EFFICIENT SCHEDULING OF REAL-TIME PARALLEL APPLICATIONS ON CLUSTERS

Abeer Hamdy[1], Reda Ammar[2] and Ahmed E. Youssef[3]

[1]Dept. of Computers and Systems, Electronics Research Institute, and
Faculty of Informatics and Computer Science, British University in Egypt
abeer@eri.sci.eg
[2]Dept. of computer science & Engineering, University of Connecticut, CT 06268, USA
reda@engr.uconn.edu
[3]Faculty of Engineering, Helwan University, Cairo, Egypt, and
Dept. of Information Systems, King Saud University, Riyadh, KSA
ahyoussef@ksu.edu.sa

ABSTRACT

This paper proposes a novel approach to schedule real-time applications represented by Fork-Join (FJ) task graphs on a cluster platform. The novelty of our approach lies in its ability to efficiently utilize the power of the cluster's computational resources to improve the system throughput. Our approach integrates two heuristic scheduling algorithms: the first algorithm (partition algorithm) works on the cluster level to search for the best allocation scheme for the application's tasks on the cluster's processors. This search is guided by an objective function that aims to optimize the utilization of cluster's resources. The second algorithm (local scheduler) works on the individual processor level to efficiently utilize the processing power of each processor. A set of simulation experiments have been conducted to evaluate the performance of our scheduling approach on both homogeneous and heterogeneous clusters. The results show that our approach improves the acceptance rate of the parallel applications on the cluster compared to traditional approach.

KEYWORDS

Processor utilization, workload allocation, scheduling, parallel task graph, Cluster computing.

1. INTRODUCTION

Cluster Computing is one of the important platforms for solving large scale problems. A computer cluster is a group of loosely coupled computers connected through fast local area network and work together closely so that in many respects it can be viewed as a single computer. Cluster computing is currently used in several high performance applications that exhibit real-time characteristics (i.e. should respond within certain pre-set deadlines). A middleware called scheduler is an urgent demand to manage the allocation of the cluster's resources to the admitted applications. Scheduler plays a critical role in the efficient utilization of the cluster's computing resources, such that the cluster can accommodate as many applications as possible (i.e. maximizing the system throughput). The problem of scheduling tasks over a multi-processor or distributed platform is known to be NP-complete in general [16]. Efficient algorithms that give the optimal schedule can only be obtained when some restrictions are imposed on the models representing the application and the multiprocessor or distributed system. There are only few known deterministic polynomial-time scheduling algorithms [10,17,18]; therefore, solving the

DOI : 10.5121/ijcsit.2012.4516

general scheduling problem in polynomial-time requires the use of heuristic algorithms that provide near-optimal solutions.

Scheduling real time (RT) applications on these platforms adds another challenge to this problem, as it is required to satisfy the timing constraints of the RT-applications. A vast body of research has been conducted to investigate the problem of scheduling RT-applications across the computing resources of multiprocessor/multicore platforms [1,2,4,7,9,19] or distributed platforms [13,15,21,22,23]. Some proposed partition algorithms to schedule independent RT-tasks [1,15] while others proposed partition algorithms to schedule task graph applications [3,6,7,8,11,12,13,20]. Each proposed algorithm tries to maximize certain performance criteria such as saving energy over platforms with dynamic voltage scaling (DVS) [1, 15, 19, 23]. Other algorithms are concerned with achieving effective fault-tolerant in RT-systems [21,22]. A third category of algorithms aim to minimize the application makespan [12, 13]. However, a very few number of them tried to improve the processing power utilization. Moreover, most of the researchers restrict their attention to developing the partition algorithm on the cluster level and employ any previous simple algorithm, at the processor level, for processing power reservations. Some employs algorithms that assigns each task a fixed percentage CPU reservation and don't efficiently utilize the processor [13]. Others use the earliest-deadline-first scheduler [5]; which doesn't include a mechanism that prevents one task from causing others to miss their deadlines.

In this paper, we propose a new approach for scheduling RT-applications represented by Fork-Join task graph on a cluster. Our approach has the advantage of efficient utilization of the power of the cluster's computing resources in order to provide deadline guarantees to the RT-applications and improve their acceptance rate (acceptance rate is defined as the number of schedulable applications to the total number of submitted applications). Our scheduling approach consists of two heuristic algorithms namely: partition algorithm and local scheduler. Both of them cooperate for efficient utilization of the power of the processors. On the cluster level, the partition algorithm partitions the application and searches for the best allocation scheme of the tasks of the application on the cluster's processors such that the acceptance rate of the applications is maximized. On the processor level, each individual processor has a local scheduler that manages the execution of the assigned tasks to that processor in a way that efficiently utilizes the processor and provides guarantees to deadlines requirements. This paper discusses the proposed partition algorithm in detail and its integration with the efficient local scheduler proposed by us in [14].

The rest of the paper is organized as follows: Section 2 presents models for the scheduling environment (i.e., the cluster and the application). Our scheduling approach is discussed in Section 3. Section 4 presents an illustration example for the proposed approach. Section 5 evaluates the performance of our scheduling approach with respect to the traditional approach through a set of simulation experiments. Finally, section 6 concludes the paper and suggests extension to our approach as a future work.

2. MODELLING APPROACH

This section describes briefly each of the cluster and application models that are used in our approach.

2.1. Cluster Model

The cluster system is modelled by an undirected graph, $G = (\boldsymbol{N},\boldsymbol{E})$ where:

$\boldsymbol{N} = \{N_1, N_2, \dots, N_k\}$ is a set of nodes (computing resources) constituting the cluster.
$\boldsymbol{E}$ is a set of edges (communication links) between the nodes.

The cluster is functioning under the following assumptions:

- Cluster network graph G is complete (i.e., the nodes are fully connected), as shown in figure 1a.
- Cluster processors may be homogeneous (have the same speed) or heterogeneous (have different speeds).
- Each node has a local scheduler maintains a data structure called reservation table that has information about the executing tasks on this processor which includes their starting times, deadlines and processing power percentages. The reservation tables of all the nodes are publicly available and dynamically changing.
- The partition algorithm uses the reservation tables to get the available workload on each processor WL_{ova-p}.
- Local scheduler guarantees that it will provide the computing resources as announced and will release them as soon as the task is completed.
- The system is fault free.

2.2. Application Model

A set of N applications, $\boldsymbol{A} = \{A_1, A_2, \ldots, A_N\}$, compete for the cluster resources. Each submitted application $A_i = \{T_{i,1}, T_{i,2}, \ldots, T_{i,ni}\}$ is represented by a set of n_i parallel real-time tasks in the form of a Fork-Join task graph as shown in figure 1b. We made the following assumptions:

- There is no communication between the parallel tasks during execution.
- The whole set of application tasks are known at submission time. The reason for such an assumption is to schedule all tasks of an application at once.
- Applications randomly arrive.
- The tasks belong to one application A_i have the same start time S_i and the same deadline D_i. Each task $T_{i,j}$ of an application A_i is characterized by its estimated execution time $\tau_{i,j}$.

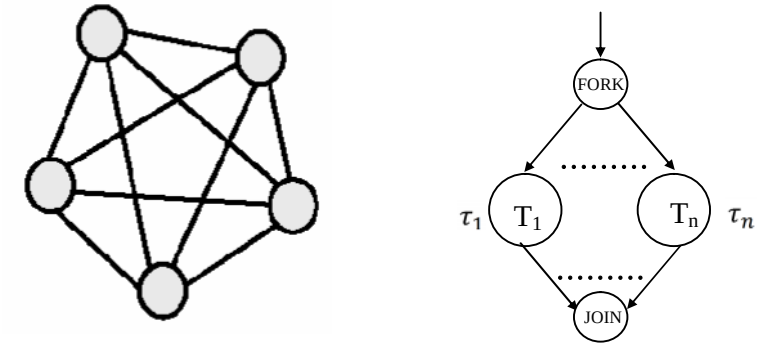

Figure 1. (a) Cluster model. (b) Fork-Join application model

3. PROPOSED SCHEDULING APPROACH

The main goal of our scheduling approach is to improve acceptance rate of F-J task graph applications on the cluster. This goal is achieved by the efficient utilization of the power of the cluster's computing resources. Our scheduling approach consists of two schedulers, namely: the partition algorithm and the local scheduler. The following subsections describe the role of each scheduler in allocating the tasks on the cluster such that the required goal is achieved.

3.1. Local Scheduler

In our approach, each processor in the cluster has its own local scheduler, which is responsible for executing the tasks on this processor. The local scheduler adopted in this paper is discussed in detail in [14]. It efficiently utilizes the processing power of the cluster's processors to accommodate as many tasks as possible. It relies basically on two ideas:

- **First,** when allocating a task on a processor a variable processing power is used to allocate the task's workload when enough constant processing power cannot be guaranteed.

- **Second,** between the arrivals of two applications or when a task finishes and its processing power is released; the available processing power is exploited and redistributed among the executing tasks to reduce their execution time and to give the chance for the new admitted applications to be accepted.

Within this scheduler, the processor alternates between two modes during scheduling, the reservation mode and the execution mode:

- **Reservation Mode (RM):** is activated at the arrival of a new application. In this mode, new reduced processing powers are assigned to the executing tasks such that they guarantee no deadlines violation. Consequently, more available workload is provided to the new arrived applications. This gives a higher chance for these applications to be accepted.

- **Execution Mode (EM):** is activated between two successive arrivals of the applications or at the departure of a task. In this mode, available workload is distributed over the executing tasks proportional to their current workloads.

3.2. Partition Algorithm

Partition algorithm is responsible for partitioning the application and allocating its tasks across the available processors. It is a heuristic algorithm that searches for a scheduling scheme on the cluster that maximizes acceptance rate of the applications. This search is guided by an objective function that integrates two performance criteria namely: processing power fragmentation and context switching. Optimizing these criteria results in assigning the processors to the tasks in a way that leads to better utilization of the processors. Consequently, the acceptance rate of the applications is improved. The optimal scheduling scheme is the one with the maximum value of the objective function.

The Objective Function

The objective function is given by the following equation:

$$\omega_{i,j,p} = \frac{1}{f_{i,j,p} * c_{i,j,p}}$$

Where:

$f_{i,j,p}$: Fragmentation when task $T_{i,j}$ is allocated on processor P_p.

$C_{i,j,p}$: Context switching on processor P_p in the window of $T_{i,j}$.

- **Fragmentation Term ($f_{i,j,p}$):** the objective of this term is to minimize processing power fragmentation by attempting allocate tasks on the processor with the minimum enough available workload leaving other processors that have more available workload to allocate new applications. This increases acceptance rate of the newly arrived applications. This term is expressed by the following equation:

$$f_{i,j,p} = \frac{WL_{ava\_p} - WL_{i,j}}{D_i}$$

Where:

D_i : Deadline of the application A_i

WL_{ava_p} : Available workload on processor P_p .

$WL_{i,j}$: Required workload of $T_{i,j}$.

- **Context Switching Term ($C_{i,j,p}$):** the objective of this term is to minimize the context switching time cost. This is the time used by the processor to switch from one task to another during execution and is added to the execution time of each task allocated on the processor. Adding this time to the execution time of a task will be compensated by additional processing power for the task to finish at its desired deadline which reduces acceptance rate of the tasks on the processors. It can be measured as the number of times a processor P_p switches from the execution of one task to another task within the deadline of $T_{i,j}$.

As can be noticed there is a conflict between minimizing processing power fragmentation and minimizing context switching. While the former tends to allocate the task on a busy processor the latter tends to allocate it on an empty processor. In our approach, we select the candidate processor based on the resultant of merging these objectives together in one function, which is the objective function $\omega_{i,j,p}$.

PARTITION ALGORITHM STEPS

When an application is submitted to the cluster, the partition algorithm proceeds as follows:

- The partition algorithm stimulates the local schedulers to convert the processors from execution mode (EM) to reservation mode (RM).
- Then, the partition algorithm checks out the initial acceptance of the application on the cluster. The application is accepted initially if the summation of the available workloads on the cluster's processors during the application's deadline is greater than or equal to the summation of the required workloads by the tasks of the application, (i.e. $\sum_{p=1}^{k} WL_{ava-p} \geq \sum_{j=1}^{ni} WL_{i,j}$). If an application fails the initial acceptance test, it is rejected immediately.
- If the application is initially accepted, its tasks are sorted in descending order according to their required workload in order to ensure that the task demanding the largest workload will be allocated the first (has the highest priority), then the next demanding task and so on till all the tasks are allocated. We set these priorities because all the application's tasks

have the same start time and deadline, and allocating one task on a processor reduces its available workload and consequently affects the acceptance of other tasks.

- For the highest priority task $T_{i,j}$, the set of processors on which $T_{i,j}$ can be accommodated (i.e., available processors set $\varphi_{i,j}$) is determined. A task $T_{i,j}$ can be accommodated on a processor P_p if $WL_{ava-p} \geq WL_{i,j}$.
- If $\varphi_{i,j}$(p is an empty set, the application is rejected, otherwise, the values of the objective function are computed for the task $T_{i,j}$ when allocated on each processor in its available processors set (i.e. $\omega_{i,j,p}$ $\forall p \in \varphi_{i,j}$). The task $T_{i,j}$ is allocated on the processor P_p with the maximum value of the objective function ($\max \omega_{i,j,p}$).
- Then the algorithm proceeds with the next highest priority task until all the tasks of the application $T_{i,j}$, $j = \{1..n_i\}$ are scheduled or an empty processor set $\varphi_{i,j}$ is encountered and the application is rejected.
- After that, the partition algorithm stimulates the local schedulers such that each of them distributes the processing power of his processor over the allocated tasks; then switches its processor from RM back to EM.

The steps of the algorithm are summarized in figure2. The example in the following section further illustrates the idea of our partition algorithm.

The computational complexity of this algorithm is $O(m*n)$ where n is the number of tasks in the application; m is the number of processors in the cluster.

Algorithm: Partition algorithm
Input: F-J task graph application A_i that includes n_i tasks { $T_{i,1}$, $T_{i,2}$, ... , T_{i,n_i}}
Output: Scheduling scheme on the cluster

Begin

1. Stimulates local schedulers to switch from EM to RM
2. If $\sum_{p=1}^{k} WL_{ava-p} \geq \sum_{j=1}^{ni} WL_{i,j}$ then executes steps from 3 to 8.
3. Sort $T_{i,j}$ in descending order based on $WL_{i,j}$, $\forall j = \{1..n_i\}$
4. For each $T_{i,j}$
5. $\varphi_{i,j} = \text{empty}$
6. For each processor P_p
7. if: $WL_{ava-p} \geq WL_{i,j}$ then add P_p to $\varphi_{i,j}$
8. If ($\varphi_{i,j}$ = empty) break; else
9. Compute the objective function $\omega_{i,j,p}$ of $T_{i,j}$ on each available processor P_p (i.e. $\omega_{i,j,p}$ $\forall P_p \in \varphi_{i,j}$)
10. Allocate $T_{i,j}$ on the processor P_{p-max} with the maximum $\omega_{i,j,p}$
11. Reduce the available work load of processor P_{p-max} by $WL_{i,j}$ (i.e. $WL_{ava-P_{p-max}} = WL_{ava-P_{p-max}} - WL_{i,j}$)

End

Figure 2. The partition Algorithm

4. ILLUSTRATION EXAMPLE

Suppose a F-J application consists of 4 parallel tasks is arrived to a cluster of eight nodes. The proposed partition algorithm stimulates the local schedulers to switch from EM to RM, and then calculates the available workload on each processor. Table1 shows the application and table 2 shows the available workloads on the cluster processors during the application's deadline (these values are selected for illustration purpose only).

Table 1. Tasks of F-J application.

	S	τ_i	D
T_1	100	140	200
T_2	100	100	200
T_3	100	60	200
T_4	100	20	200

Table 2. Available workload on each processor at the arrival of the application.

Processor	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$WL_{ava\text{-}p}$	110	90	150	200	350	80	190	300

The required workload for each task is calculated and initial acceptance test is performed. Table 3 shows the required workloads of the tasks.

Table 3. Required workloads of the tasks.

	τ_i	D	PP_i	WL_i
T_1	140	200	0.7	140
T_2	100	200	0.5	100
T_3	60	200	0.3	60
T_4	20	200	0.1	15

As $\sum_{i=1}^{4} WL_i = (140 + 100 + 60 + 15) = 31 <$

$$\sum_{p=1}^{8} WL_{ava-p} = (110 + 90 + 150 + 200 + 350 + 80 + 190 + 300)$$

The application is accepted initially and the tasks are sorted in descending order based on their workload requirements as shown by table 3.

The available processor set φ_1 for task T_1 (task with greatest workload requirements) is determined as shown in table 4.

Table 4. Available processor set for task T_1

Processor	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$WL_{ava\text{-}p}$	110	90	150	200	350	80	190	300
φ_1			●	●	●		●	●

The values of the objective function on the processors belong to φ_1 are computed (table 5). The context switching values of the processors shown in table 5 are selected for illustration only. Table 5 shows that assigning task T_1 to processor P_3 maximizes the objective function. So T_1 is allocated on P_3

Table 5. Values of the obj. fn. $\omega_{1,p}$ when T_1 is allocated on the processors $\in \varphi_1$.

Processor	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$WL_{ava\text{-}p}$	110	90	150	200	350	80	190	300
Context switching	7	9	5	10	8	12	6	16
φ_1			●	●	●		●	●
$\omega_{1,p}$			800	66.67	23.81		133.33	23.58

In Table 6, T_1 is allocated to P_3 and the available workload on P_3 is updated by subtracting the required workload of T_3 from its current available workload. Also, the context switching of P_3[1] is updated.

Table 6. Context switching and available workloads on processors after allocating T_1 on P_3.

Processor	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$WL_{ava\text{-}p}$	110	90	10	200	350	80	190	300
Context Switching	7	9	8	10	8	12	6	16
Schedule Scheme			T_1					

In the tables from 7 to 12, we repeat the same procedure for all other tasks in the application. As can be seen in Table 12 the application is given a schedule that allocates T_1 on P_3, T_2 on P_1, and T_3 and T_4 on P_6.

Table 7. Available workload on processors at the arrival of T_2 and the values of obj. fn.

Processor	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$WL_{ava\text{-}p}$	110	90	10	200	350	80	190	300
Context Switching	7	9	8	10	8	12	6	16
Schedule Scheme			T1					
φ_2	●			●	●		●	●
$\omega_{2,p}$	571.42			40	20		74.07	12.5

Table 8. Available workload on processors after allocating T_2 on P_1

Processor	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$WL_{ava\text{-}p}$	10	90	10	200	350	80	190	300
Context Switching	12	9	8	10	8	12	6	16
Schedule Scheme	T_2		T_1					

[1] This update is done for illustration purpose only and does not reflect a specific case study.

Table 9. Available workload on processors at the arrival of T_3 and the values of obj. fn.

Processor	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$WL_{ava\text{-}p}$	10	90	10	200	350	80	190	300
Context Switching	12	9	8	10	8	12	6	16
Schedule Scheme	T_2		T_1					
φ_3		●		●	●	●	●	●
$\omega_{3,p}$		148.15		28.57	17.24	166.67	51.28	10.42

Table 10. Available workload on processors after allocating T_3 on P_6

Processor	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$WL_{ava\text{-}p}$	10	90	10	200	350	20	190	300
Context Switching	12	9	8	10	8	15	6	16
Schedule Scheme	T_2		T_1			T_3		

Table 11. Available workload on processors at the arrival of T_4 and the values of obj. fn.

Processor	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$WL_{ava\text{-}p}$	10	90	10	200	350	20	190	300
Context Switching	12	9	8	10	8	15	6	16
Schedule Scheme	T2		T1			T3		
φ_4		●		●	●	●	●	●
$\omega_{4,p}$		59.26		21.62	14.93	533.33	38.1	8.77

Table 12. Available workload on processors after allocating T_4 on P_6.

Processor	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$WL_{ava\text{-}p}$	10	90	10	200	350	5	190	300
Context Switching	12	9	8	10	8	18	6	16
Schedule Scheme	T2		T1			T3, T4		

5. SIMULATION EXPERIMENTS

We have conducted a set of simulation experiments to evaluate the performance of our proposed scheduler. Two local schedulers have been experimented along with the proposed partition algorithm: 1) Our proposed local scheduler (WLEM) [14], 2) A previous scheduler that is developed by Microsoft [13] (Traditional). The two local schedulers have been experimented to show that an inefficient local scheduler may degrade the performance of the partition algorithm. The following two subsections discuss the simulation experiments setup and the results.

5.1. Experiments Setup

The simulation experiments have been conducted on both homogeneous and heterogeneous clusters. The performance metric used in these experiments is the acceptance rate of the applications on the cluster. We measure acceptance rate at different values of mean inter-arrival time of the applications (1/λ) and at different values of mean execution time (1/μ) (λ and μ are the arrival and the departure rate respectively).

In each experiment, sets of applications are generated using the following settings:

1. Each set contains 10000 randomly generated applications. Each application is represented by a F-J task graph (i.e. application tasks have the same start time, finish time, and deadline)
2. Different sets of application have different values of (1/λ). However, they have the same value of (1/μ).
3. In a set of application, an exponential probability distribution with mean (1/λ) is used to generate random values for the inter-arrival time of the applications.
4. In a set of applications, a uniform probability distribution of mean (1/μ) is used to generate random values for task execution times.
5. A uniform probability distribution is used to generate a unified deadline of the tasks such that it is not less than the maximum execution time of the tasks.
6. A uniform probability distribution, *U(2,12)*, is used to generate number of tasks in the application.
7. A uniform probability distribution, *U(1.5,5)*, is used to generate values for $\alpha = 1/pp$.

5.2. Experimental Results

Figures 3, 4, and 5 show the acceptance rate vs. mean inter-arrival time for each experiment in case of homogeneous clusters. In all experiments, results show that the performance of our proposed partition algorithm when adopting our proposed local scheduler is superior to the performance when adopting traditional local scheduler approach. Results also show that in both cases more applications are rejected when applications arrive faster than the cluster can handle (i.e. when the cluster is overloaded). However, our algorithm is still superior to the traditional one. By contrast, both approaches perform competitively well for large values of 1/λ (slow arrival). This is because the applications arrive apart from each other; hence, the scheduling process becomes easier.

Figure 6 shows the percentage improvement in the acceptance rate achieved by our approach over that of the traditional approach at different values of 1/μ. As can be seen on the graph, the improvement diminished as 1/λ increases. This is due to the fact that both approaches perform well for large values of mean inter-arrival time (slow arrival). The graph also shows that we achieve higher amount of improvement for small values of 1/λ (fast arrivals) and large value of 1/μ (slow departure). Hence, we conclude that our proposed approach has a major improvement when long tasks arrive at high arrival rate. In this case the scheduling process becomes difficult and the traditional approach has relatively low acceptance rate.

Figures 7, 8, 9 and 10 show the results in case of heterogeneous clusters. Conclusions similar to the above conclusions can be derived from these graphs.

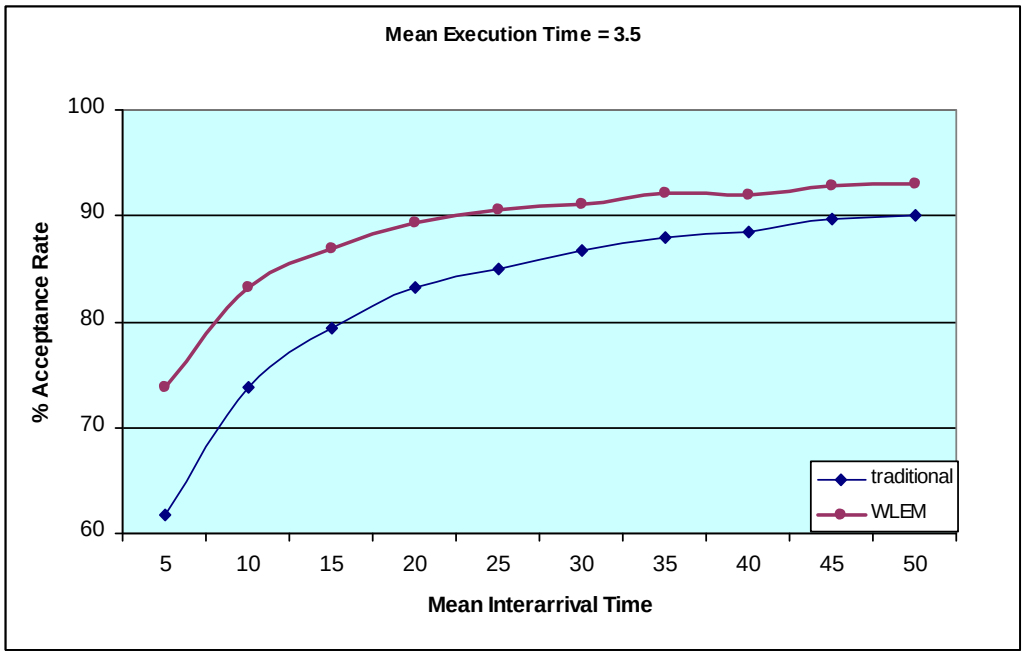

Fig. 3: Acceptance Rate at $1/\mu = 3.5$ (homogenous cluster)

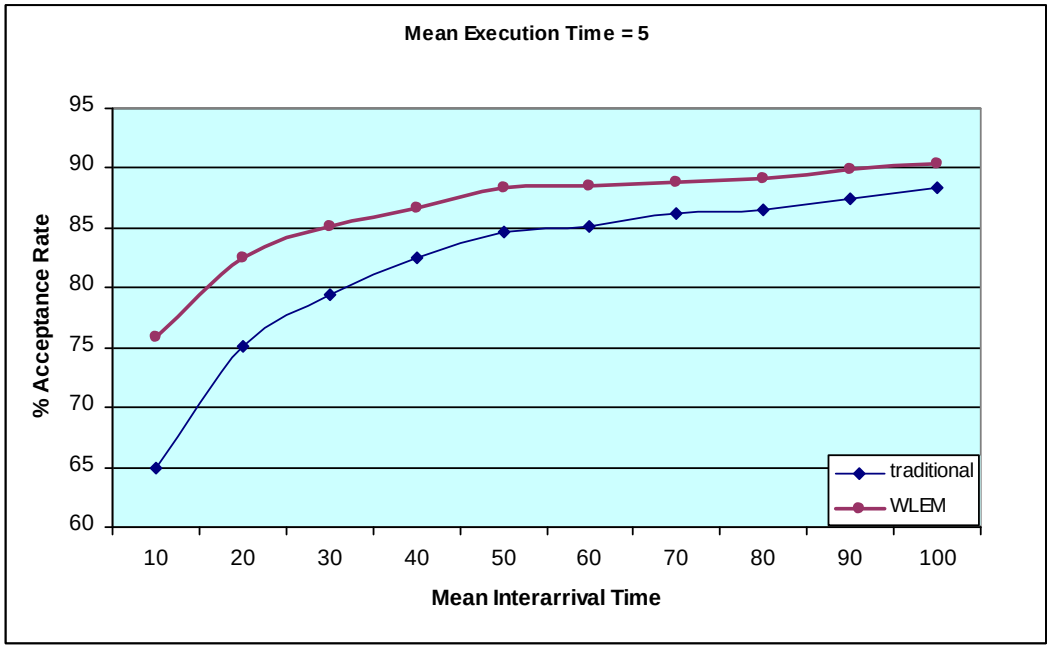

Fig. 4: Acceptance Rate at $1/\mu = 5$ (homogenous cluster)

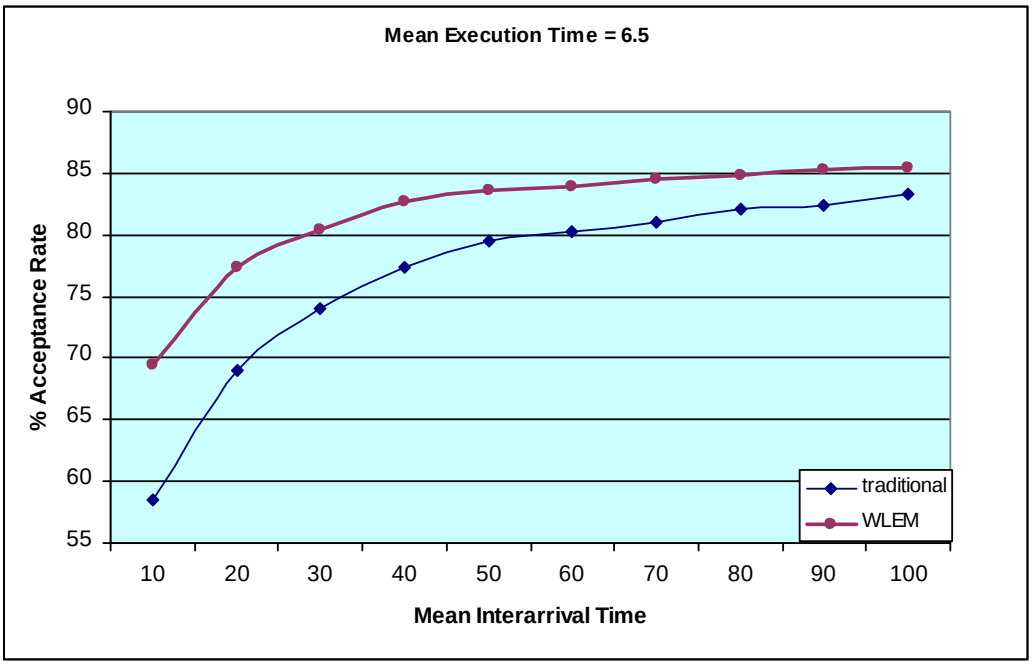

Fig. 5: Acceptance Rate at 1/μ = 6.5 (homogenous cluster)

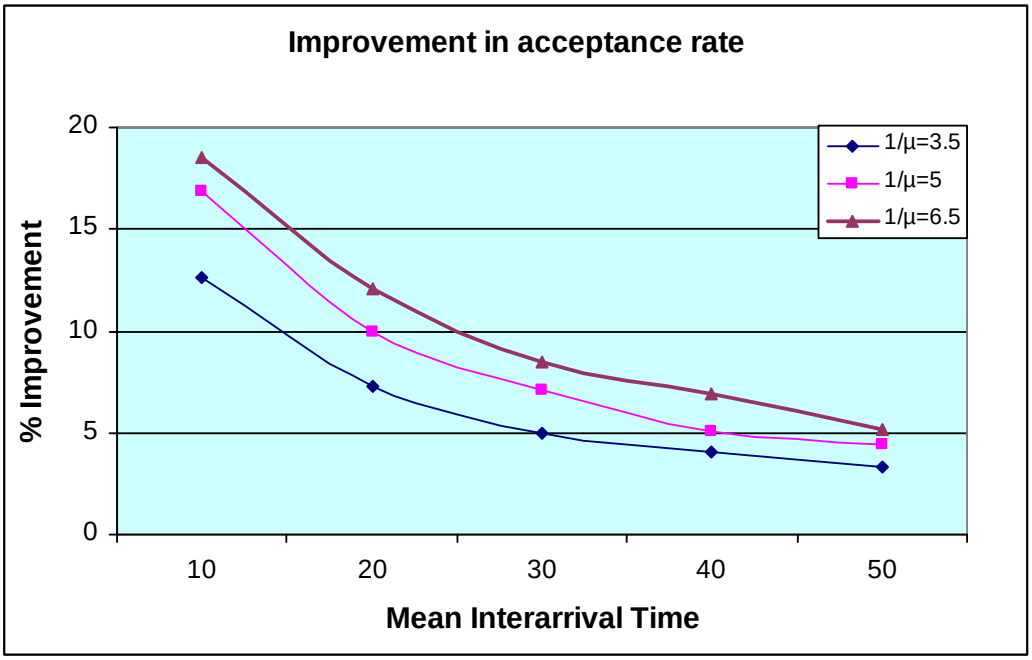

Fig. 6: Improvement at different values of 1/μ (homogenous cluster)

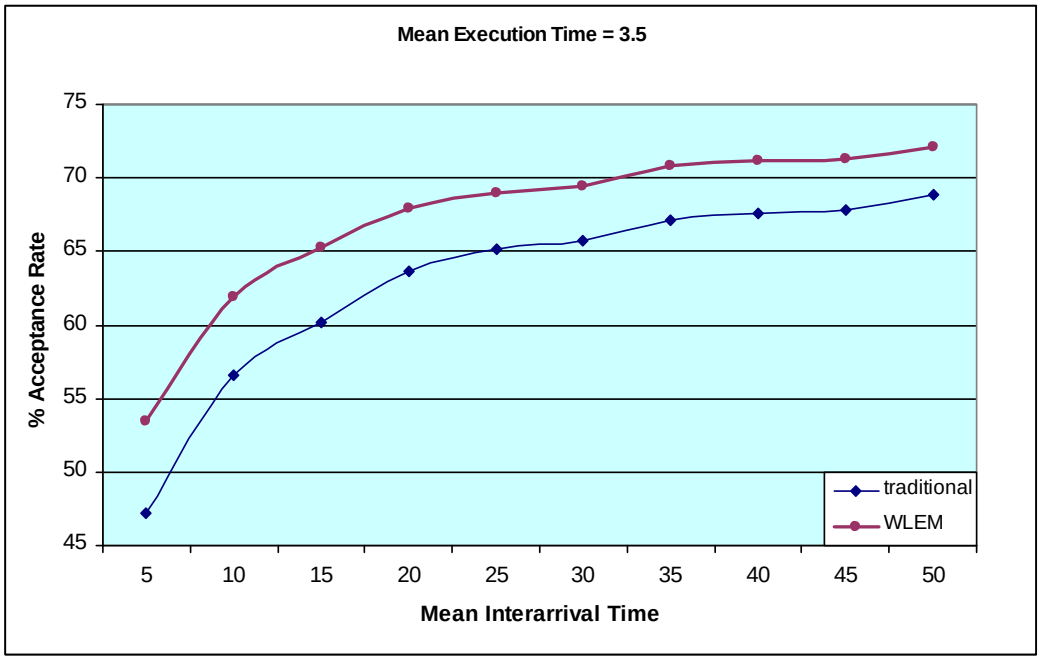

Fig. 7: Acceptance Rate at 1/μ = 3.5 (heterogeneous cluster)

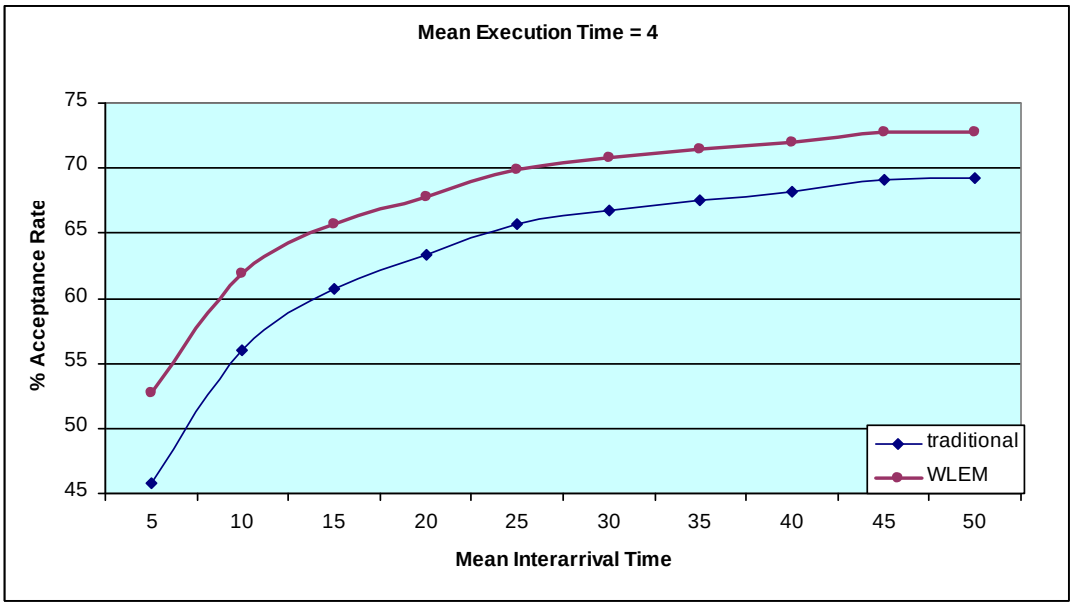

Fig. 8: Acceptance Rate at 1/μ = 4 (heterogeneous cluster)

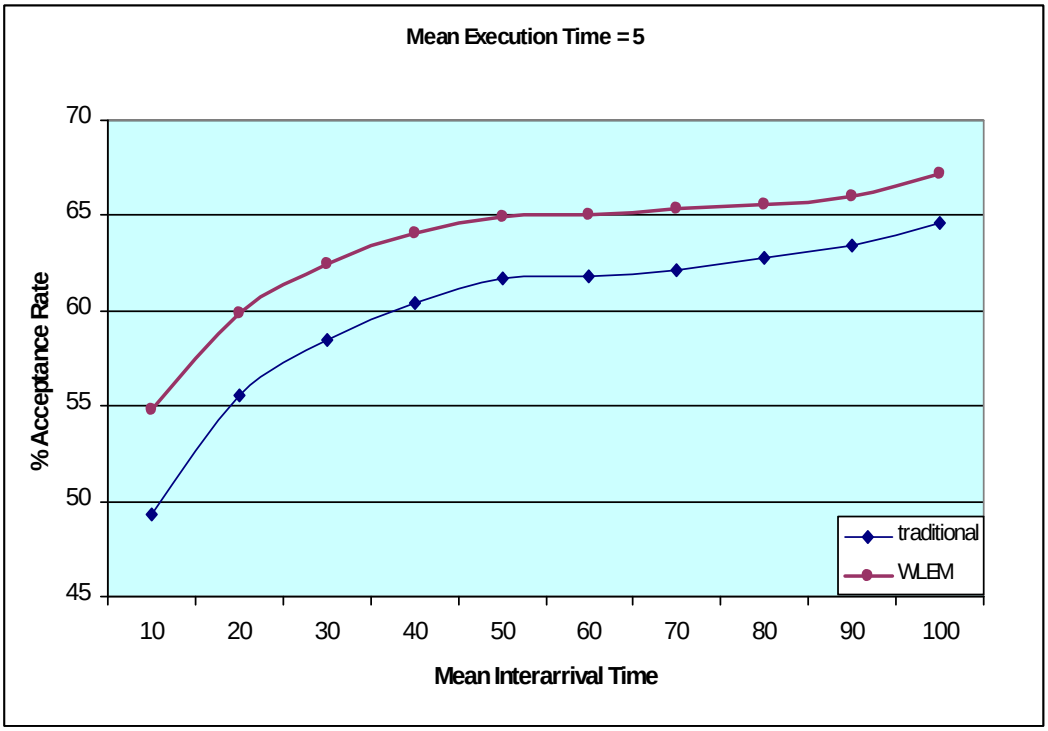

Fig. 9: Improvement at 1/μ = 5 (heterogeneous cluster)

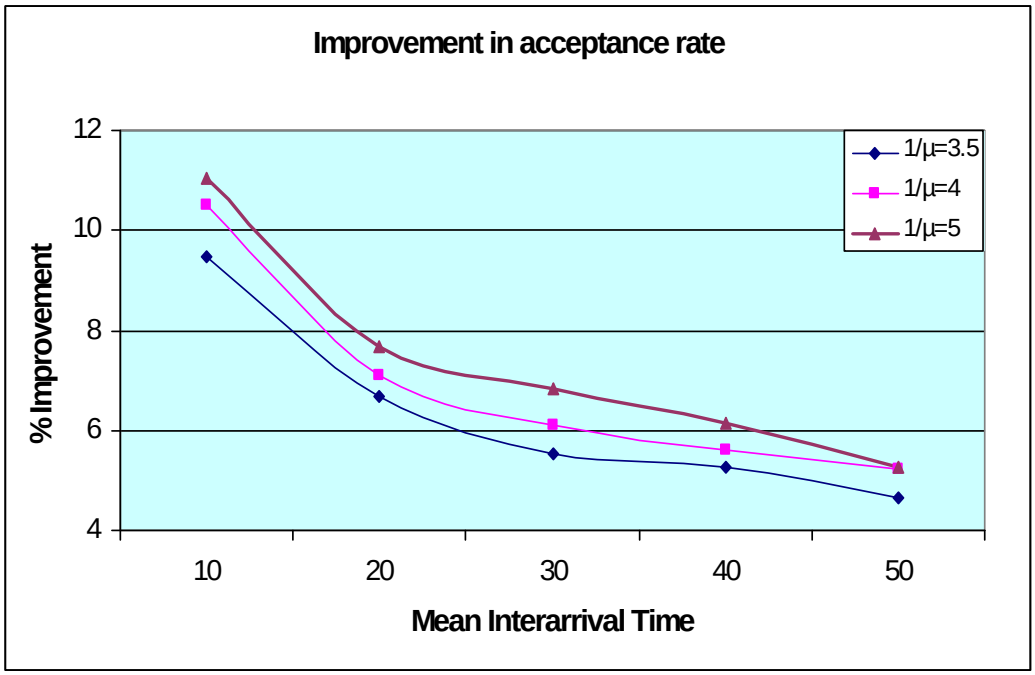

Fig. 10: Improvement at different values of 1/μ (heterogeneous cluster)

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed a heuristic scheduling algorithm to schedule real-time applications represented by F-J task graph on a cluster platform. This algorithm consists of two hierarchical schedulers. Both of them efficiently utilize the processing power of the cluster's processors and, consequently, improve acceptance rate. A simulation study has been conducted to evaluate the performance of our approach through a set of experiments on homogeneous and heterogeneous clusters. The results showed that our approach considerably outperforms a traditional scheduling approach in terms of acceptance rate. We plan to investigate applying our scheduling approach to multiprocessor/distributed platforms that include processors with dynamic voltage scaling with the intent of reducing power consumption.

REFERENCES

[1] B. N. Alahmad, S. Gopalakrishnan, Energy efficient task partitioning and real-time scheduling on heterogeneous multiprocessor platforms with QoS requirements Sustainable Computing: Informatics and Systems vol.1 ,issue 4,(2011) 314– 328.

[2] F. Fauberteauy, M. Qamhieh, S. Midonnet "Partitioned scheduling of parallel real-time tasks on multiprocessor systems", In the proceedings of 23rd Euromicro Conference on Real-Time Systems (ECRTS 2011), 2011

[3] G.Cordasco and A. Rosenberg, "Area-Maximizing schedules for series-parallel DAGs", Proceedings of Euro-Par, pp. 380-392, 2010.

[4] Hadis Heidari and Abdolah Chalechale, Scheduling in Multiprocessor System using Genetic Algorithm, International Journal of Advanced Science and Technology, vol.43, june 2012,pp.81-93.

[5] I.M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The design and implementation of an operating system to support distributed multimedia applications" IEEE journal on selected areas in communications, vol. 14, pp. 1280-97, 1996.

[6] J. Goossens and V. Berten, "Gang ftp scheduling of periodic and parallel rigid real-time tasks," in Proc. Of RTNS, 2010, pp. 189–196.

[7] K. Lakshmanan, S. Kato, and R. (Raj) Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in Proc. of RTSS, 2010, pp. 259–268.

[8] Michle Lombardi, Michela Milano, Luca Benini, "Robust Scheduling of Task Graphs under Execution Time Uncertainty", IEEE transactions on computers, 2011.

[9] M. Niemeier, A. Wiese, S. Baruah, Partitioned real-time scheduling on heterogeneous shared-memory multiprocessors, Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS 2011) , 2011.

[10] M. J. Oudshoorn, H. Lin, "Evolving toward an optimal scheduling solution through adaptivety", Journal of parallel and distributed computing, vol.62, issue,7, pp.1203-1222, july 2002.

[11] N. R. Satish, K.Ravindran, K. Keutzer, Scheduling task dependence graphs with variable task execution times onto heterogeneous multiprocessors, In EMSOFT '08 Proceedings of the 8th ACM international conference on Embedded software, 2008, pp. 149-158.

[12] O. Jaewon and W. Chisu, Genetic Algorithm Based Real Time Task Scheduling with Multiple Goals, Journal of systems and software, vol. 71, issue 3, pp. 245-258, May 2004.

[13] R. Ammar, A. Alhamdan, "Scheduling real-time fork-join structures in cluster computing", Int. J. of High Performance Computing and Networking , Vol.3, No.4, 2005, pp.262 – 271.

[14] R. Ammar, A. Hamdy and A. Hussein, "Efficient Execution of Real-Time Tasks on a Single Processor", International journal of Intelligent Computing and Information sciences, Vol.9, No.2, July 2009, pp. 31-40

[15] S.Baskaran, P. Thambidurai, Energy efficient real-time scheduling in distributed systems, IJCSI International journal of computer science issues, vol.7, issue 3, No.4, May 2010.

[16] S. Jin, G. Schiavone, D. Turgut, A performance study of multiprocessor task scheduling algorithms, J Supercomputer (2008) 43: 77–97, 2008.

[17] Bokhari S.A. "A shortest tree algorithm for optimal assignment across space and time in distributed processor system". IEEE Transaction on Software Engineering, 7(6), 1981.

[18] Cheng S.T. and Hwang S., "Optimal real-time scheduling with minimal rejections and minimal finishing time", Real-Time systems, vol. 20, pp. 229-53, 2001.
[19] W. Y. Lee, Energy-Efficient Scheduling of Periodic Real-Time Taks on Lightly Loaded Multicore Processors, IEEE Transactions on Parallel and Distributed Systems, Vol. 23, No. 3, March 2012.
[20] Xuan Lin, Anwar Mamat, Ying Lu_, Jitender Deogun, Steve Goddard , Real-time scheduling of divisible loads in cluster computing environmentsI, J. Parallel Distrib. Comput.Elsevier, 2010, pp. 296_308.
[21] Xiaomin Zhu, Xiao Qin, Meikang Qiu, "QoS- Aware fault-Tolerant Scheduling for Real Time Tasks on Heterogeneous clusters" IEEE transactions on Computers,Volume 60, Issue 6,June 2011, pp.800-812.
[22] Xiaomin Zhu,Jianghan Zhu,Manhao Ma, Dishan Qiu, "SAQA: A self adaptive QoS-aware Scheduling Algorithms for Real Time Tasks on Heterogeneous Clusters" Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, pp.224-232.
[23] X. Zhu, C. He, K. Li, Xiao Kin, "Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters" , Journal of parallel and distributed computing, volume 72, Issue 6,2012, pp. 751-763.
[24] Z. Deng, J.W.-s Liu, and S. Sun, "Dynamic scheduling of hard real-time applications in open system environment" presented at the real-time systems symposium, Washington, DC, 1996.