EXECUTION TIME COMPARISON OF ALGORITHMS FOR THE ASSIGNMENT OF REAL TIME TASKS FOR MULTIPROCESSORS

Héctor Silva-López[1], Sergio Suárez Guerra[2]

[1] Physics Department, CINVESTAV-IPN, Av. IPN. 2508, A.P. 14-740, 07360, DF, Mexico.
hsl@fis.cinvestav.mx

[2] Center for Computing Research, National Polytechnic Institute, Mexico, A.P. 75-476, C.P. 07738, Zacatenco, DF, México.
ssuarez@cic.ipn.mx

Abstract. Optimal scheduling of real-time tasks on multiprocessor systems is known to be computationally intractable for large task sets. Any practical scheduling algorithm for assign real-time tasks to a multiprocessor system presents a trade-off between its computational complexity and its performance. In this work, we give in this paper a survey on the execution time and performance of these algorithms, structuring this work in two steps. In the first step, we simulate their execution to obtain their performance and the execution time of four algorithms, using a low number of tasks. After these results were extrapolated to obtain the execution time that it would take for a bigger number of tasks than the simulated. In the second step, we chose two algorithms for partitioned its and executing on a shared memory system using Pthreads and on a distributed memory system employing Parallel Virtual Machine and Message Passing Interface tools.

1 Introduction

There are two strategies for real-time tasks scheduling on a multiprocessor system. In a Global scheme each occurrence of a real-time task may be executed on a different processor. In contrast, a partitioning scheme enforces that all occurrences of a particular task are executed on the same processor. Among the two methods, the partitioned method has received the most attention in the research literature. The main reason for this is that the partitioned method can easily be used to guarantee run-time performance (in terms of schedulability). Dhall and Liu in [1], two heuristic assignment schemes are proposed, referred to as the RMNF (Rate Monotonic Next-Fit) and the RMFF (Rate Monotonic First-Fit). The schemes are based on the next-fit and first-fit bin-packing heuristic, respectively. In both schemes, tasks are sorted in no decreasing order according of their periods before the assignment is started. The FFDUF (First-Fit Decreasing-Utilization Factor)

method is a variation of the first-fit heuristic scheme. Here tasks are sorted in the order their load factor [2]. In [3], a best-fit bin-packing heuristic is used as the basis for the monotonic best-fit (RMBF) schemes. Similar to RMFF, the RMBF attempts to assign tasks to processors that have been marked as full. In [4], two heuristic assignments proposed, the RMST (Rate Monotonic Small Tasks) algorithm. It is for systems in which tasks assigned to each processor are scheduled rate-monotonically. It first sorts the periodic tasks in nondecreasing order according to their parameters X_i's, which are calculated with the following equation:

$$X_i = \log_2 p_i - \lfloor \log_2 p_i \rfloor$$

It then assigns the tasks in this order on processors in the first-fit manner. The other algorithm is the RMGT (Rate Monotonic General Tasks), first partitions all periodic tasks two subsets according to their utilization. Tasks whose utilization is equal to or smaller than 1/3 are in one subset. These tasks are first assigned to processors according to RMST algorithm. Then, the large tasks whose utilization is larger than 1/3 are assigned on the first-fit basis to processors each of which has at most one task assigned by RMST algorithm.

The non-partitioned method has received considerably less attention, mainly because the following limitations. First, no efficient schedulability tests currently exist for the non-partitioned method. The only known necessary and sufficient schedulability test for non-partitioned method has an exponential time-complexity [5]. The complexity can be duced with sufficient schedulability tests to a polynomial [6, 7, 8, 9] or pseudo-polynomial [8] time complexity. Second, no efficient optimal priority-assignment scheme has been found for the non-partitioned method.

The purpose of this paper is to survey execution time of four algorithms for the assignment of Real-Time tasks for multiprocessors.

The remainder of this paper is organized as follows. In Section 2 the system model described. Later on in the Section 3, we present four representative algorithms which simulated to obtain the execution time for each Algorithm. Once obtained this time, results are extrapolated to obtain the execution time of a number of tasks are bigger than the simulated. In the Section 4 the two algorithms are assigned to a shared-memory multiprocessor system and to local memory distributed system. The obtained results are analyzed in the section 5. Finally in the section 6, summarizes the results reported in paper.

2 System model

We considerer a set of tasks $T=\{T_1,..., T_n\}$ of n periodic preemptive real-time tasks running on different processors. The tasks are independent (they don't share resources) have no precedence constrains. Each task T_i arrives in the system at time a_i. The life-time

of each task T_i consists of a fixed number of instances r_i. After the execution of r_i instances, the task leaves the system. The time interval between the arrival of the first instances of two consecutive tasks T_x and T_y is defined as $L_{xy} = a_y - a_x$. In this model, T_i is the period and C_i is the worst case computation time of task. In their seminar work, Liu and Layland [10] showed that the utilization of system is the amount of processor load in percentage in the system and it is expressed as:

$$U = \sum_{i=1}^{n} C_i / T_i \tag{2}$$

A schedule of periodic tasks is feasible if each task T_i is assigned at least C_i before its deadline at every instance. The problem of planning a set of tasks with small load factors is taken from [5], expressed as:

$$\alpha := \max_{i=1,\ldots,n} U_i \tag{3}$$

It represents the maximal load factor of any single task. For all practical purposes, we may assume that a task set contains only tasks if $\alpha \leq ½$.

We initially assume the following system model:

1) Tasks are independent, arrive periodically and can be preempted. Hence, at every moment, a dispatcher determines which task to execute. Tasks do not require exclusive access to any other resource than a processor.
2) The cost of pre-emption is zero because we consider cache misses that occur when a task arrives to be included in the execution time.
3) Worst case execution time is known a priori.
4) Only partitioned scheme algorithms are considered.

3 Task assignment algorithms simulation

The algorithms to simulate are the RMNF, RMFF, RMST and the RMGT. The simulation consists on executing from 100 to 1000 tasks with increments of 1. In each increment the algorithm is evaluated from 1 to increment. The tasks are generated in the following way:
1) The number of tasks of $100 \leq n \leq 1000$
2) The period is generated following uniform distribution function with $1 \leq T_i \leq 500$
3) The compute time is generated of $0 < C_i \leq \alpha T_i$
4) The performance of the system is measured using two values of load factor, $\alpha = 0.2$ and $\alpha = 0.5$
5) Every increment runs 10 times and an average is obtained

The executed time of each algorithm is obtained with a Pc Intel Pentium III 650 MHz with 128 Mb of RAM and running on the Operating System Linux. The function used for the measurements is psched_get_time ().

Since an optimal task assignment cannot be calculated for large task sets, we use the total load $\left(U = \sum_{i=1}^{k} u_i\right)$ used in [11], to obtain a Lower Bound for the number of processors required, where u_i is the utilization of a task T_i.

The results can be seen in the figure 1 and figure 2 with load factor of α =0.2 and α =0.5, the worst algorithm is the RMNF, followed by the RMFF, because they need a bigger number of processors to execute the same quantity of tasks than the others. The two algorithms proposed by Burchard are better, specifically for the RMGT needs a smaller number of processors than the RMST, for example for α =0.2 and 1000 tasks, the RMST needs 137 processors, while for the RMGT only needs 133 processors, for α =0.5 and 1000 tasks, the RMST occupies 339 processors and the RMGT only occupies 300 processors.

3.1 Execution time extrapolation for α=0.2 and α =0.5

In the table 1and table 2 represents execution time of each algorithm with α =0.2 and =0.5. The time in clock ticks.

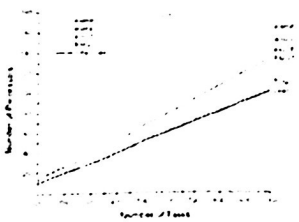

Fig. 1 Sequential simulation for α =0.2.

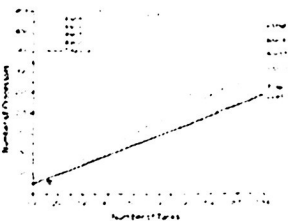

Fig. 2 Sequential simulation for α =0.5.

Table 1. Sequential Time, α=0.2

Alg \ Tasks	100	200	300	400	500	600	700	800	900	1000
Gen_tasks	2	3	4	6	7	8	10	10	12	14
RMNF	0	0	0	1	2	4	4	5	6	8
RMFF	1	9	33	74	151	261	410	632	934	1237
RMST	1	9	32	76	150	262	419	637	1207	1219
RMGT	1	9	31	76	151	262	419	634	895	1217
Total_Load	0	0	0	0	0	0	1	1	1	2

Table 2. Sequential Time, α=0.5

Alg \ Tasks	100	200	300	400	500	600	700	800	900	1000
Gen_tasks	2	3	4	5	7	8	9	11	12	14
RMNF	0	0	0	1	2	3	4	5	7	9
RMFF	2	15	43	108	228	378	611	895	1363	1835
RMST	2	14	42	110	219	369	595	875	1266	1794
RMGT	1	14	37	115	212	281	455	671	958	1324
Total_load	0	0	0	0	0	0	0	0	0	1

3.2 Polynomial approximation evaluation, α =0.2 and α =0.5

Using the approaches polynomial for each algorithm, we have calculated the time would take the program (in clock ticks) for a large number of tasks, see table 3 and table 4.
We can convert the clock ticks to seconds, with the following expression: clock ticks / CLK_TCK. Where: CLK_TCK is equal to 18.2.

Table 3. Execution time for n tasks, α=0.2

Alg \ Tasks	1,000	5,000	10,000	50,000	100,000	500,000	1,000,000
Gen tasks	13.43667	65.31667	130.16667	648.96667	1297.46667	6485.46667	12970.4667
RMNF	7.81394	162.94858	629.59433	15271.8583	60847.4333	1516361.83	6063029.33
RMFF	1244.6365	153486.713	1220438.05	151686883	1212564573	1.5148E+11	1.2117E+12
RMST	1309.15333	51092.8333	214572.433	5572409.23	22394705.2	561973073	2248946033
RMGT	1219.378	140923.05	1108327	135932407	1089586921	1.3598E+11	1.0876E+12
Tot Load	1.85121	84.08033	357.32133	9361.43333	37662.0333	945885.233	3785710.23
Total	3796.26965	345814.942	2544454.57	293216981	2324646006	2.8802E+11	2.3016E+12

4 Parallel programming

We choose the RMST and RMGT algorithms for obtained the better performance than the others four algorithms. In the following sections the two algorithms were implemented for a multiprocessor system and for a distributed system.

Table 4. Execution time for n tasks, α=0.5

Alg \ Tasks	1,000	5,000	10,000	50,000	100,000	500,000	1,000,000
Gen_tasks	13.47	66.55	132.9	663.7	1327.2	6635.2	13270.2
RMNF	8.26943	256.13583	1043.24333	26431.0833	105893.333	2650689.33	10604429.3
RMFF	1845.26233	256048.158	2081968.63	263781271	2113875044	2.646E+11	2.1172E+12
RMST	1836.90367	267825.117	2209237.57	285183961	2274161939	2.8502E+11	2.2809E+12
RMGT	1321.13	197046.45	1638365.6	211667789	1700557618	2.133E+11	1.7071E+12
Total_load	2.005	39139571.3	7349812549	7.6553E+14	1.0147E+17	8.1499E+21	1.0468E+24
Total	5027.04043	39860813.7	7355743297	7.6553E+14	1.0147E+17	8.1499E+21	1.0468E+24

4.1 Parallel programming for shared memory

The execution of parallel programs for shared memory was realized in a Multiprocessing Computer integrated by 4 Processors (Pentium III to 750 MHz) in cascade and an Operating System Linux NET4.0 for Linux 2.4

For the implementation with Pthreads, the time were obtained in second for load factors of α=0.2 and α=0.5, the results are compared between the RMST and RMGT algorithms. In the figure 3, we can observe that the execution time for the RMST and RMGT algorithms are the same for a maximum number of threads of 10, while in the figure 4, the RMST algorithm has an execution time less than the RMGT algorithm. In the two figures is obtained that when grow the number of threads, the execution time is the same when the number of threads starting of four threads, due principally because the maximum number of processors in the computer used is four.

4.2 Parallel programming for distributed memory

In this part of paper two tools were used for the distributed memory. The first tool is Parallel Virtual Machine (PVM) and the second tool is Message Passing Interface (MPI).
For the implementation of these two tools we uses a Cluster of 9 computers (1 server and 8 clients) Pentium II to 450 MHz and 256 Mb of RAM, with an Operating System LINUX Net Hat. For the communication of each node it uses a Switch Fast Ethernet (Switch Intel Express 510T 10/100 Fast Ethernet) and cabled type FTP.

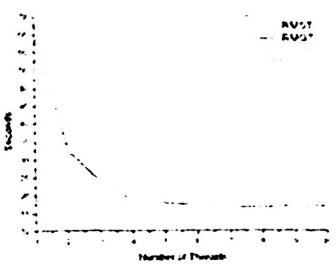

Fig. 3 Threads for α=0.2.

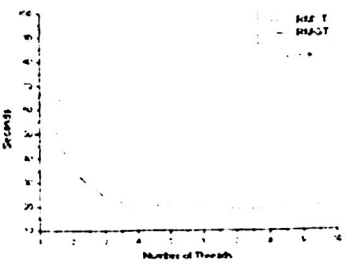

Fig. 4 Threads for α=0.5.

4.2.1 Performance Model

The objective of Performance Model is developing mathematical expressions that specify the execution time. The execution time of a parallel program is the lapse time since the first processor begins its execution until the last processor finishes its execution.

Because of space restrictions, we do not include here the details of the Performance Model of our application.

4.2.2 Parallel Virtual Machine.

Based on the communications time, to the model and the experimental results the following data were obtained, the data can be observed in the table 7, the time in seconds.
In the figure 5 correspond to experimental results and to performance model. We can appreciate that the algorithm RMST has a similar behaviour to performance model, while for the RMGT algorithm has a execution time bigger that the RMST algorithm. Based on the communications time, to the model and the experimental results, the following corresponding data were obtained for α=0.5, they are shown in the table 8, the time in seconds.

Table 7. Theoretical and experimental results with PVM, α=0.2

Slaves	Tcomunications	Tmodel	Texp-RMST	Texp-RMGT
1	0.253	78.41	97.50	125.27
2	0.429	39.51	52.09	66.82
3	0.606	26.66	34.17	42.40
4	0.782	20.32	27.48	35.41
5	0.958	16.59	20.64	25.87
6	1.134	14.14	19.24	24.24
7	1.311	11.86	14.99	20.54
8	1.486	9.24	13.95	18.78

Table 8. Theoretical and experimental results with PVM, α=0.5

Slaves	Tcomunications	Tmodel	Texp-RMST	Texp-RMGT
1	0.253	78.41	99.07	123.62
2	0.429	39.51	59.48	73.60
3	0.606	26.66	34.24	58.30
4	0.782	20.32	27.30	48.43
5	0.958	16.59	20.42	35.98
6	1.134	14.14	18.26	30.85
7	1.311	11.86	15.23	27.89
8	1.486	9.24	13.76	26.60

In the figure 6 correspond when the load factor is equal to 0.5, the RMST algorithm has behaviour seemed to presented in the figure 5, compared against the model. But the RMGT algorithm has a different behaviour to present in the figure 5. This is due mainly to great load factor (0.5), and for the number of tasks to be assigned to the processors by the RMFF algorithm is bigger than the load factor (0.2). Besides the RMFF algorithm presents a bad performance.

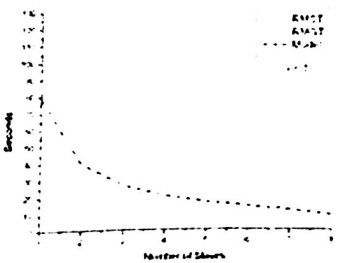

Fig. 5 PVM, α=0.2.

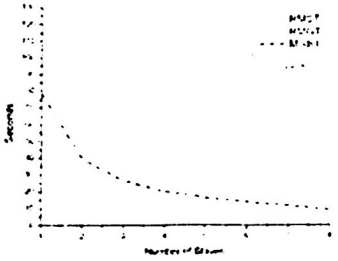

Fig. 6 PVM, α=0.5.

4.2.3 Message Passing Interface

Based in the communication time, to the model and the experimental results for the load factor equal to 0.2, we can observed in the table 9 the results obtained when the RMST and RMGT algorithms were executed in MPI, the time in seconds.

Table 9. Theoretical and experimental results with MPI, α=0.2.

Slaves	Tcomunications	Tmodel	Texp-RMST	Texp-RMGT
1	0.253	78.41	122.35	125.42
2	0.429	39.51	62.84	78.78
3	0.606	26.66	41.95	50.07
4	0.782	20.32	31.08	32.37
5	0.958	16.59	25.60	26.28
6	1.134	14.14	21.11	21.36
7	1.311	11.86	18.14	18.34
8	1.486	9.24	16.11	16.33

In the figure 7 a similar behaviour but no equal is observed in the figure 5, but in MPI for the RMST algorithm has behaviour almost similar to the RMGT algorithm when the

number of slaves are between 4 and 8 and when the number of slaves is equal to 1. Based in the communications time, to the model and the experimental results for α=0.5 (see table 10), the time in seconds.

Table 10. Theoretical and experimental results with MPI, α=0.5.

Slaves	Tcomunications	Tmodel	Texp-RMST	Texp-RMGT
1	0.253	78.41	173.95	124.21
2	0.429	39.51	85.06	65.86
3	0.606	26.66	55.34	50.26
4	0.782	20.32	42.82	40.64
5	0.958	16.59	33.21	32.22
6	1.134	14.14	28.15	22.86
7	1.311	11.86	24.28	21.85
8	1.486	9.24	21.29	20.25

In the figure 8 the behaviour of the algorithms is different compared with the figures 5, 6 and 7, the RMST algorithm consumes more execution time than the RMGT and model. It has behaviour almost similar to the RMGT algorithm when the number of slaves is from 4 to 5 and from 7 to 8 slaves.

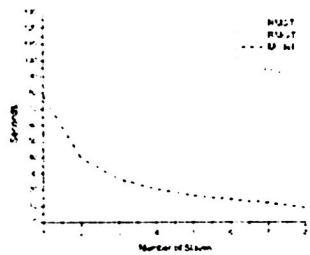

Fig. 7 MPI, α=0.2.

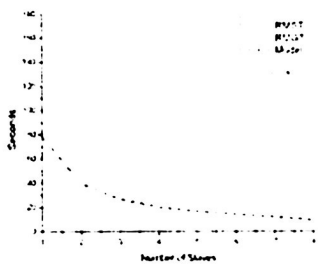

Fig. 8 MPI, α=0.5.

5 Results Comparison

In this section the execution times are presented for executing the RMGT algorithm in the sequential time as in the parallel time. The RMGT algorithm was chosen because it has a better performance that the RMST algorithm and as the execution times are very similar, it doesn't have any interest in presenting the two algorithms.

In sequential form the RMGT algorithm present an execution time less than RMST gorithm for a load factor of 0.2 and 0.5, the times obtained in experimental and theoretical form for 1,000 tasks can be observed in the table 11 (the time in seconds). In this table execution times in theoretical and experimental form are very similar for that we concludes that the theoretical results presented in the table 2 are very near to the real execution for a quantity of a lot tasks.

Table 11. Sequential Time for RMGT

Load Factor	Experimental	Theoretical
0.2	66.87	70.00
0.5	72.75	72.59

The parallel times for threads, PVM and MPI can be observed in the table 12, for load factor of 0.2 and 0.5, the time in seconds.

Table 12. Parallel Time for RMGT

Load Factor	Threads	PVM	MPI
0.2	17.18	22.35	20.57
0.5	20.56	30.33	24.29

In the table 12, the time presented for the threads is obtained of an average between time obtained of 4 and 10 threads, for the two load factors. For PVM and MPI the average time was obtained between 5 and 8 slaves.

The time presented is the desired, for PVM and MPI have a execution time bigger than the execution time of the threads, this is due to the communication time is implicit in execution of a distributed system.

6 Conclusions

We have analyzed the performance of four algorithms for the real time tasks scheduling in multiprocessor systems where their executed time were obtained. These algorithms are the RMNF, RMFF, RMST and the RMGT. In the first part of this work, the executed time obtained when simulating them were extrapolated to obtain approximate polynomial that give us an idea of the time that would take each algorithm in executing for a large number of tasks, being the time practically impossible of carrying out. For example, for load factor equal to 0.2, it would be impossible its execution after of 50,000 tasks since would take the program in execute in 6.215 months and for a load factor of 0.5, it is possible after of 10,000 tasks since would take 12.82 years.

In the second part of this work, two algorithms were chosen whose performance was much better than the other ones. The RMST and RMGT algorithms were partitioned for shared memory using threads and for distributed memory using PVM and MPI tools.

The final comparison of results shows us that when obtaining the approximate polynomials they give very near results to the obtained in experimental form, for what, the results were presented for a large number of tasks that not this very far from the reality. On the other hand, to the partitioned two algorithms the results were obtained that we were expected, that is, for the threads from 4 to 8 threads, the execution time vary very little besides were compared with the obtained in PVM and MPI. Using threads presents a better execution time than in PVM and MPI, because to execution time we need to add the communication time, implicit in a distributed system.

References

[1] J. Y. T. Leung and J. Whitehead. "On the complexity of fixed-priority scheduling of periodic, real time tasks". Performance Evaluation, 2(4):237-250, December 1982.

[2] S.K. Dhall and C. L. Liu. "On a real time scheduling problem". Operations Research, 26(1):127-140, January/February 1978.

[3] S. Davari and S. K. Dhall, "An on line algorithm for real time allocation", 19th Ann. Hawaii Int'l Conf. System Sciences, pp 133-141, 1986.

[4] Y. Oh and S. H. Son. "Tight performance bounds of heuristics for a real time scheduling problem", Technical Report CS-93-24, Univ. of Virginia, Dept. of Computer Science, May 1993.

[5] Burchard, J. Liebeherr, Y. Oh, and S.H, Son. "New strategies for assigning real time tasks to multiprocessor systems". IEEE Transactions on Computers, 44(12):1429-1442, December 1995.

[6] J.Y. T. Leung. "A new algorithm for scheduling periodic, real time tasks". Algorithmica, 4(2):209-219, 1989.

[7] B. Andersson. "Adaption of time-sensitive tasks on shared memory multiprocessor: A framework suggestion. Master's thesis", Department of Computer Engineering, Chalmers University of Technology, January 1999.

[8] S. Lauzac, R. Melhem, and D. Mossé. "Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor". In 10th Euromicro Workshop on Real Time Systems, pages 188-195, Berlin, Germany, June 17-19, 1998.

[9] L. Lundberg. "Multiprocessor scheduling of age constraint processes". In 5th International Conference on Real Time Computing Systems and Applications, Hiroshima, Japan, October 27-29, 1998.

[10] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard Real Time environment", J. ACM, Vol. 20, no. 1, pp. 46-61, Jan. 1973.

[11] A. Burchard, J. Liebeherr, Y. Oh, and S.H, Son. "A linear time Online Task Assignment scheme for multiprocessor systems", Proc. 11th IEEE Workshop Real-Time Operating Systems and Software, pp. 28-31, May 1994.