPM-SFF (2.5inch) [33].

Table 3. Specification of CPU and hard disk drive

HP 300GB-6G-SAS 15k RPM-SFF(2.5inch)	
Name	**Value**
Capacity	300,000MB
Interface	SAS
Transfer Rate Synchronous(Maximum)	6 Gb/sec
Physical Configuration	Bytes/Sector 512 Logical Blocks 585,937,500 Rotational Speed: 5400(Min), 7200, 10,000 and 15,000(Max)rpm
Operating Temperature(System Inlet Air Temperature)	50 °to 95 °F (10 °to 35 ° C)
Number of Platters	2
Intel® Xeon® Processor E5-2670 (30M Cache, 2.30 GHz)	
Name	**Value**
High Frequency Mode(HFM)	Frequency 3.1 GHz Voltage 1.3V TDP(Thermal Design Power)
Low Frequency Mode(LFM)	Frequency 2.3 GHz Voltage 0.65V TDP(Thermal Design Power)

The parameters of the simulation are shown in Table4. As can be seen ready time (R_t), deadline time (D_t), are the main characteristics for each task, transfer time (T) and Average Seek time (AS_t) are the main characteristics for hard disk drive and CPU Execution time ($E_{t_{cpu}}$) is the main characteristic for the processor. In the simulation, the HDD and CPU characteristics are randomly generated within the specific ranges that are obtained from [31-33]. Also, the ready time and deadline time are generated randomly. However, I should be mentioned that this work can be applied on any kind of CPUs and hard disk drives.

Table 4. Simulation parameters

Parameter	Value
R_t	[0,5] (ms)
D_t	[5,10] (ms)
$E_{t_{cpu}}$	[0. 0002,0.5] (ms)
T_t	[0,0.136] (ms)
AS_t	2.9 (ms)
D	64 bit
CR	500 MHz[32]
N_{core}	12
Transfer rate of HDD	0.75 GB/s
Number of tasks	50, 100, 200
Number of CPU and HDD	4, 16, 32, 64

A. Utilization

In the task set utilization, U is the fraction of C to W, where C is the sum of the execution times of CPU and R/W is the times of hard disk drive, and W is the sum of six times including the ready time, the waiting time, the idle time, the context switch time, the execution time of CPU, and the R/W time of HDD, which is given by:

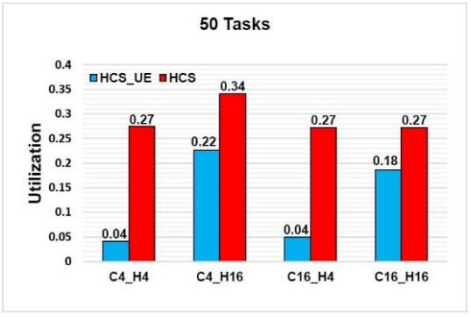
$$U = \frac{\sum_{i=1}^{N} C_i}{\sum_{i=1}^{N} W_i} \quad (21)$$

Due to the reduction of the waiting times of tasks, the task set utilization is increased. Ci_Hj which is used in the following analysis, i and j are the number of CPUs and HDDs, respectively. As shown in Fig. 10, the utilization of HCS algorithm is improved in comparison with HCS_UE algorithm. The percentage of improvements is shown in Table5. The key observation is that improvement of the utilization is most visible when HDD number is minimum. In addition, by considering the same number of HDDs more improvement is achieved by reducing the number of CPUs.

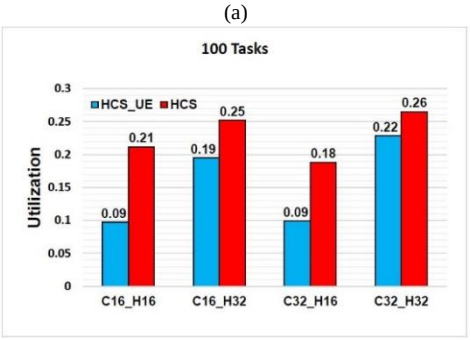
In general, the best improvement is achieved when the HDDs and CPUs have the least number.

Table 5. Percentage improvements of utilization

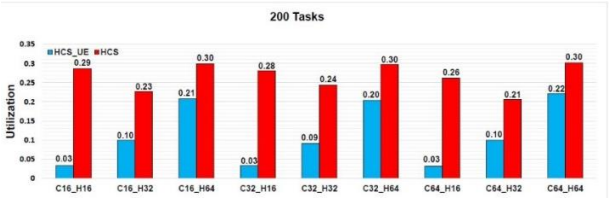
50 tasks		
No. of CPU	No. of Hard	Improvement (%)
4	4	84.98
4	16	33.59
16	4	81.93
16	16	31.44
Total Avg.:		**57.98**
100 tasks		
No. of CPU	No. of Hard	Improvement (%)
16	16	53.99
16	32	22.57
32	16	47.28
32	32	13.88
Total Avg.:		**34.43**
200 tasks		
No. of CPU	No. of Hard	Improvement (%)
16	16	88.30
16	32	56.25
16	64	30.67
32	16	88.48
32	32	62.65
32	64	31.73
64	16	87.87
64	32	51.80
64	64	26.76
Total Avg.:		**58.28**



(a)



(b)



(c)

Fig.10. The utilization of tasks (a) 50 tasks, (b) 100 tasks, (c) 200 tasks

B. Total and average execution times

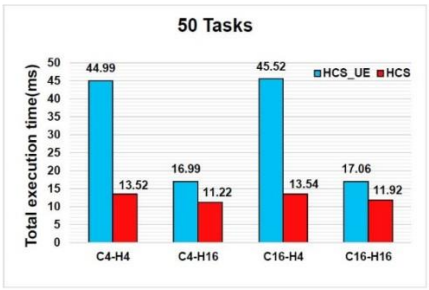
In various numbers of CPUs and HDDs, the total execution time is the time taken to finish the execution of all the tasks. Therefore, the maximum execution time of tasks will be considered as the total execution time. The total execution time improvements in HCS algorithm in comparison with HCS_UE algorithm in 50, 100 and 200 tasks are shown in Fig. 11. The percentage of improvements is shown in Table 6.

The results show that the total execution times are considerably decreased up to about 2x for 50 and 100 tasks when the HDDs have the minimum number. Also, it is observed that the total execution time for the least number of HDD is improved up to about 2x and 3x compared to 32 and 64 HDDs for 200 tasks.

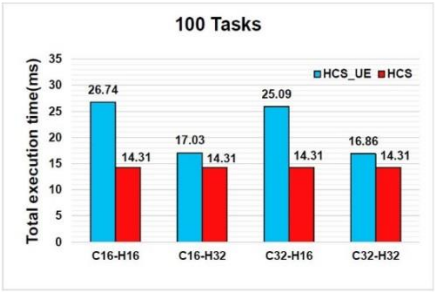
As shown in Fig. 12, the average execution time of tasks in HCS algorithm is improved in comparison with HCS_UE algorithm. The percentage improvements is reported in Table 7.

Table 6. Percentage improvements of total execution time

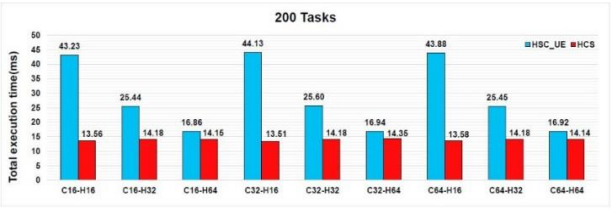
50 tasks		
No. of CPU	No. of Hard	Improvement (%)
4	4	69.94
4	16	34
16	4	70.25
16	16	29.97
Total Avg.:		**51.04**
100 tasks		
No. of CPU	No. of Hard	Improvement (%)
16	16	46.47
16	32	15.97
32	16	44.75
32	32	15.11
Total Avg.:		**30.57**
200 tasks		
No. of CPU	No. of Hard	Improvement (%)
16	16	68.62
16	32	44.24
16	64	16.10
32	16	69.39
32	32	44.60
32	64	15.25
64	16	69.06
64	32	44.26
64	64	16.41
Total Avg.:		**43.10**



(a)



(b)



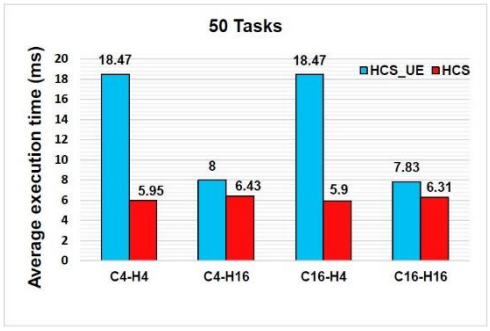
(c)

Fig.11. The total execution time (a) 50 tasks, (b) 100 tasks, (c) 200 tasks

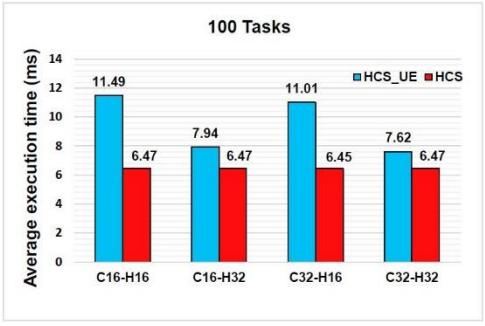
The results show that the average execution times are considerably decreased up to about 3x and 2x for 50 and 100 tasks when the HDDs have the least number. Also, it is observed that the average execution time for the least number of HDD is improved up to about 2x and 3x compared to 32 and 64 HDDs for 200 tasks.

Table 7. Percentage improvements of Avg. execution time

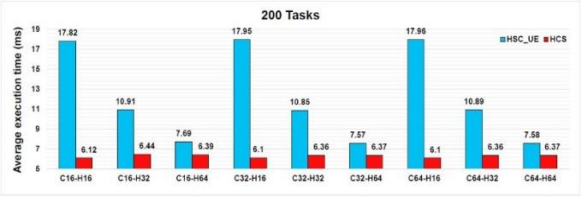
50 tasks		
No. of CPU	No. of Hard	Improvement (%)
4	4	67.80
4	16	19.60
16	4	68.04
16	16	19.45
Total Avg.:		**43.72**
100 tasks		
No. of CPU	No. of Hard	Improvement (%)
16	16	43.65
16	32	18.55
32	16	41.46
32	32	15.03
Total Avg.:		**29.67**
200 tasks		
No. of CPU	No. of Hard	Improvement (%)
16	16	65.61
16	32	41
16	64	16.89
32	16	66.12
32	32	41.40
32	64	15.80
64	16	66.15
64	32	41.53
64	64	15.84
Total Avg.:		**41.15**



(a)



(b)

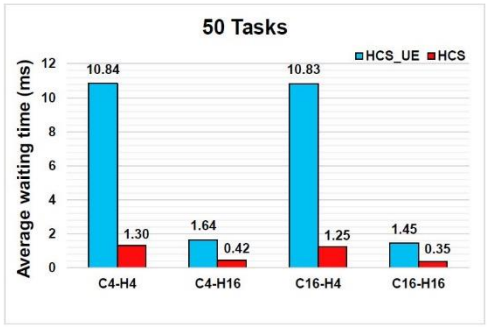


(c)

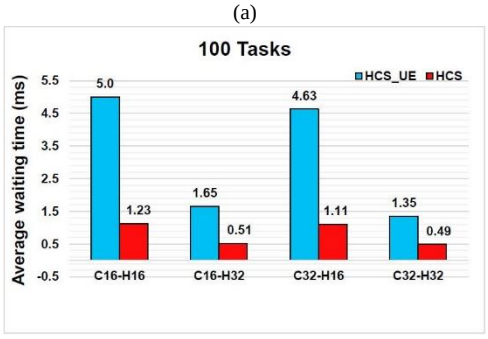
Fig.12. The average execution time(a) 50 tasks, (b) 100 tasks, (c) 200 tasks

C. Average waiting time

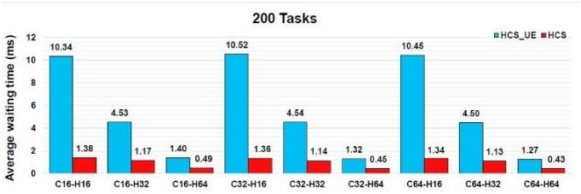
In order to reduce the waiting time of unscheduled tasks, the execution time of scheduled tasks is decreased. As illustrated in Fig. 13, the average waiting time of tasks in HCS algorithm is improved in comparison with HCS_UE algorithm The percentage improvements is reported in Table 8. Generally, the best improvement is achieved when the HDDs have the least number.



(a)



(b)



(c)

Fig.13. The average waiting time (a) 50 tasks, (b) 100 tasks, (c) 200 tasks

D. Number of missed tasks

In order to reduce, as much as possible, the number of missed tasks, we attempt to reduce the waiting time of unscheduled tasks by decreasing the execution time of tasks. In other words, the number of missed tasks is decreased by increasing the frequency of CPUs and/or the RPM level of HDDs. Fig. 14 shows the numbers of missed tasks in HCS_UE and HCS algorithms. The percentage improvements is reported in Table 9.

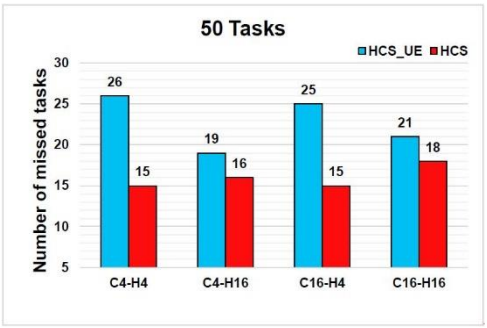
Table 8. Percentage improvements of Avg. waiting time

50 tasks		
No. of CPU	No. of Hard	Improvement (%)
4	4	87.98
4	16	74.62
16	4	88.44
16	16	76.09
Total Avg.:		**81.78**
100 tasks		
No. of CPU	No. of Hard	Improvement (%)
16	16	77.51
16	32	69
32	16	76.10
32	32	63.79
Total Avg.:		**71.6**
200 tasks		
No. of CPU	No. of Hard	Improvement (%)
16	16	86.61
16	32	74.13
16	64	65.28
32	16	87.10
32	32	74.89
32	64	65.77
64	16	87.13
64	32	74.98
64	64	65.88
Total Avg.:		**75.75**

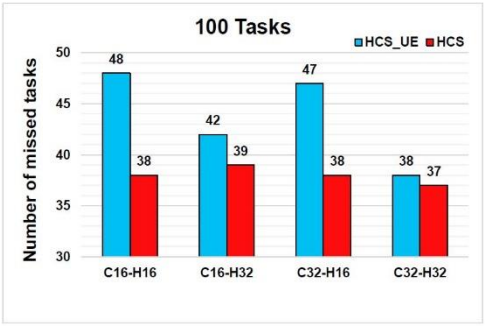
Table 9. Percentage improvements of missed tasks

50 tasks		
No. of CPU	No. of Hard	Improvement (%)
4	4	42.31
4	16	15.79
16	4	40
16	16	14.28
Total Avg.:		**28.1**
100 tasks		
No. of CPU	No. of Hard	Improvement (%)
16	16	20.83
16	32	7.14
32	16	19.15
32	32	2.63
Total Avg.:		**12.44**
200 tasks		
No. of CPU	No. of Hard	Improvement (%)
16	16	41.82
16	32	23.96
16	64	16.67
32	16	40.37
32	32	24.49
32	64	13.25
64	16	41.28
64	32	20
64	64	12.5
Total Avg.:		**26.04**

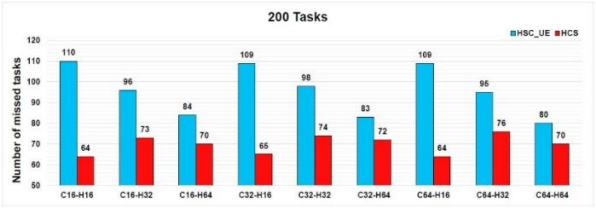
The results show that the number of missed tasks are considerably decreased more than 2x for 50 and 100 tasks when the HDDs have the least number. Also, the result of simulation shows that the number of missed tasks for the least number of HDD is improved less than 2x and more than 2x compared to 32 and 64 HDDs for 200 tasks.



(a)



(b)



(c)

Fig.14. The number of missed tasks (a) 50 tasks, (b) 100 tasks, (c) 200 tasks

V. Conclusion

In this paper a new scheduling algorithm is proposed for real-time periodic tasks called Hard disk drive and CPU Scheduling (HCS), which is applicable in multi-HDD and multi-CPU with multi-core. The tasks are prioritized according to the ready time, execution time of CPU, finishing time of CPU and R/W time of HDD. Furthermore, the execution time of tasks is changed based on the variations of CPU frequency and RPM levels of hard disk drive. HCS is significantly reduced the number of missed tasks, the average and total execution time and average waiting time of tasks for task sets of size ranging from 50 tasks to 200 tasks. Moreover, result of simulation shows that the approach has been successful in obtaining satisfactory result in the improvement of the task set utilization.

The experimental results show that on the average 50.22% improvement in the task set utilization, 41.57% improvement in the total execution time, 38.18% improvement in the average execution time, 75.63% improvement in the average execution time and 22.18 % improvement in the number of missed tasks are achieved in comparison with the Hard disk drive and CPU Scheduling _Unchanged Execution time (HCS_UE) algorithm.

In the future, we plan to modify the proposed algorithms in order to apply it on real environment and also extract actual data on real environments to evaluate the performance of the algorithms by considering the memory effects like memory-wall effects, bandwidth and latency of multi-core systems. Also, we would like to perform the simulations by using more complex and complete simulator like SPICE.

References

[1] Kamga, C.M., 2012. CPU Frequency Emulation Based on DVFS, In: Utility and Cloud Computing (UCC), pp: 367–374. DOI: 10.1109/UCC.2012.34.

[2] Cho, S.J., S.H. Yun and J.W. Jean, 2015. A power saving DVFS algorithm based on Operational Intensity for embedded systems, In: IEICE Electronics Express, vol. 12, no. 3, Jan., pp: 1–7. DOI: http://doi.org/10.1587/elex.12.20141128.

[3] Da-Ren, Ch., Ch. Young-Long and Ch. You-Shyang, 2014. Time and Energy Efficient DVS Scheduling for Real-Time Pinwheel Tasks. In: Journal of Applied Research and Technology, vol. 12, issue. 6, Dec., pp: 1025–1039. DOI: 10.1016/S1665-6423(14)71663-3.

[4] Tang, Z., L. Qi, Z. Cheng, K. Li, S.U. Khan and K. Li, 2015. An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment. In: Journal of Grid Computing, April. DOI: 10.1007/s10723-015-9334-y.

[5] Babaii, N., Rizvandi, J. Taheri and A.Y. Zomaya, 2011. Some observations on optimal frequency selection in DVFS-based energy consumption minimization. In: Journal of Parallel and Distributed Computing, vol. 71, issue 8, pp: 1154–1164. DOI: 10.1016/j.jpdc.2011.01.004.

[6] Laszewski, G.V., L. Wang, A.J. Younge and X. He, 2009. Power-Aware Scheduling of Virtual Machines in DVFS-enabled Clusters. In: Cluster Computing and Workshops, pp: 1–10. DOI: 10.1109/CLUSTR.2009.5289182.

[7] Tchamgoue, G.M., J. Seo, K.H. Kim and Y.K. Jun, 2015. Compositional Power-Aware Real-Time Scheduling with Discrete Frequency Levels. In: Journal of Systems Architecture. DOI:10.1016/j.sysarc.2015.05.003.

[8] Wu, J., 2015. Energy-Efficient Scheduling of Real-Time Tasks with Shared Resources. In: Future Generation Computer Systems, May. DOI: 10.1016/j.future.2015.05.012.

[9] Zhu, X., C. He, K. Li and X. Qin, 2012. Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters. In: Journal of Parallel and Distributed Computing, pp: 751–763. DOI: 10.1016/j.jpdc.2012.03.005.

[10] Pedram, M. and K. Choi, 2005. Dynamic Voltage and Frequency Scaling for Energy-Efficient System Design. In: the Association for Computing Machinery. ISBN: 0-542-20387-1.

[11] Liu, J. and J. Guo, 2015. Energy efficient scheduling of real-time tasks on multi-core processors with voltage

islands. In: Future Generation Computer Systems, Jun. DOI: 10.1016/j.future.2015.06.003.

[12] Tavares, E., P. Maciel, B. Silva and M.N. Oliveira, 2008. Hard real-time tasks' scheduling considering voltage scaling, precedence and exclusion relations. In: Information Processing Letters, vol. 108, issue. 2, Sept., pp: 50–59. DOI: 10.1016/j.ipl.2008.03.020.

[13] Swaminathan, V., K. Chakrabarty and S.S. Iyengar, 2001. Dynamic I/O Power Management for Hard Real-time Systems. In: Hardware/Software Codesign, pp: 237–242. DOI: 10.1145/371636.371742.

[14] Zhang, Y. and R. Guo, 2014. Power-aware fixed priority scheduling for sporadic tasks in hard real-time systems. In: Journal of Systems and Software, vol. 90, pp: 128–137. DOI:10.1016/j.jss.2013.12.032.

[15] Awan, M.A. and S.M. Petters, 2015. Intra-task device scheduling for real-time embedded systems. In: Journal of Systems Architecture, vol. 61, issue. 8, Sep., pp: 321–340. DOI: 10.1016/j.sysarc.2015.07.001.

[16] Fan, M., Q. Han, S. Liu, S. Ren, G. Quan and S.Ren, 2015. Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship. In: Journal of Systems and Software, vol. 99, Jan., pp: 85–96. DOI:10.1016/j.jss.2014.09.010.

[17] Kong, F., Y. Wang, Q. Deng and W. Yi, 2010. Minimizing Multi-Resource Energy for Real-Time Systems with Discrete Operation Modes. In: Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on, pp: 113–122. DOI: 10.1109/ECRTS.2010.18.

[18] Zhang, Y-W. and R-f. Guo, 2013. Power-aware scheduling algorithms for sporadic tasks in real-time systems. In: Journal of Systems and Software, vol. 86, issue. 10, Oct., pp: 2611–2619. DOI:10.1016/j.jss.2013.04.075.

[19] Jianjun, L., LihChyun, S. and Jian-Jia, C., 2013 Energy-Efficient Scheduling in Non preemptive Systems With Real-Time Constraints. In: IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 43, issue. 2, March., pp: 332-344. DOI: 10.1109/TSMCA.2012.2199305.

[20] Mihai, P. and Tulika, M., 2014. Task Scheduling on Adaptive Multi-Core. In: IEEE Transactions on Computers, vol. 63, issue. 10, Oct., pp: 2590-2603. DOI: 10.1109/TC.2013.115.

[21] Inoue, T., A. Aikebaier, T. Enokido and M. Takizawa, 2011. A Power Consumption Model of a Storage Server. In: Network-Based Information Systems (NBiS), pp: 382–387. DOI 10.1109/NBiS.2011.64.

[22] Hylick, A., R. Sohan, A. Rice and B. Jones, 2008. An Analysis of Hard Drive Energy Consumption. In: Modeling, Analysis and Simulation of Computers and Telecommunication Systems, pp: 1–10. DOI: 10.1109/MASCOT.2008.4770567.

[23] Mountrouidou, X., Riska, A. and Smirni, E. 2011. Saving power without compromising disk drive reliability. In: Green Computing Conference and Workshops (IGCC), pp: 1–6. DOI bookmark: http://doi.ieeecomputersociety.org/10.1109/IGCC.2011.6008570

[24] Yun, H., A., Waqar and S., Gondi, 2016. BWLOCK: A Dynamic Memory Access Control Framework for Soft Real-Time Applications on Multicore Platforms. In: IEEE Transactions on Computers, vol. 66, issue. 7, Dec., pp: 1247-1252. DOI: 10.1109/TC.2016.2640961.

[25] Kiani, V., Z. Mohseni and A.M. Rahmani, 2015. Real Time Scheduling for CPU and Hard Disk Requirements-Based Periodic Task with the Aim of Minimizing Energy Consumption. In: International Journal of Information Technology and Computer Science (IJITCS). DOI: 10.5815/ijitcs.2015.10.07.

[26] Jacob,B. and Wang, D. 2007 Memory systems, Cache, DRAM, Disk eBook ISBN: 978-0-12-379751-3, Release date: Sep. 2007, http://store.elsevier.com/product.jsp?isbn=9780123797513. eBook ISBN: 9780080553849.

[27] Torres, G., 2007. How The Memory Cache Works. In: Hardware secrets, Sep. http://www.hardwaresecrets.com/how-the-cache-memory-works/.

[28] Anjum, M.D.M. and H., Wang, 2016. Dynamic scheduling and analysis of real time systems with multiprocessors. In: Digital Communications and Networks, vol. 2, issue. 3, Aug., pp: 130-138. DOI: https://doi.org/10.1016/j.dcan.2016.06.004.

[29] Konar, D., S. Bhattacharyya, K. Sharma and S. Sharma, 2017. An improved Hybrid Quantum-Inspired Genetic Algorithm (HQIGA) for scheduling of real-time task in multiprocessor system. In: Applied Soft Computing, vol. 53, pp: 296-307. DOI: https://doi.org/10.1016/j.asoc.2016.12.051.

[30] Kumar Samal, A., R. Mall and C. Tripathy, 2014. Fault tolerant scheduling of hard real-time tasks on multiprocessor system using a hybrid genetic algorithm. In: Swarm and Evolutionary Computation, vol.14, pp: 92-105. DOI: https://doi.org/10.1016/j.swevo.2013.10.002.

[31] Intel® Xeon® Processor E5-2670 v3. http://ark.intel.com/products/81709/Intel-Xeon Processor-E5-2670-v3-30M-Cache-2_30-GHz, (Accessed on 09/25/2015).

[32] Intel® Xeon® Processor E5 v2 Product Family, Datasheet, vol. 2, March. 2014. http://www.intel.com/content/www/us/en/processors/xeon/xeon-e5-v2-datasheet-vol-2.html.

[33] HP SAS Hard Drives, c04311358 – DA – 12244 North America – Version 50 – June 1, 2015. http://www8.hp.com/us/en/products/oas/product-detail.html?oid=5163353.

[34] Grochowski, E. and R.D. Halem, 2003. Technological impact of magnetic hard disk drives on storage systems. In: IBM SYSTEMSJOURNAL, vol. 42, no. 2. DOI: 10.1147/sj.422.0338.

Authors' Profiles

Zeynab Mohseni received her Associates Degree in Software Computer from Sabzevar Technical and Vocational Collage, Khorasan, Iran in 2007, the B.S. in Computer Engineering from khorasan Institute of Higher Education, Mashhad, Khorasan, Iran in 2009 and the M.S. in Computer Engineering from Science and Research Branch, Islamic Azad University, Tehran, Iran in 2015. Her research interests are in the areas of Fault-tolerance and Reliability, embedded and real-time systems, scheduling algorithms and Network on Chip.

Vahdaneh Kiani received her B.S. in Computer Engineering from South Tehran Branch, Islamic Azad University, Tehran, Iran in 2011 and the M.S. in Computer Engineering form Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran in 2015. Her research interests are in the areas of embedded and real-time systems, scheduling algorithms, digital circuit design, memory systems and fault tolerance in Microprocessors and Network on Chip.

Amir Masoud Rahmani received his B.S. in Computer Engineering from Amir Kabir University, Tehran, in 1996, the M.S. in Computer Engineering from Sharif University of technology, Tehran, in 1998 and the Ph.D. degree in Computer Engineering from IAU University, Tehran, in 2005. He is an assistant professor in the Department of Computer and Mechatronics Engineering at the IAU University. He is the author/co-author of more than 80 publications in technical journals and conferences. He served on the program committees of several national and international conferences. His research interests are in the areas of distributed systems, ad hoc and sensor wireless networks, scheduling algorithms and evolutionary computing.

How to cite this paper: Zeynab Mohseni, Vahdaneh Kiani, Amir Masoud Rahmani, "A Task Scheduling Model for Multi-CPU and Multi-Hard Disk Drive in Soft Real-time Systems", International Journal of Information Technology and Computer Science(IJITCS), Vol.11, No.1, pp.1-13, 2019. DOI: 10.5815/ijitcs.2019.01.01

