Global Journal of Computer Science and Technology: B
Cloud and Distributed
Volume 15 Issue 4 Version 1.0 Year 2015
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Hybrid Genetic Algorithms for Scheduling High-Speed Multimedia Systems

By Oluwadare Samuel Adebayo, Olabode Olatunbo,
Iwasokun Gabriel Babatunde & Akinyede Raphael Olufemi
The Federal University of Technology, Akure, Nigeria

Abstract- It has been observed that most conventional operating systems could not cope with the scheduling of multimedia tasks owing to the large size of these files. For instance, processing of multimedia tasks using the traditional operating systems are fraught with problems such as low quality of service and delay jitters. In order to address these problems, a scheduling algorithm christened hybrid genetic algorithm for multimedia task scheduling (HGAMTS) was developed. It employed heuristic knowledge of the problem domain to model a hybrid genetic algorithm in a multiprocessor environment. The system is made up of the scheduler model and the task model. The scheduler model consist a centralized dynamic scheduling scheme. In this scheme, all tasks arrive at a central processor (scheduler). The model has a minimum of five and maximum of ten processors. Attached to each processor is a dispatch queue.

Keywords: scheduling algorithms, hybrid genetic algorithm, multimedia system, operating system, multiprocessor system.

GJCST-B Classification : B.2.4

Strictly as per the compliance and regulations of:

Hybrid Genetic Algorithms for Scheduling High-Speed Multimedia Systems

Oluwadare Samuel Adebayo [α], Olabode Olatunbosun [σ], Iwasokun Gabriel Babatunde [ρ] & Akinyede Raphael Olufemi [ω]

Abstract- It has been observed that most conventional operating systems could not cope with the scheduling of multimedia tasks owing to the large size of these files. For instance, processing of multimedia tasks using the traditional operating systems are fraught with problems such as low quality of service and delay jitters. In order to address these problems, a scheduling algorithm christened hybrid genetic algorithm for multimedia task scheduling (HGAMTS) was developed. It employed heuristic knowledge of the problem domain to model a hybrid genetic algorithm in a multiprocessor environment. The system is made up of the scheduler model and the task model. The scheduler model consist a centralized dynamic scheduling scheme. In this scheme, all tasks arrive at a central processor (scheduler). The model has a minimum of five and maximum of ten processors. Attached to each processor is a dispatch queue. Communication is established between the scheduler and the processors through the dispatch queues. The scheduler ensured that each dispatch queue is filled with minimum number of tasks so that a processor could always find a task in its dispatch queue when it finishes executing a task. The algorithm was implemented using Java programming language. The experimental results were compared with two real-time conventional algorithms: rate monotonic (RM) and early deadline first (EDF). The result showed that the proposed algorithm has higher success rate ratio and guaranteed number of deadlines met.

Keywords: scheduling algorithms, hybrid genetic algorithm, multimedia system, operating system, multiprocessor system.

I. Introduction

The advent of multimedia files has placed additional challenge on the traditional operating systems. This is due to the fact that multimedia tasks are characterized by large files running sometimes into hundreds of gigabytes. Most of these files has to be processed in real-time and in continuous stream. Any delay in the processing would lead to low quality of service. Multimedia files are made up of text, graphics, audio and video. Although a little delay in the processing of text may not be noticeable but such little delay in audio or video may seriously affect the quality of service. In client-server systems, the high number of concurrent users trying to download data, sometimes in continuous streams may lead to considerable slow down. Some researchers have noted that the traditional Operating Systems as well as their extensions could not cope with these demands of multimedia applications (Plagemann et al., 2000; Yau and Lam, 1996; Neih and Lam, 1997; Goyal et al., 1996a; Lesilie et al., 1996).

Aurhor α σ ρ ω : Department of Computer Science, The Federal University of Technology, P.M.B. 704, Akure, Nigeria.
e-mails: samoluwadare2013@gmail.com, oolabode@futa.edu.ng, biwasokun@futa.edu.ng, olufemi_akinyede@yahoo.com

II. Review of Related Works

Scheduling algorithms is an active area of research which has received considerable attention over the years. In the literature, a number of scheduling algorithms has been proposed, for instance, in Tanenbaum (1994), a First-Come-First-Served (FCFS) algorithm which selects the task with the earliest arrival time was proposed. The system ensures that if it contains periodic tasks, their release time will be considered. The major drawback of this algorithm is that it makes no effort to consider a task's deadline. Liu and Layland (1973) proposed the Early Deadline First (EDF) algorithm which will always choose the task with the earliest deadline. The algorithm is optimal in a uni-processor system but does not consider priority and therefore cannot analyze it. The algorithm fails under overloading conditions (Thai, 2002; Tanenbaum, 1994).

A fuzzy scheduling algorithm is also proposed in (Lee et al., 1994). The algorithm uses task laxity and task criticality as system parameters. The simulation model which contains small number of tasks on a uni-processor system did not consider system overloads. All the tasks are assumed to be real-time and fairness is not considered in scheduling. In terms of real-time distributed systems, Thai (2002) developed a model in which the task with higher computation time is assigned to bottleneck processor and system's worst case processing time is computed. How the task with higher computational time is detected was not explained but the proposed algorithm has acceptable resistance to system overload especially when number of processors is increased. Also, the algorithm needs communication time between processors. Another drawback of the work is that the algorithm does not consider heterogeneous tasks and fairness.

Sabeghi et al. (2006) used fuzzy inference to schedule non-preemptive periodic tasks in soft real-time multiprocessing systems. Priority and deadline was used as tasks parameters while fuzzy inference engine was used to compute each task's priority and to select

the task with maximum priority to process. All tasks are assumed to be periodic and it is not clear whether the multiprocessor system that was proposed was homogeneous or heterogeneous. Since the system does not consider task's processing time the results are more similar to EDF and therefore not suitable for multiprocessing systems. Chen et al. (2005) proposed a scheduling algorithm that is suitable for both uni-processor and multiprocessor systems. It provides a method to detect work overloading and try to balance load with task dispatching. It is however, doubtful if the proposed model could handle multimedia data efficiently. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time tasks was proposed in (Brandt et al., 2003). The model can generate feasible schedules but the model is restricted to periodic tasks and the tasks' periods are changed dynamically when overloading occurs.

Alberto et al. (1994) proposed a dynamic scheduling of computer tasks using Genetic Algorithms (GA). The scheduling algorithm which is non-preemptive hard real-time is aimed at dynamically scheduling as many tasks as possible such that each task meets its execution deadline while minimizing the total delay time of all the tasks. It implements a sequential MicroGA that uses a small population size of 10 chromosomes running for 10 trials and using a rather high mutation rate with a sliding window of 10 tasks. They also developed parallel MicroGA model for parallel processors. The performance of the sequential MicroGA model and the parallel MicroGA model were compared with other algorithms namely FIFO, EDLF and SRTF for solving similar problem. The results showed that the sequential MicroGA and the parallel MicroGA models produced superior task scheduling compared to other algorithms tested. The work is limited because it was meant to handle hard real-time task and it used a small population size.

In Stutar et al. (2006) a memetic algorithm for task scheduling in multiprocessor systems was developed. The memetic algorithm was produced by hybridizing Genetic Algorithm with Simulated Annealing (SA). SA transverses the search space by testing random mutations on an individual. The mutation that increases fitness is accepted. Tasks are distributed among the processors in such a way that the precedence constraints are preserved and total execution time is minimized. It defines an order of processing tasks that are ready to run in a given processor. The memetic algorithms represent tasks in a task graph which are then mapped onto a multiprocessor system in a way that maintains precedence relations and ensure that all tasks are completed in shortest possible time. Even though the memetic algorithms seem to offer a promise at mitigating the shortcomings of GA, it was not implemented. Hence, the efficiency of the algorithms could not be ascertained.

Hamzel et al. (2007) also proposed a soft real-time fuzzy task scheduling for multiprocessor systems. The algorithm arranges real-time periodic and non-periodic tasks in multiprocessor systems. Since most static and dynamic optimal scheduling algorithms fail with non-critical overload, the fuzzy approach attempt to balance task loads of processors successfully, prevent starvation and ensure fairness which causes higher priority tasks to have higher running probability. Experimental results show that the proposed fuzzy scheduler creates feasible schedules for homogeneous and heterogeneous tasks. It also, considers task priority which causes higher system utilization and lowers deadline misses. However, the model is deficient because it does not consider scheduler processing time.

Mahmood (2000) proposed a hybrid scheduling algorithm for task scheduling in multiprocessor real-time systems. The system recorded significant improvements in guarantee ratio of tasks that arrived in the system. The system was not however, designed to handle multimedia tasks which consists both hard real-time and soft real-time components. Seyed et al. (2014) developed a genetic algorithm for optimization of integrated scheduling of cranes, vehicles and storage platforms at automated container terminals. The proposed algorithm introduced a random string of tasks to define precedence relations between tasks. The performance of the algorithm was evaluated using 10 small size test cases. A fairly near optimal solution that is similar to the existing simulated annealing algorithm was obtained. It was also reported that the proposed GA outperforms the existing algorithm when the number of tasks to be scheduled increase from 30 to 100.

Faghihi et al. (2014) employed the use of GA to schedule construction based on building information model (BIM). The project management triangle includes time, cost and quality. The task of developing project schedules that will satisfy the constraints imposed by time, cost and quality could be troublesome. The proposed GA model was applied to 21 construction projects and stable construction schedules were successfully generated.

Chiu-Hung et al. (2015) applied greedy GA to solve the problem of teacher transferring problems (TVPs). An improved neighborhood search algorithm was introduced into mutation operator. The result produced an optimal solution which performed better than classical methods for solving such problems. A hardware-aware rate monotonic scheduling algorithm for embedded multimedia systems was proposed in (Park and Yoo, 2010). The experimental results show that the algorithm improved the responsiveness of hardware tasks with little impact on software tasks. The output jitter reduced drastically. An improved CPU

scheduling algorithm based on multiprogramming environment was proposed in Arora et al. (2013). The introduction of pipelining into the CPU scheduling led to reduction in time latency. The proposed algorithm outperforms existing scheduling algorithms by 40-50%.

Notario et al. (2012) presents a multi-objective GA for task assignment on heterogeneous nodes. The assignment strategy used was based on GA to maximize task execution quality while minimizing energy bandwidth consumption. The result offered Pareto optimal solutions which were better than previous works that were reviewed. Similarly, Khan and Govil (2013) proposed a cost optimization technique of task allocation in heterogeneous distributed computing system (DCS). The proposed model considered the allocation of *m* tasks to *n* processors in a DCS using a modified tasks allocation technique which considers the processing capacity of each processor. The results show drastic reduction in processing cost.

This paper presents a hybrid genetic algorithm for scheduling high-speed multimedia systems using GA and maximum urgency first algorithm. A detailed discussion of the optimization model using mixed-integer linear programming is documented in (Oluwadare and Akinnuli, 2012).

a) *Hybrid Genetic Algorithms for Multimedia Task Scheduling in a Multiprocessor System (HGAMTS)*

HGAMTS combined a classical algorithm, Maximum Urgency First (MUF) with the multiprocessor algorithm proposed in (Mahmood, 2000). Detailed listing of the algorithm is as follows:

Let m = number of processors; task [i] = the initial task queue; l = length of a chromosome; pop = population size; n = number of tasks in a chromosomes which is initially set to 0.

While (more task queue to be scheduled) **do**
If not (task queue empty) **then**
Order the tasks in the task queue in non-decreasing order of their criticality.
Select the task with highest criticality.
If tasks have same highest criticalness, select the task having the highest
dynamic priority.
If tasks have same highest criticalness and equal dynamic priority then select
task with highest user priority.
If tasks have common on all three above factors, execute the task based on
first come, first served basis.
{Create the population as follows}
Select first *t* (*t* < = *l* – n) tasks from the task queue
Set *n* = *n* + 1
For *j* = *n* + 1 to pop **do**
For *i* = 1 to t **do**
Generate a random number *p (l…m)*
Call procedure random selector
Select randomly a position *q* (*l…n* + 1) in the chromosome that has not been occupied so far.
Insert (task [*i*], *p*) at position *q* of a chromosome *j* { that is (task [*i*],*p*} is the ith locus of chromosome).
Endfor
Endfor
Endif
If *n* > 1 **then**
Evaluate each member's fitness (call procedure evaluate)
Call procedure keep the best in order to select the best fit off springs
While not (termination condition) **do**
Perform reproduction based on relative fitness values of chromosomes
Call procedure crossover to perform crossover
Call procedure mutation
Endwhile
Select the best chromosome in the population as a solution.
Newtaskset = false
While not (newtaskset) **do**
Update the dispatch queues, if required, by assigning first *f* feasible tasks from the best chromosome.
Find the tasks that in the best chromosome that cannot meet their deadlines by performing the feasibility check. Let *r* be the count of such tasks.
Remove the tasks that have been sent to the dispatch queues or found not to be feasible from all the chromosomes of the population {*f* + *r* tasks will be removed}
Set *n* = *n* – *f(f*+ *r*)
Arrange the remaining tasks in the first *n* positions of each chromosome
Set newtaskset = true
Endif
Endwhile
Endif
Endwhile

The algorithm starts by ordering the task in non-decreasing order of their criticality. Ordering the tasks in this manner is to ensure that those tasks that are more critical in terms of timeliness are assigned higher criticality values. Depending on the consequences of missing a deadline, multimedia tasks are typically classified into three categories: hard real-time systems, firm real-time systems and soft real-time systems (Ramamritham, 1996).

Based on this categorization, tasks coming into the system are assigned criticality values with hard real-time systems tasks having the highest criticality value followed by firm real-time tasks; and soft real-time tasks. After arranging the tasks on the basis of their criticality values, the algorithm then checks to see if there are two

or more tasks with same criticality value. It selects the task among such category of tasks with a tie in their criticality value and orders them using dynamic priority.

Dynamic priority is imposed by the immediate conditions at the time of processing. In the design of HGAMTS, the tasks arriving are submitted dynamically at the scheduler (the central processor) which performs the schedulability check before passing the task to any of the dispatch queues associated with the processors in the multiprocessor environment.

In a situation whereby tasks have same criticality and dynamic priority another selection factor known as user priority is introduced. If tasks tie on criticality, dynamic priority and user priority the tasks are selected on first-come-first-served basis. The central goal of HGAMTS is to incorporate traditional scheduling heuristics to generate a feasible schedule by determining the processor which a task should be assigned to and the order in which tasks should be executed so as to meet their deadlines. This measure increases the guarantee ratio (percentage of scheduled tasks that meet their deadlines).

The algorithm then selects a set of tasks from the sorted list and generates the initial population. Each chromosome in the initial population is generated by assigning task in the tasks set to a randomly selected processor and inserting the pair (task, processor) in a randomly selected unoccupied locus of the chromosome. If the number of tasks is less than the chromosome size, then the first n loci of the chromosomes are used in solution encoding and active chromosome size is set to n. This ensures that genetic operators are applied only to the active part of the chromosome. The second while loop determines the best schedule by applying the genetic operators discussed earlier. The second while loop terminates, if the best chromosome has a fitness value equal to the number of tasks considered for scheduling (the size of active chromosome) or a maximum number of iterations have been completed. Once the best schedule for a set of tasks has been found, the dispatch queues are updated, if required. All the dispatched tasks along with tasks found infeasible are removed from all the chromosomes so that they are not reconsidered for scheduling. A task T_i is feasible if it could be submitted at the end of any of the dispatched queues such that $r_i \leq st(T_i) \leq d_i - c_i$ and $r_{i+} c_i \leq ft(T_i) \leq d_i$. Any task that fails to meet this condition is not feasible. The new task set includes the newly arriving tasks along with the tasks that have not been dispatched so far.

The chromosome syntax employed is such that each gene is a pair of decimal values (T_i, P_i) which indicates that task T_i is assigned to processor P_i. The position of genes in the chromosome indicates the order in which tasks should be executed. For instance, the chromosome representation shown in Figure 1 indicates that task 1 be executed on processor 4, task 5 on processor 1, task 2 on processor 3 and task 3 on processor 1. It also indicates that if two tasks are assigned to the same processor for instance, tasks 5 and 3, task 5 is executed first followed by task 3.

(1,4)	(5,1)	(2,3)	(3,1)

Figure 1 : Chromosome representation in HGAMTS

One of the advantages of the multiprocessor system is that two tasks assigned to two different processors may execute in parallel provided they do not require the same resource in exclusive mode. Tasks assigned to the same processor must execute in the specified order. In the chromosome representation scheme, each chromosome has a fixed length/size. This means that the maximum number of tasks that may be considered for scheduling at a time is bounded by the chromosome size. The remaining tasks together with the newly arriving tasks are kept in the task queue (Mahmood, 2000). When one set of task is scheduled, a new set of tasks from the task queue is selected for scheduling. If the number of tasks in the task queue is less than the chromosome size then only part of the chromosomes is used and application of genetic operators is restricted to that part only. The part of a chromosome being used is called the active part. It should be noted that the maximum size of the active part is equal to the chromosome size.

The position of a task on the chromosome which is a greedy consideration based on the domain-specific knowledge, determines the order in which the task will be executed. The closer the task is to the front of the chromosome, the greater are the chances of being scheduled. Also, the nature of the task that precedes a particular task may impose some constraints on where it can be placed. For instance, if two tasks require a resource, and one of them is in exclusive mode, the task in exclusive mode may prevent the other from being executed. The three genetic operators employed in HGAMTS are crossover, reproduction and mutation.

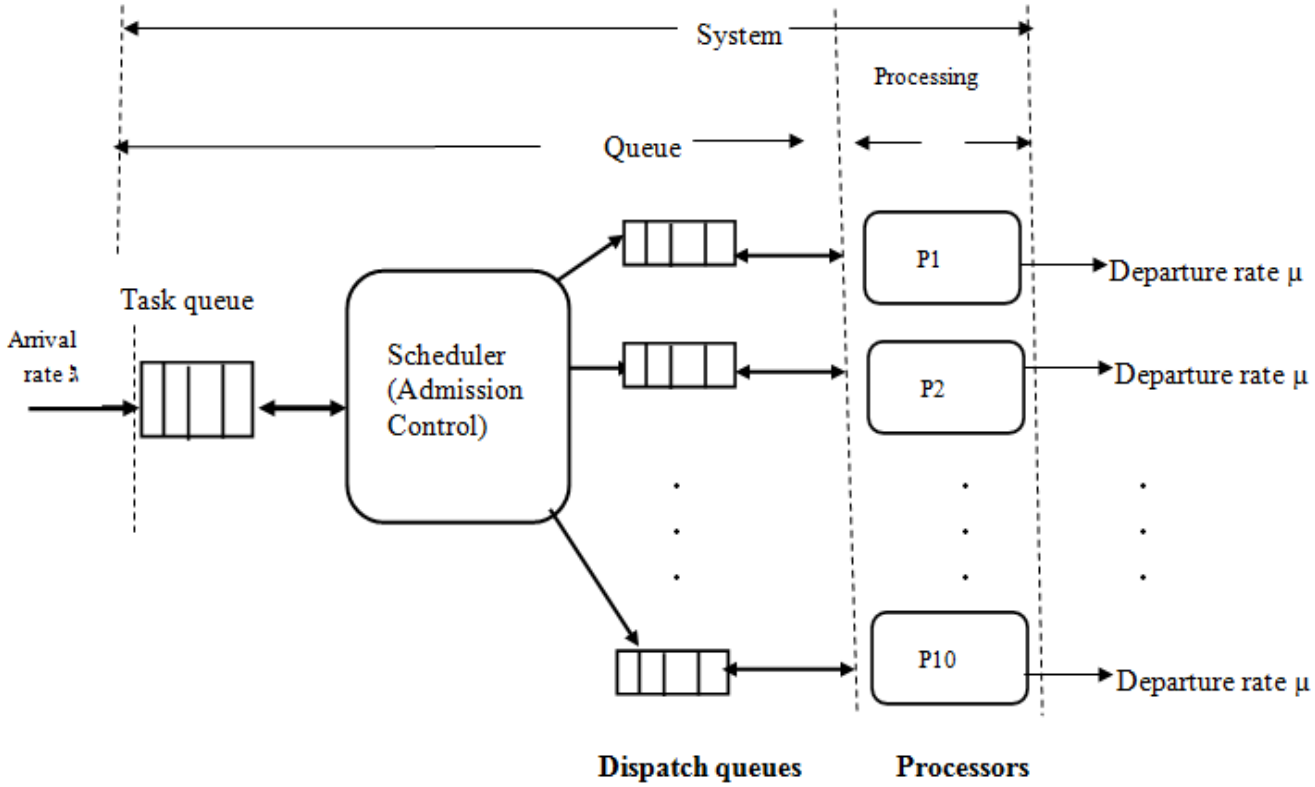

Figure 2 : Schematic representation of the partitioning scheme and the queuing

III. Experimental Set-up

Extensive simulation runs were carried out. For each simulation run, 300 tasks were randomly generated. The worst case computation time c_i of task T_i was randomly chosen between *MIN_C and MAX_C* and were set to 30 and 60 respectively. Deadline of task T_i was uniformly chosen between

$$r_i + 2 * c_i \text{ and } r_i + R * c_i$$

where $R \geq 2$

The inter arrival time of task is exponentially distributed with mean

$$1/(\lambda * m) * (MIN\_C + MAX\_C)/2$$

where m is the number of processors ($5 \leq m \leq 10$).

The value of λ was varied from 0.2 to 0.6 (Mahmood, 2000). Many values of the reproduction operator x (% of tasks to be killed before reproduction) were tried. Chromosome size was varied from 5 to 20 for different set of simulation runs. Also, population size was varied from 20 to 50. All the dispatched queues were of equal length for a particular simulation run. However, the length of dispatch queues was varied between 1 and 3 for different simulation runs.

IV. Results

The values presented in the Tables 1 through Table 2 are the average of 10 simulation runs and the maximum iteration was set to 700. The results obtained were compared with Rate Monotonic (RM) and Earliest Deadline First (EDF) algorithms.

a) Success Ratio at different Task Arrival Rates and Chromosome Sizes

The success ratio of the three algorithms was measured at different task arrival rates and chromosome sizes. The result is presented in Table 1

Table 1 : Success Ratio at different Task Arrival Rates and Chromosome Sizes

Task Arrival rate	Success Ratio					
	RM	EDF	HGAMTS at l = 5	HGAMTS at l = 10	HGAMTS at l = 15	HGAMTS at l = 20
0.2	88	93	95	95	100	100
0.25	86	89	91	93	100	100
0.3	82	84	87	90	98	100
0.35	78	78	80	86	98	100
0.4	66	69	77	84	95	98
0.45	61	65	70	83	93	97
0.5	60	64	68	80	89	97
0.55	60	58	65	76	85	95
0.6	57	55	60	72	80	95

l = Chromosome size

Source : Simulation studies, 2014

Table 1 revealed that at lower task arrival rates (0.2 – 0.3) the success ratio of the three algorithms is between 82 and 100%. RM recorded the least success ratio while HGAMTS recorded the highest. The success ratio drops as the task arrival rate increases. It was also revealed that HGAMTS has higher success ratio at all task arrival rates followed by EDF and RM. The success ratio of HGAMTS increases as chromosome size increases. In fact, the success ratio of HGAMTS is 100% at task arrival rate 0.25 and below for chromosome size 15. On the other hand, the success ratio reached 100% at task arrival rate 0.35 and below for chromosome size 20. Hence, it could be inferred that the higher the chromosome size, the better the performance of HGAMTS. Also, the gap between the performance of HGAMTS and the other two algorithms (RM and EDF) increases with increase in chromosome size.

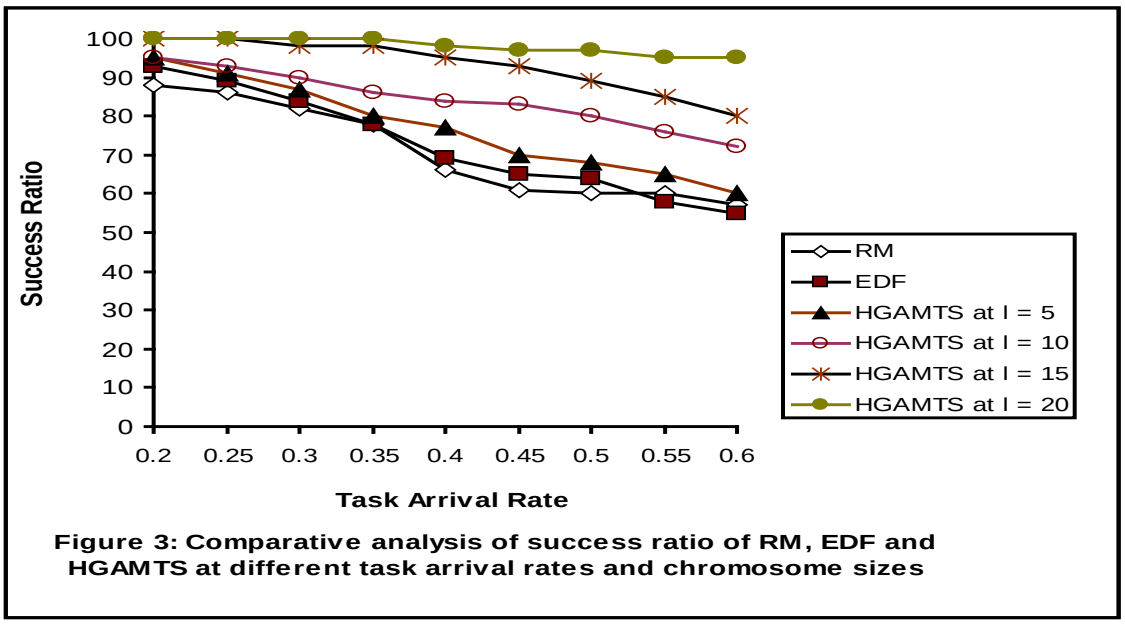

Figure 3: Comparative analysis of success ratio of RM, EDF and HGAMTS at different task arrival rates and chromosome sizes

b) Success Ratio at different number of iterations and chromosome sizes

A comparative analysis of the performance of the three algorithms was also carried out at different number of iterations. The result is presented in Table 2.

Table 2 : Success Ratio at different number of iterations and chromosome sizes

Number of iterations	Success Ratio					
	RM	EDF	HGAMTS at l = 5	HGAMTS at l = 10	HGAMTS at l = 15	HGAMTS at l = 20

100	10	21	41	43	47	48
200	22	24	46	47	58	59
300	25	26	48	51	62	65
400	28	29	52	58	69	72
500	31	35	57	63	83	85
600	36	43	60	66	89	96
700	45	51	61	67	92	98

Population size (p) = 40, l = Chromosome size

Source : Simulation studies, 2014

Table 2 revealed that success ratio increases with increase in number of iterations. RM has the least success ratio followed by EDF while HGAMTS has the highest success ratio across all iterations. It could also be observed that for HGAMTS, success ratio increases with increase in chromosome size.

c) *Performance of RM, EDF and HGAMTS at different utilization factors*

The maximum utilization factor was varied between 0.2 and 1.0 and processors were also varied between 5 and 10. The metric used is the guarantee ratio, which denotes the percentage of feasibly scheduled task sets that meet their deadline. The utilization was varied between 5 and 10. A task is schedulable on *m* processors if no task in the set misses its deadline.

Table 3 : Guarantee Ratio at different Utilization Factors

Utilization factor (α)	RM				EDF				HGAMTS			
	m=5	*m*=7	*m*=9	*m*=10	*m*=5	*M*=7	*M*=9	*m*=10	*m*=5	*M*=7	*m*=9	*m*=10
0.2	85	87	88	89	91	93	95	97	97	98	100	100
0.5	81	83	85	87	89	91	94	96	94	96	99	100
0.8	72	78	81	83	85	88	92	95	91	95	99	99
1.0	68	75	76	79	79	85	90	92	86	94	98	98

m = number of processors

Source : Simulation studies, 2014

Table 3 revealed that guarantee ratio decreases with increase in utilization factor. The best performance was obtained at $\alpha = 0.2$ for all the algorithms. However, HGAMTS recorded the best performance followed by EDF while RM had the least performance. Results in Table 3 also revealed that guarantee ratio increases with increase in the number of processors (*m*).

V. Conclusion

Multimedia files are usually large in size and must be processed within a time frame in order to guarantee quality of service. Meeting deadlines and achieving high resource utilization are the major goals of multimedia task scheduling. Multiprocessor systems offer a veritable opportunity to address some of the processing challenges presented by multimedia data. However, the complexity involved in the design of appropriate algorithms for task scheduling in multiprocessor systems is another area of challenge.

This research has shown that Hybrid Genetic Algorithm could be used to schedule multimedia tasks dynamically that arrives in the system. The algorithm used fixed size chromosome to encode the solution. However, the algorithm also allows part of the chromosome to be used whenever there are fewer tasks to occupy all the loci on the chromosome. When a part of the chromosome is used, the genetic operators are applied to the active part only. Tasks not submitted to the dispatch queues are reconsidered with newly arriving tasks. The simulation results confirm the effectiveness of the hybrid genetic algorithm for scheduling high-speed multimedia tasks in a multiprocessor system.

References Références Referencias

1. Alberto, C., Pico, G., and Wainwright (1994). Dynamic Scheduling of Computer Tasks using Genetic Algorithms. Proc. of the first IEEE Conference on Evolutionary Computation, IEEE World Congress on Computation Intelligence, June 26 – July 2, Orlando, Florida, pp.829– 833.
2. Arora H, Arora D., Goel D.A. and Jain P. (2013). An Improved CPU Scheduling Algorithm. International Journal of Applied Information Systems, Volume 6– No. 6, pp7-9
3. Brandt, S.A., Banachowski, S., Caixue, L., and Bisson, T. (2003). Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes, in Proc. 24th IEEE Intl. Real-Time

Systems Symposium, Cancun, Mexico, pp. 396 – 407.
4. Chen, G., Ozturk, O., and Kandemir, M. (2005). An adaptive locally-conscious process scheduler for embedded systems, in Proc. 11th IEEE Real-Time and Embedded Technology and Applications Symposium, San Francisco, CA, pp. 354 – 364.
5. Chiu-Hung C., Tung-Kuan L., Jyh-Horng C., Chung-Hung T. and Hsiu W. (2015). Optimization of teacher volunteer transferring problems using greedy genetic algorithms. Expert Systems with Applications 42 (2015) 668–678, www.elsevier.com/locate/eswa, Accessed on 04/02/2015
6. Faghihi V., Reinschmidt K.F and Kang J.H. (2014). Construction scheduling using Genetic Algorithm based on Building Information Model. Expert Systems with Applications 41 (2014) 7565–7578, www.elsevier.com/locate/eswa, Accessed on 04/02/2015
7. Goyal, P., Guo, X., and Vin, H.M. (1996a). A hierarchical CPU scheduler for multimedia operating systems. Proc. of USENIX sypm. on operating systems design and implementation (OSDI'96), Seatle, WA, USA, October 1996, pp. 107-121.
8. Hamzeh, M., Fakhraie, S.M., and Lucas, C. (2007). Soft real-time fuzzy task scheduling for multiprocessor systems, Int. Journal of Intelligent Technology, vol. 2 No. 4 pp. 211 – 236.
9. Khan F.N. and Govil K. (2013). Cost Optimization Technique of Task Allocation in Heterogeneous Distributed Computing System. Int. J. Advanced Networking and Applications Volume: 05, Issue: 03, pp:1913-1916
10. Lee, J., Tiao, A., and Yen, J. (1994). A fuzzy rule-based approach to real-time scheduling, in Proc. 3rd IEEE Conf. Fuzzy Systems, IEEE World Congress Computational Intelligence, Fl, vol. 2, pp. 1394 – 1399.
11. Leslie, I., McAuley, D., Black, R., Roscoe, T., Barham, P., Evers, D., Fairbairns, and Hyden, R. (1996). The design and implementation of an operating system to support distributed multimedia applications. IEEE Journal on selected areas in communications, vol. 14. No.7, Sept. pp. 1280-1297.
12. Liu, C.L. and Layland, J.W. (1973). Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment Journal of the ACM, Vol. 20, No. 1, January 1973, pp. 46 – 61.
13. Mahmood, A. (2000). A Hybrid Scheduling Algorithm for Task Scheduling in Multiprocessor Real-Time Systems. Technical Paper, Dept of Computer Science, University of Bahrain
14. Neih, J., and Lam, M.S., (1997). The design, implementation and evaluation of SMART: a scheduler for multimedia applications. Proc. of 16th ACM sypm. on operating system principles (SOSP'97), St. Malo, France, Oct. pp. 184-197.
15. Notario C.B.P, Rogier Baert R. and D'Hondt M (2012).Multi-Objective Genetic Algorithm for Task Assignment on Heterogeneous Nodes. International Journal of Digital Multimedia Broadcasting Volume 2012, Article ID 716780, 12 pages
16. Oluwadare, S.A. and Akinnuli, B.O. (2011). A Mixed Integer Linear Programming Model for Real-Time Task Scheduling in Multiprocessor Computer System. *Journal of Information and Communication Technology*, Universiti Utara, Malaysia.
17. Park J. and Yoo J. (2010). Hardware-Aware Rate Monotonic Scheduling Algorithm for Embedded Multimedia Systems. ETRI Journal, Volume 32, Number 5 pp. 657- 664
18. Plagemann, T., Goebel, V., Halvorsen, P., and Anshus, O. (2000). Operating system support for multimedia systems. The Computer Communications Journal, Elsevier, Vol. 3, No. 3, February, pp. 267-289.
19. Ramamritham, K. (1996). Dynamic Priority Scheduling, Real-time Systems Specification, Verification and Analysis, in M. Joseph (Ed.), Prentice Hall.
20. Sabeghi, M., Naghibzadeh, M., and Taghavi, T. (2006). Scheduling non-preemptive periodic tasks in soft real-time systems using fuzzy inference, in Proc. 9th IEEE Intl. Symp. Object and Component-Oriented Real-Time Distributed Computing, Gyeongju, Korea, pp. 27 – 32.
21. Seyed M.H., Said H.T. and Omid M. (2014) A genetic algorithm for optimization of integrated scheduling of cranes, vehicles, and storage platforms at automated container terminals. Journal of Computational and Applied Mathematics, 270 (2014) pp 545-556, www.elsevier.com/locate/cam, Accessed 10/01/2015
22. Sutar, S.R, Sawant, J.P and Jadhav, J.R. (2006). Task Scheduling for Multiprocessor Systems using Memetic Algorithms, p 27/1 – 27/9.
23. Tanenbaum, A.S. (1994). Distributed operating systems: Prentice Hall. pp. 250 – 275.
24. Thai, N.D. (2002). Real-time scheduling in distributed systems, in Proc. International Conference Parallel Computing in Electrical Engineering, Warsaw, Poland, pp. 165 –170.
25. Yau, D.K.Y., and Lam, S.S. (1996). Operating system techniques for distributed multimedia. Technical Report, TR-95-36 (revised), Dept., of Computer Sciences, University of Texas at Austin, TX, USA.