Multiprocessor Real-time Scheduling Using an Optimization-based Technique

Anca Hangan, Gheorghe Sebestyen and Lucia Vacariu
Computer Science Department, Technical University of Cluj Napoca, Cluj Napoca, Romania

Keywords: Real-time Scheduling, Optimization, Genetic Algorithm, Multiprocessor Systems, Real-time Transactions.

Abstract: The paper presents an optimization-based technique that enhances the schedulability of real-time transactional multiprocessor systems. The technique addresses two important aspects: task allocation and task deadline assignment. In order to satisfy real-time restrictions we combine genetic search and simulation to fine tune the system's configuration. To reduce the solution search space, we propose a hybrid technique for finding feasible scheduling solutions. We determine task deadlines through a heuristic and then use the optimization-based approach to find a solution for task allocation to processors. We evaluate the performance of the proposed techniques by using automatically generated transaction sets. Finally, we compare the optimization-based technique with related work and we analyze the results.

1 INTRODUCTION

One of the main tasks of a real-time system's designer is to configure the application on a given platform and find a proper scheduling strategy so that all tasks satisfy their time. In the case of real-time uniprocessor systems, there are several, widely studied optimal algorithms that find a feasible schedule for a given setup (Liu and Layland, 1973), such as Rate Monotonic (RM) and Earliest Deadline First (EDF).

For real-time multiprocessor systems the search space for a feasible scheduling solution is multi-dimensional. There are far more restrictions and therefore finding a feasible solution is much more complex (Davis and Burns, 2009). Most multiprocessor scheduling algorithms offer real-time guarantees only at very low resource utilization rates (Bertogna and Baruah, 2011) compared to their uniprocessor equivalents, or they are very difficult to implement in real-world cases (Baruah, et al, 1996). As multicore and distributed systems are becoming the typical computing platforms for a wide range of real-time applications, recent research efforts are mostly directed towards finding pragmatic solutions for multiprocessor systems.

In order to reduce the complexity of the real-time multiprocessor scheduling problem, one often used approach is to divide it into a number of sub-problems, such as:

- Allocation of tasks to available execution nodes;
- Setting deadlines for tasks contained in distributed transactions;
- Applying local (uniprocessor) scheduling techniques for tasks' execution.

In this context, we propose an optimization-based technique that combines genetic search and simulation in order to find feasible solutions for scheduling real-time applications on multiprocessor systems. The genetic engine looks for a feasible task-to-processor allocation and deadline setting that meets the given real-time and dependency restrictions. The simulator is used to evaluate the behaviour and consequently the quality of different candidates. Through an iterative process, we obtain a feasible scheduling solution by choosing the best candidate result.

Our contribution addresses the adaptation of a genetic algorithm to the multiprocessor scheduling problem and specifically the definition of a multi-criteria fitness function that describes the quality of a schedule related to the imposed time restrictions. Through experiments, we determined the parameters of the genetic algorithm (e.g. mutation and crossover ratios, selection strategy, population set cardinality, etc.) that yield the best performance for the given setup. In order to reduce the search space and consequently the search time, we propose a heuristic for the generation of intermediate deadlines as an

Hangan A., Sebestyen G. and Vacariu L..
Multiprocessor Real-time Scheduling Using an Optimization-based Technique.
DOI: 10.5220/0005076202360243
In *Proceedings of the International Conference on Evolutionary Computation Theory and Applications* (ECTA-2014), pages 236-243
ISBN: 978-989-758-052-9

alternative for random mutations. We compared the pure genetic solution with the combined technique (genetic plus heuristic) in terms of success rate, solution fitness values and execution speed.

The rest of the paper is organized as follows. Related work is presented in section 2. In section 3, we describe the system model. The proposed scheduling technique is presented in section 4 and the proposed technique for reducing the search space in section 5. The experiments are described in detail in section 6. Section 7 concludes the paper.

2 RELATED WORK

Real-time scheduling on multiprocessor systems includes some important sub-problems: the allocation of tasks to processors and the assignment of task priorities, which consequently establishes the order of execution.

In transactional systems, that consider precedence restrictions between tasks, an additional problem is to establish priorities not only for the end of a transaction but also for the tasks contained in it. In systems that use EDF schedulers, the priority is given by the task's deadline. Usually, the intermediate task deadlines are not determined by the nature of the real-time application, but they may have an important impact on the schedulability of the system. Intermediate task deadlines are necessary to the local scheduling of tasks (on each individual processor). An important number of research works investigate the two scheduling sub-problems separately, or as a composite solution.

Task allocation in distributed real-time systems is known to be an NP-complete problem (Lupu, et al, 2010), so an algorithm that generates an optimal solution in polynomial time does not exist. To generate sub-optimal solutions for task allocation in polynomial time, one can use heuristics such as First Fit, Best Fit, Worst Fit or Next Fit. If the system workload is heavy, a feasible solution may not be found even if such a solution exists.

Solutions that address only the tasks' order of execution consider that task allocation to processors is already resolved. The most complex issues appear in the case of distributed applications, which are modeled as transactions or sequences of tasks with end-to-end deadlines. In this situation, intermediate tasks do not have predefined deadlines, so the only imposed restriction is that the last task of the sequence finishes its execution before the end-to-end deadline. To obtain a schedule, researchers proposed different algorithms and heuristics for assigning deadlines to intermediate tasks. In (Serreli, Lipari, and Bini, 2009) the authors investigate a deadline assignment that reduces resource utilization. They consider a component-based approach, analyzing each transaction individually, and use separate windows of execution for the tasks in a transaction. In (Di Natale and Stankovic, 1994) and (Kao and Garcia-Molina, 1997) the authors propose two similar heuristics for intermediate deadline assignment, which distribute the end-to-end deadline evenly or proportionally between all tasks. In (Gutierrez Garcia and Gonzalez Harbour, 1995) the authors use an iterative optimization algorithm to assign deadlines to the tasks inside a transaction set.

In the case of composite solutions, which solve both task allocation and priority assignment, optimization techniques such as simulated annealing (Tindell, Burns, and Wellings, 1992) or genetic algorithms (Azketa, et al, 2011 (2)), (Yoo and Gen, 2007) and (Samal, Mall, and Tripathy, 2014) can be used.

It is rather difficult to make a comparison between the existing scheduling techniques, as they use different system models, schedulability analysis methods and different metrics for performance evaluation. For example, (Lupu, et al, 2010) evaluate task allocation heuristics using a periodic task model, with independent tasks, under EDF and Fixed Task Priority (FTP) scheduling. In (Hladik, et al, 2008) the authors use periodic task models with inter-task communication and FTP scheduling, while in (Azketa, et al, 2011 (1)) and (Gutierrez Garcia and Gonzalez Harbour , 1995) the authors use linear transaction models with end-to-end deadlines under FTP scheduling. In (Oh and Wu, 2004), and (Yoo and Gen, 2007) the authors consider transactions described by directed acyclic graphs that contain non-preemptive tasks and communication costs, under FTP scheduling.

Compared to (Azketa, et al, 2011 (1)) and (Azketa, et al, 2011 (2)), our optimization technique uses different methods for solution representation and solution evaluation. In our case, the solution is given by a processor allocation and a deadline assignment setting for each task, which will determine a schedule. In (Azketa, et al, 2011 (1)), (Azketa, et al, 2011 (2)) and (Yoo and Gen, 2007) the authors consider a gene that contains both processor assignment and task priority parameters.

The range for intermediate deadlines is less restricted in our approach compared to the range used in (Serreli, Lipari, and Bini, 2009), which, in our opinion, will increase the chance of finding feasible solutions. Because we encode the deadline

as a distinct gene, we can explore different deadline assignments for a task, for the same processor allocation.

3 SYSTEM MODEL

Any generic approach of a scheduling problem requires a given system model composed of a platform, a workload and a scheduling model.

3.1 Workload Model

Distributed real-time applications are usually modeled as sets of transactions. We represent by means of a list the precedence dependency between tasks in a transaction. A real-time transaction has the following defining elements:

- *Task list* (L) – describes the dependencies between tasks and determines the execution order restrictions;
- *Period* (T) – the repetition period of a transaction;
- *Deadline* (D) – the time limit for a transaction, relative to its release time.

A task has the following parameters: e*xecution time* (C), d*eadline* (d), *CPU affinity*.

Transactions are released periodically in accordance with some external or functional requirements. A transaction instance contains task instances generically called jobs. The execution of a transaction starts with the execution of jobs that do not have precedence dependencies. A job is considered for scheduling only if its dependencies are solved (jobs that precede it are executed).

In case of a real application, only transaction deadlines are specified, intermediate task deadlines being unknown. However, the scheduling algorithm, in our case EDF, requires such deadlines in order to establish the execution priorities. One of the main goals of our research is to determine these intermediate deadlines in a way that all real-time, precedence and resource restrictions are satisfied.

Our workload model may be configured to represent independent sets of tasks as well as distributed applications, including network communication tasks.

3.2 Platform and Scheduling Models

The processing resources of a platform are modeled as a multiprocessor system $P=\{P_1, P_2, \ldots, P_m\}$ composed of processors and possibly network segments. This model can cover a wide range of system configurations that span from parallel systems to distributed ones.

We assume that each processing resource has its own scheduler. The scheduler chooses the job with the highest priority to be executed, at a certain point in time, on the processing resource. The priority is computed by the scheduling algorithm implemented in the scheduler. In our experiments, we used the EDF algorithm that assigns priorities to jobs according to their deadlines, so the job that has the closest deadline will have the highest priority. It is known that the EDF algorithm is optimal for single processor systems (Liu and Layland, 1973) and it can handle task set utilizations of up to 100% in the case of independent tasks.

4 OPTIMIZATION-BASED SCHEDULING TECHNIQUE

Our work addresses two important aspects of multiprocessor scheduling: task allocation to processors and intermediate task deadline assignment. We propose a composite solution to these problems by employing an optimization method.

The task allocation and deadline assignment problems generate a multidimensional solution space, which increases with the number of processors and the number of tasks in the system. As mentioned in section 2, finding an optimal solution is an NP-complete problem. However, sub-optimal solutions are acceptable in our case if transactions' end-to-end deadlines are not exceeded, even if some intermediate task deadlines are missed. Our objective is to find this type of sub-optimal scheduling solution.

We choose a genetic algorithm as search and optimization method, because it is well suited for problems with a large search space and multiple optimization objectives. A genetic algorithm starts with an initial set of possible solutions called population. At each iteration step, it generates a new population by means of natural selection, crossover and mutation. Each solution is evaluated with a fitness function. The fittest solutions will propagate their characteristics to later populations, generating improved new solutions.

We adapted the continuous genetic algorithm to our problem domain. Scheduling variables that must be optimized are task allocations to processors and task deadlines. The parameters that need to be minimized are transaction response times, task response times and the processor utilization factor.

We start from initial solutions obtained by applying known heuristics (Lupu, et al, 2010) for both task allocation to processors and intermediate task deadline assignment. We generate new populations mostly through crossover (with tournament selection), but also by keeping the best individuals from the previous population. After obtaining a new population, mutation is applied. The variables of the genetic algorithm are population size, crossover method and mutation probability. The fitness of a solution is evaluated through simulation. The simulator receives a workload model obtained from the individual representation created in the genetic algorithm, and creates the execution schedule using the platform and the scheduling predefined models. We simulate each configuration setup and we use the timing results obtained through simulation for computing the fitness function.

The optimization-based technique comprises three steps: the genetic algorithm generates a solution population; the solutions are evaluated through simulation; based on the evaluation, the genetic algorithm generates a new solution population, which replaces the previous one. These steps are iterated until a feasible solution is reached.

4.1 Genetic Representation

Solutions to the processor allocation and deadline assignment problems are represented as individuals in a population. Each individual (chromosome) is composed of a sequence of genes. A gene is an integer value that may represent a processor identifier on which the task is allocated, or a task deadline. Each gene has its own domain of values (Δ). For processor allocation, the domain is the task's CPU affinity list:

$$\Delta_P^{Task_j} = \{P_i | P_i \in CPUAffinityList_{Task_j}\} \quad (1)$$

In the case of task deadlines, the value domain is continuous between the execution time of the task and the largest possible deadline.

$$\Delta_d^{Task_j} = [d_j^{min}, d_j^{max}] \quad (2)$$

$$d_j^{min} = C_j \quad (3)$$

$$d_j^{max} = D_k - \sum C_l, Task_l \in Trans_k \quad (4)$$

A task is not released until all its predecessors finished their execution, so the assigned deadlines do not determine or enforce the precedence between tasks. Even though the transactions have hard deadlines, the tasks inside transactions have soft deadlines. This happens because even if several intermediate tasks miss their deadlines, it is still possible for the last task to meet its deadline. For this reason, we allow a less constrained value domain for genes that represent task deadlines. A less constrained deadline domain can increase the chances of finding a good scheduling solution.

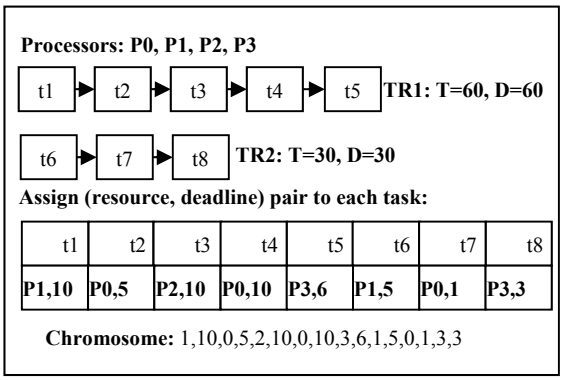

Figure 1: Chromosome construction based on the transaction model.

The chromosome contains two adjacent genes for each task in the workload. The first gene corresponds to the processor to which the task is allocated and the second is the task's deadline. The order of the genes is given by transactions. Figure 1 gives an example of how the chromosome is constructed based on the system model. The example is composed of four processors and two transactions (with the corresponding tasks). Tasks can be allocated to any of the four processors. For instance, the first gene shows that task t1 is allocated to processor P1 and its intermediate deadline is 10 time units. The next gene is for task t2 allocated on P0 and with deadline of 5 time units.

We generated the initial population in two steps. First, we created a small number of individuals using task allocation heuristics such as First Fit or Round Robin. We assigned the deadlines for tasks and messages choosing the smallest possible value as deadline. In the second step, we applied mutations to obtain new individuals. The second step is repeated until the specified population's size is reached (we considered a population of 60 individuals).

4.2 Genetic Operators

We used two types of genetic operators: crossover and mutation. These operators are applied on the current population with predefined probability rates, to create new generations with better fitness.

For our scheduling problem, we use a two-point

crossover operator. The parents are selected by tournament. From two randomly selected individuals, the fittest is chosen to be one of the parents. This way we give a chance to individuals that do not have the best fitness but may have good partial solutions to propagate their genes to later generations. The crossover points are selected at random. We eliminate cases when the points coincide or when they are at the two ends of the chromosome. A certain task can inherit the allocation gene from a parent and the deadline gene from the other parent, so it is not necessary that the genes between the crossover points include both allocation and deadline genes for a task.

We applied two different mutation operators. We mostly applied the classic mutation operator that selects a random gene from the chromosome and changes its value, with another random value from the gene's value domain. We also implemented a mutation operator, which chooses the new value from a sub-interval around the current value of the gene. The limited interval is set as a percentage of the maximum allowed interval.

Experimental results improved when the interval was limited to 50% of the initial domain and later to 25%. We applied this type of mutation only on populations with good average fitness (when the scheduling solution is close to being feasible), because we suppose that their genes are close to their best values and we do not want to spoil good possible solutions through mutations that radically change the gene's value.

4.3 The Fitness Function

The fitness function evaluates the quality of a given scheduling solution related with some chosen optimization objectives. A scheduling solution is considered the best if all transactions finish before their deadlines, all tasks finish their execution before their designated deadlines and if the tasks are uniformly allocated to the available processors. But, as we mentioned before, we only look for sub-optimal solutions, in which at least all transactions finish before their deadlines. For this purpose, we establish three optimization objectives, with different weights.

The most important optimization parameter is the transaction's completion time. A solution is considered feasible if the transaction's completion time is less or equal to the transaction's absolute deadline. Another optimization objective is to have a rather uniform allocation of tasks over the existing resources. This will increase the system's robustness and it can increase the effectiveness of the search. The third optimization objective is the task response time, which has to be less than the task's deadline. We differentiate between satisfying transactions deadlines and intermediate task deadlines in the sense that transaction deadlines are much more important and task deadlines are artificially introduced for the scheduling algorithm.

In order to express all the above optimization goals, we defined a fitness function as a weighted combination of three fitness expressions. A smaller value means a better solution.

The first component f_{TR} measures the quality related to the transaction's completion time (equation 5). It is computed as a sum of terms, each term representing a transaction that does not meet its deadline. A term is an exponential function of the difference between the maximum response time R_i and the deadline D_i of a transaction. The goal is to have all the differences equal to zero, which means that all transactions meet their deadlines.

$$f_{TR} = \sum_{R_i<D_i} 2^{max(R_i-D_i)} \quad (5)$$

The second component measures if the intermediate tasks meet their deadline (equation 6). This component has a smaller weight.

$$f_t = \sum_{tasks} 2^{\max(r_j-d_j)} \quad (6)$$

The next component measures the degree of uniform allocation of tasks on processors (equation 7). It is the sum of the differences between the actual processor's utilization factor U_p and an average value. The goal is to obtain an allocation as close as possible to the average value (equation 9).

$$f_{alloc} = \sum_{p\in P} \left|U_p - \bar{U}\right| \quad (7)$$

$$U_p = \sum_{tasks\in p} \frac{C_k}{d_k} \quad (8)$$

$$\bar{U} = \frac{\sum_{tasks} \frac{C_k}{d_k}}{card(P)} \quad (9)$$

C_k and d_k are the execution time and deadline of task k. Card(P) is the number of available processors.

Below is the complete expression of the fitness function:

$$F = f_{TR} * w_1 + f_{alloc} * w_2 + f_t * w_3 \quad (10)$$

Based on experiments, we found that a good combination of weights is: w_1 = 1000, w_2=100 and

w_3=1. The most important factor should receive the largest weight. With this weight configuration, at the beginning of the genetic search the first criterion is dominant, then in the middle part the second one is more active and in the final part the uniform task allocation criterion selects the best candidate.

5 A SEARCH SPACE REDUCTION TECHNIQUE

As the task allocation and deadline assignment problems generate very large solution spaces that increase with the number of processors and the number of tasks, the execution time of the genetic algorithm can be very large (e.g. hours). Moreover, as the solution space increases, the algorithm's success rate reduces because it doesn't always converge to an acceptable solution in an acceptable number of iterations, or it deadlocks (finds a local minimum).

Therefore, we investigated if a smaller search space has a good impact on the genetic algorithms success rate, its speed and the fitness of the best solutions. In this context, we propose a technique that composes our optimization-based approach for finding a task to processor allocation with a non-iterative approach for intermediate task deadline assignment.

The four steps for finding a scheduling solution are:

1. Create the chromosomes;
2. Apply a non-iterative algorithm or heuristic to assign deadlines to all tasks;
3. Compute the fitness;
4. The genetic algorithm chooses the best individuals and applies the genetic operators to obtain a new population.

The second, third and fourth steps are iterated until an acceptable solution is found. The genetic algorithm will create the chromosomes as described in section 4, but the genes that represent task deadlines will have constant values (determined in the second step) and will not be modified (mutated) during genetic iterations. We implemented two variants of the proposed technique, by using two distinct solutions for task deadline assignment. In the first variant, we used the algorithm proposed in (Serreli, Lipari, and Bini, 2009), where the authors find a deadline allocation by constructing an ordered list of tasks. The obtained deadline assignment depends on the distribution of computation times among the tasks and on task to processor allocation.

In the second variant, we propose a deadline assignment heuristic similar to the ones presented in (Kao and Garcia-Molina, 1997) and (Di Natale and Stankovic, 1994). We computed the transaction laxity time (l) and we divided it proportionally between the transaction's intermediate tasks. The deadline is computed as the execution time to which is added the portion of the transaction's laxity time (equation 12).

$$l = D - \sum_{Tasks} C_k \tag{11}$$

$$d = C + l * \frac{C}{\sum_{Tasks} C_k} \tag{12}$$

This deadline allocation heuristic is influenced by the transactions time parameters (deadline and execution time) and not by the task to processor allocation.

6 EVALUATION

For the experimental part, we developed a tool composed of a genetic engine linked to a real-time systems simulator, RTMultiSim (Hangan and Sebestyen, 2012). The genetic engine receives as input the application model composed of tasks and transactions. The transactions are translated into chromosomes. Afterwards, the genetic engine generates populations in search for better individuals. The simulator executes a given system model. During simulation, the maximum response times of tasks and transactions are determined. Based on these parameters, the simulation tool computes the fitness value of the individual, which is supplied to the genetic algorithm that continues the search with a new generation.

We evaluate the optimization-based scheduling technique in the case of distributed transaction scheduling. Experiments allowed us to adjust the parameters of the genetic algorithm for a faster generation of a good solution. In case of mutation probability the best values was 1% for shorter chromosomes (20-30 genes) and 0.6% for longer chromosomes (more than 100 genes), and 70% for the crossover probability. In some cases, a variable mutation ratio with a tendency of decreasing the probability in every generation had better results. The initial population was set to 60 individuals. A feasible solution has the fitness less than 1000. We generated two types of system models composed of 6 transactions on 4 processors (case A) and of 10 transactions on 8 processors (case B). We generated transaction sets with average system utilization of

60% to 99%. We obtained results from 3 sets of experiments: with the proposed optimization-based technique applied on both task allocation and deadline assignment sub-problems (OPT); with the optimization-based technique composed with the deadline assignment algorithm in (Serreli, Lipari, and Bini, 2009) (ORDER-OPT) and with the deadline assignment heuristic that divides the laxity time (equation 9) between tasks (LAX-OPT).

For the evaluation, we used the following metrics: *success rate*, a*verage number of iterations, average fitness*. Figure 2 shows the success rate of the three proposed optimization-based approaches as a function of system utilization (a description of the system load). As reference, we use a heuristic (not optimized) EDF scheduler that assigns deadlines with equation 9 and then allocates tasks to the first available processor (LAX-EDF). All three optimization-based algorithms have better success rate than LAX-EDF by a maximum of 30%. LAX-OPT has slightly better results than OPT, while ORDER-OPT has the worst performance. Figure 3 shows the average number of steps executed by the genetic algorithm until a feasible solution is found, as a function of system utilization (for case B).

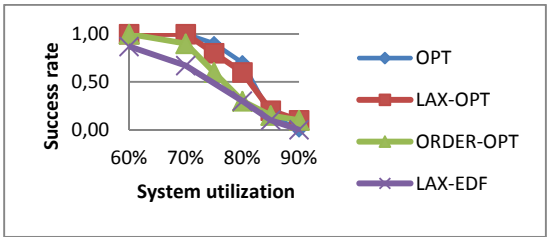

Figure 2: Case B - The success rates.

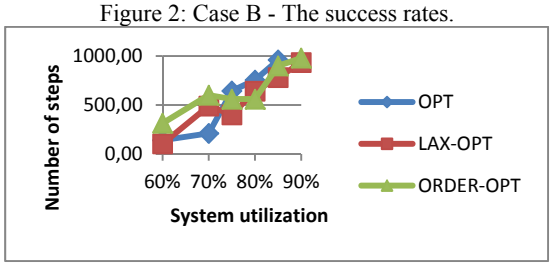

Figure 3: Case B - Average number of steps.

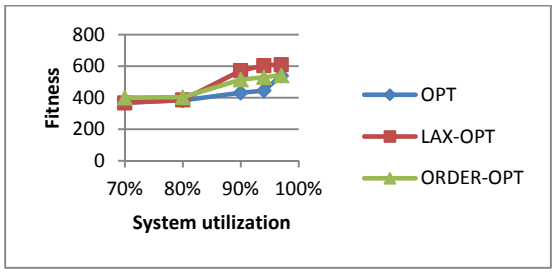

Figure 4: Case A - Average fitness.

OPT and LAX-OPT find feasible solutions in fewer steps. In case A, we observed that for 6 transactions OPT has the best results, but for utilizations greater than 90%, it finds less feasible solutions than LAX-OPT. Figure 4 shows the average fitness of feasible solutions found with our approach, as a function of system utilization for case A.

Table 1: OPT vs GA-Azketa.

Characteristics	OPT	GA-Azketa
Workload	Chain transactions	Chain transactions
Platform	Identical multiprocessor	Heterogeneous multiprocessor
Scheduler	EDF	FTP
Chromosome	Genes for processor allocation and task deadline	Gene for processor allocation, priority given by the gene's position.
Operators	Mutation, Crossover	Mutation, Crossover, Clustering
Fitness function	Minimize transaction response time and task response time; Uniform utilization	Minimize transaction response time and resource utilization
Fitness evaluation	Simulation	Holistic schedulability analysis

The solutions obtained by OPT have the best fitness. The solutions obtained by LAX-OPT have the worst fitness between utilizations of 80% and 97%, but, on the same interval, LAX-OPT has the highest success rate. In the case B the results are similar. Overall, the best average performance is obtained by OPT, but in some cases, LAX-OPT finds more (5-10%) feasible solutions than OPT.

We further evaluate the performance of our approach by making a performance comparison with related work. In (Azketa, et al, 2011 (1)) and (Azketa, et al, 2011 (2)) the authors solve a similar problem to ours by using a genetic algorithm. We show the differences between our approach and the one presented in (Azketa, et al, 2011 (2)) in Table 1. In (Azketa, et al, 2011 (1)) the authors make a statistical evaluation of their approach by randomly generating 2 types of transaction sets, small (6 transactions) and large (12 transactions). Their algorithm started failing at system loads of around 70% in the case of small systems. For large systems, their algorithm starts failing at loads of around 60%.

To make a comparison with our approach, we replicated their tests based on the description, as they do not provide any data sets. Our approach is better in terms of success rate, since it starts failing at over 70% systems loads, for large systems.

7 CONCLUSIONS

In this paper, we proposed an optimization-based technique for scheduling real-time transactions on multiprocessor systems. The technique implies the use of a genetic search engine and a simulator to find feasible solutions for mapping tasks to processors and for task deadline assignment. In order to reduce the solution search space, we combined our optimization-based approach for task allocation with non-iterative approaches for deadline assignment. We demonstrated through experiments that our approach improves non-iterative scheduling techniques by an approximate 30%. Compared to other similar techniques, our approach is at least 10% better in terms of scheduling success rate. Due to its flexibility, our solution may be used as a pragmatic off-line tool for allocating tasks on multiprocessor platforms and establishing time parameters for tasks in order to assure meeting global time restrictions.

ACKNOWLEDGEMENTS

This work was supported by a grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012.

REFERENCES

Azketa, E., Uribe, J., Marcos, M., Almeida, L., Javier Gutierrez, J., 2011 (1). Permutational genetic algorithm for fixed priority scheduling of distributed real-time systems aided by network segmentation *Proceedings of the 1st Workshop on Synthesis and Optimization Methods for Real-time Embedded Systems (SOMRES)*.

Azketa, E., Javier Gutierrez, J., Marcos, M., Almeida, L., 2011 (2). Permutational genetic algorithm for the optimized mapping and scheduling of tasks and messages in distributed real-time systems. *Proceedings of the 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, TRUSTCOM* .

Baruah, S., Cohen, N., Plaxton, G., Varvel, D., 1996. Proportionate progress: A notion of fairness in resource allocation. Algorithmica, vol. 15, no. 6, pp. 600–625.

Bertogna, M., Baruah, S., 2011. Tests for global EDF schedulability analysis. In Journal of Systems Architecture, no. 57, pp. 487–497.

Davis, R.I., Burns, A., 2009. A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems. In Techreport YCS-2009-443, University of York, Department of Computer Science.

Di Natale, M., Stankovic, J.A., 1994. Dynamic end-to-end guarantees in distributed real-time systems. *Proceeding of the 15th IEEE Real-Time Systems Symposium*, pp. 215-227.

Gutierrez Garcia, J.J., Gonzalez Harbour, M., 1995. Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems. *Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems*. IEEE Computer Society, pp. 124.

Hangan A., Sebestyen Gh., 2012, RTMultiSim: A versatile simulator for multiprocessor real-time systems, *Proceedings of The 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pp. 15

Kao, B., Garcia-Molina, H., 1997. Deadline assignment in a soft real-time system, IEEE Transactions on Parallel and Distributed Systems, vol. 8, no.12, pp. 1268-1274.

Liu, C.L., Layland, J.W., 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. In Journal of the ACM, vol. 20, no. 1, pp. 46-61.

Lupu, I., Courbin, P., George, L., Goossens, J., 2010. Multi-Criteria Evaluation of Partitioning Schemes for Real-Time Systems. *15th International conference on Emerging Technologies and Factory Automation, ETFA'2010*, Bilbao, Spain.

Oh, J., Wu, C., 2004. Genetic-algorithm-based real-time task scheduling with multiple goals. In Journal of Systems and Software, Vol. 71, Issue 3, pp.245-258.

Samal, A.K., Mall R., Tripathy C., 2014, Fault tolerant scheduling of hard real-time tasks on multiprocessor system using a hybrid genetic algorithm, Swarm and Evolutionary Computation vol.14, pp.92–105

Serreli, N., Lipari, G., Bini, E., 2009. Deadline assignment for component-based analysis of real-time transactions. *2nd Workshop on Compositional Real-Time Systems*, Washington, DC, USA.

Tindell, K., Burns, A., Wellings, A., 1992. Allocating Hard Real-TimeTasks: An NP-Hard Problem Made Easy. In Real-Time Systems, vol.4, no. 2, pp. 145-165.

Yoo, M., Gen, M., 2007. Scheduling algorithm for real-time tasks using multiobjective hybrid genetic algorithm in heterogeneous multiprocessors system. In Computers & Operations Research, vol. 34, no. 10, pp. 3084-3098.