Permutational genetic algorithm for fixed priority scheduling of distributed real-time systems aided by network segmentation

Ekain Azketa, Juan P. Uribe
Software Technologies
Ikerlan Research Center
Mondragón, Spain
{*eazketa, jpuribe*}*@ikerlan.es*

Marga Marcos
Systems Eng. and Automation
University of Basque Country
Bilbao, Spain
marga.marcos@ehu.es

Luís Almeida
Electrical and Computer Eng.
University of Porto
Porto, Portugal
lda@fe.up.pt

J. Javier Gutiérrez
Computers and Real-Time
University of Cantabria
Santander, Spain
gutierjj@unican.es

***Abstract*—The fixed priority scheduling of distributed real-time systems is an NP-hard problem, and therefore it is a suitable problem to be approached with generic search and optimization algorithms. On the other hand, the segmentation of the network can contribute positively to the schedulability of distributed real-time systems. This paper proposes a genetic algorithm with a permutational solution encoding that solves the holistic assignment of fixed priorities in distributed real-time systems aided by the optimized segmentation of the network.**

***Keywords*-distributed; real-time; holistic scheduling; priority; CAN segmentation; permutational genetic algorithm**

I. Introduction

The scheduling of distributed real-time systems is still an open NP-hard problem [1] for whose optimal solving no polynomial time method is known. However, it can be accomplished by means of generic search and optimization algorithms along with some strategies, such as the segmentation of the network, that can contribute positively.

This paper proposes a genetic algorithm with a permutational solution encoding for the assignment of fixed priorities to tasks and messages in distributed real-time systems. Our technique takes advantage of the network segmentation to facilitate the scheduling, and can be configured to minimize and/or meet constraints of both the schedule and the number of bridges used in the segmentation. Different genetic algorithms have been applied in the literature to make cyclic schedules, but as far as we know, they have not been used to assign fixed priorities in distributed real-time systems based on a holistic scheduling. On the other hand, we do not know any previous work that segments the network to ease the schedulability of distributed real-time systems.

This paper is structured as follows. Section II describes the system model. Then, Section III discusses related work. Section IV explains the genetic algorithm while Section V shows the experiments and results that validate the proposal. Finally, Section VI outlines conclusions and future work.

II. System Model

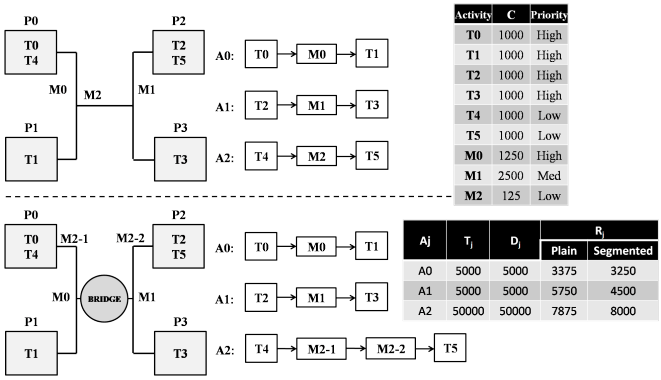
The physical architecture is composed of several processors $Ph \in P$ $(h = 0, 1, ..., Q_P)$ that implement preemptive fixed priority scheduling. The network has a baudrate WB and a maximum packet length L_{pck}. The packet has a payload length L_{pay} and cannot be preempted by the fixed priority scheduler of the network once its transmission has begun. Bridges $Bh \in B$ $(h = 0, 1, ..., Q_B)$ can be used to arbitrarily divide the network in segments $Sh \in S$ $(h = 0, 1, ..., Q_S)$. Subsection II-A exposes the advantages provided by the segmentation.

The logical architecture consists of one or several transactions $Aj \in A$ $(j = 0, 1, ..., Q_A)$ that have defined an activation period or a minimum interarrival time between activations T_j, as well as a deadline D_j. The transactions are composed of one or several tasks $Ti \in T$ $(i = 0, 1, ..., Q_T)$ that execute in the processors with worst-case execution times C_i and fixed priorities. Tasks with precedence relations exchange messages $Mi \in M$ $(i = 0, 1, ..., Q_M)$ that have lengths L_i and fixed priorities. A message cannot cross more than one bridge, i.e. at most two segments, but it can have a different priority in each one of them because a bridge can forward a message under another identifier. The network controller fragments each message in $\lceil L_i/L_{pay} \rceil$ packets, obtaining $\lceil L_i/L_{pay} \rceil - 1$ packets of length L_{pck} and one packet of length $(L_i \ mod \ L_{pay}) + L_{pck} - L_{pay}$. The complete length of a packetized message is $L_i^{pck} = L_{pck} \cdot \lceil L_i/L_{pay} \rceil - [L_{pay} - (L_i \ mod \ L_{pay})]$. The priorities of the tasks and messages are unknown in advance. When the priorities are assigned, all the packets inherit the priority of the message to which they belong.

A. CAN Segmentation

A bridge is a commutation device that connects multiple network segments at the data link layer of the OSI model. There are some COTS bridges for the CAN bus.

A bridge receives a packet completely and retransmits it through the other port immediately. However, the current lack of an appropriate schedulability analysis method for it requires us to make the pessimistic assumption that all the packets of a message are received in the bridge before starting their retransmission. On the other hand, each segment is treated as a different network in order to apply



Activity	C	Priority
T0	1000	High
T1	1000	High
T2	1000	High
T3	1000	High
T4	1000	Low
T5	1000	Low
M0	1250	High
M1	2500	Med
M2	125	Low

Aj	T_j	D_j	R_j	
			Plain	Segmented
A0	5000	5000	3375	3250
A1	5000	5000	5750	4500
A2	50000	50000	7875	8000

Figure 1. Top: plain bus. Bottom: segmented network

the available schedulability analysis techniques. Anyway, the bridge-based network segmentation can contribute positively to the schedulability of distributed real-time systems, as the following example shows.

Figure 1 shows two distributed real-time systems, one connected by a plain CAN bus and the other by a segmented CAN bus. Both systems are composed of 4 processors connected by a CAN network with $L_{pck} = 125$ bits, $L_{pay} = 64$ bits and $WB = 1$ Mbps. *A0* and *A1* have high communication requirements, since they send messages that need 10 and 20 CAN packets, respectively. The application has another transaction *A3*, and their periods, deadlines and worst-case response times are shown in bottom table of Figure 1. On the other hand, the column *C* of the top table shows the worst-case execution and transmission times of the tasks and messages, respectively. Since the blocking time in a network is produced by the longest packet of a smaller priority, it is 125 microseconds in this case.

In the plain CAN bus system, it can be seen by inspection that if *M1* is assigned the highest priority, *A0* does not meet its deadline, whereas if *M0* is assigned the highest priority -as in the table of the figure-, *A1* does not meet its deadline. In fact, there is no priority assignment that makes the plain CAN bus system schedulable. The CAN bus can be segmented in order to avoid the interference caused by the messages of a segment over the messages transmitted across another segment, e.g. in opposition to the plain CAN bus, in the segmented CAN network the message *M0* is not affected by the message *M1* and vice versa. Nevertheless, the real-time model of the transactions that cross the bridge have more messages, specifically one per segment, which may make them to have larger worst-case response times.

Using the priority assignment in the table of Figure 1 and computing the worst-case response times with the offset-based optimized technique [2], the response times of bottom table are obtained. As can be seen, the segmentation of the network allows to find a schedulable priority assignment because the response times of the transactions with exclusively intra-segment traffic are reduced.

III. Related Work

Some works approach only the priority assignment using heuristics [3] and genetic algorithms [4], [5]. Some other works approach the mapping and the scheduling of tasks and messages with methods such as simulated annealing [1], branch-and-bound [6], dedicated heuristics [7], linear programming [8], and genetic algorithms [9], [10]. However, either they do not apply a transactional model or use non-preemptive cyclic schedules. Other proposals combine the mapping and scheduling of tasks and messages with the network topology design using parallel recombinative simulated annealing [11], constructive heuristics [12], iterative heuristics [13], evolutionary algorithms [14], and binary linear programming [15]. However, either they do not apply a transactional model or use non-preemptive cyclic schedules. As can be seen, few proposals approach the fixed priority holistic scheduling of distributed transactional real-time systems, and none of them takes advantage of the network segmentation to ease the scheduling.

IV. Genetic Algorithm

A genetic algorithm [16] is a search and optimization metaheuristic based on evolving an initial population of candidate solutions to the problem, named individuals, towards better solutions through generations of populations by means of biologically inspired techniques such as inheritance, natural selection, crossover and mutation.

A search and optimization problem consists of assigning values to some variables in a way that all the restrictions are met and some function is maximized or minimized. In the present scheduling and network segmentation problem, the value assignment is done to the following variables: segment the network if necessary and map each processor to one of the segments, and assign each task and message a priority different from the priorities of the tasks and messages mapped in the same processor and segment, respectively. The restrictions are that the worst-case response time R_j of each transaction has to be less than or equal to the deadline D_j of the transaction, and that no more than the configured maximum number of bridges can be used. And finally, the factor that can be minimized is the number of bridges.

In this work a genetic algorithm with a permutational solution encoding is proposed to schedule fixed-priority based distributed real-time systems aided by the network segmentation. In the following lines, candidate solution encoding, initial population creation, fitness function, and crossover, mutation, correction and clustering operations are explained supported by Figure 2, which is based on the architecture of Figure 1.

A. Encoding

A candidate solution is a set of concrete values of the variables. In a genetic algorithm, a variable is encoded with an element called gene and a candidate solution is

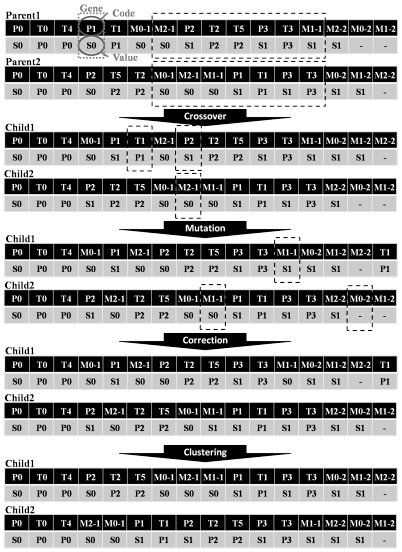


Figure 2. Crossover, mutation, correction and clustering operations

encoded with a string of genes called chromosome. This paper proposes a hybrid value-permutational representation based on the classical permutation representation [17] for encoding the candidate solutions.

The present problem consists of some variables representing the mapping of the processors in the segments, the priority of the tasks in the processors and the priority of the messages in the segments. Each variable of the problem has an associated gene with the structure shown in Figure 2. We define a gene that is composed of two fields. *Code* is a fixed field that stores the name of the variable (in this case Ph for processors, Ti for tasks and Mi for messages), which has to be unique for all the variables. *Value* is a potentially changing field that represents the mapping of the associated variable and whose value is one among the candidate values of the variable. Moreover, the relative position of the gene in the chromosome with respect to other genes with the same value denotes the priority of the associated variable, i.e. the more to the left, the higher the priority.

There are some variables that do not have priorities and hence only the value of their genes is relevant. This gene type is called *value gene* and the algorithm assigns it a concrete value among its candidates but not a concrete relative position. Each processor is represented by one value gene and its candidate values are the segments (Sh). The second type of variable has both relevant value and relative position, but the algorithm assigns its gene only a concrete relative position whereas the value either is fixed or depends on the network topology. This gene type is called *position gene*. Each task is represented by one position gene and each message by two. In the defined topology (Section II), a message can be transmitted through at most two segments and can have a different priority in each one of them. As the priority is represented by the relative position of the gene and a gene can only be in one position at each time, one gene is not enough to represent the two possible different priorities of a message. Hence, each message is represented by two position genes Mi*-1* and Mi*-2*. If the message is transmitted only through one segment, only the position of the Mi*-1* (sender and receiver segment) gene is relevant; and if it is transmitted through two segments, the position of Mi*-1* (sender segment) and Mi*-2* (receiver segment) genes are relevant.

B. Initial Population

The initial population is the set of candidate solutions that are evolved by the genetic algorithm. This paper proposes a initial population creation method composed of two phases. In the first phase, all the candidate solutions are created randomly and a clustering algorithm is applied over each one of them to segment the network and map the processors in segments. This clustering method traverses cyclically all the possible segments until the configured maximum number of them, and maps in each segment a randomly selected processor until all the processors are mapped in segments. This way, all the candidate solutions have the maximum number of segments, so their scheduling is likely to be easier. In the second phase, all the candidate solutions are pseudo-randomly scheduled by means of HOPA [3] heuristic. HOPA allows to configure two constant factors, k_a and k_r, that control the influence of the activities (tasks and messages) and resources (processors and networks), respectively, in the local deadline and priority assignment process. Different configurations of k_a and k_r values give different priority assignments even over the same architecture. This behaviour is used to find pseudorandom priority assignments for the initial candidate solutions by executing HOPA over each one of them with aleatory values of k_a and k_r.

C. Crossover

The genetic crossover operator combines information from two parent chromosomes to create two children chromosomes. This genetic algorithm uses the OX3 [17] crossover operator. OX3 chooses two cut-points randomly and the block of the genes of the first (second) parent between those two points is inherited directly by the first (second) child in the same absolute position. The genes not included in the inherited block are taken from the other parent chromosome in strict order.

D. Mutation

The mutation is an operator that maintains the genetic diversity of the population through random changes in the information of the genes. This genetic algorithm defines two mutation operators. The *position mutation* operator changes the position in the chromosome of a position gene, and hence the priority of the variable may be altered. The *value*

mutation operator changes the value of a value gene to another candidate value, and therefore the information about the mapping of the variable is altered.

E. Correction

During the crossover and mutation operations, processors may be changed to different segments. As a consequence, the messages sent or received by the tasks mapped in the changed processors have to be transmitted through the new segments. However, the mapping value of those messages reflect their segment before the change. This inconsistency is corrected by the correction operation.

F. Clustering

The clustering of genes is carried out with the aim of grouping the related genes side by side and creating a variable contiguous block per segment and processor. This operation increases the probability of the crossover operator to transmit to the children chromosomes blocks of variables whose relationship makes them more likely to belong with the current values to a solution. The clustering reorders the genes but always maintaining the relative order of task and message genes with respect to other genes with the same value, because otherwise the priority assignment done by the algorithm would be altered. First, the chromosome is clustered according to the value of the genes in ascending order of segments (*S0*, *S1*, etc.); each segment section is ordered first in ascending order of processors (*P0*, *P1*, *P2*, etc.) and then messages in that segment are clustered without altering the relative order between them. Into each processor section, tasks are ordered also strictly respecting the relative order between them.

G. Fitness Function

In a genetic algorithm the fitness function checks how well a candidate solution solves the problem. In this case, the fitness function is the weighted sum of the *time fitness* and the *segmentation fitness*.

Time fitness f_t (Equation (1)) evaluates how well the transactions meet their deadlines. The time fitness is the scheduling index factor [3] divided by the deadline. As can be seen, f_t is the total relative slack of the worst-case response times of the transactions with respect to their deadlines, divided by the total number of transactions. A relative scheduling index smaller than zero ($1-R_j/D_j < 0$) denotes the violation of a deadline. If there is at least one transaction with a relative scheduling index smaller than zero, the time fitness is the sum of only the relative scheduling indexes that are negative.

$$f_t = \begin{cases} \sum_{\forall Aj} \frac{1 - R_j/D_j}{card(A)}, & \text{if } \forall Aj : 1 - \frac{R_j}{D_j} \geq 0 \\ \sum_{\forall Aj} min\left[0, \frac{1 - R_j/D_j}{card(A)}\right], & \text{if } \exists Aj : 1 - \frac{R_j}{D_j} < 0 \end{cases} \quad (1)$$

The segmentation fitness f_s Equation (2) evaluates the utilization of the bridges used in the network segmentation. Being C_i^h the worst-case transmission time of the message Mi that cross the bridge Bh, the communication utilization in a bridge is $UBh = \sum_{\forall Mi\ in\ Bh} (C_i^h/T_i)$, and its average value is $\overline{UB} = \sum_{\forall Bh} (UBh/card(B))$, being $card(B)$ the number of elements in the set B. The numerator of the fraction in Equation (2) is the weighted sum of three factors. The first one increases the penalty if the corresponding bridge is used; the second one increases the penalty with the utilization level of the bridge; and the last factor reduces the penalty with the utilization deviation of the bridge, because a bigger deviation means that some bridges are closer from zero utilization, and hence closer from being dispensed. The sum of the weights has to be 1 and experiments have shown that good values are $w_u^1 = 0.6$, $w_u^2 = 0.3$ and $w_u^3 = 0.1$.

$$f_s = 1 - \sum_{\forall Bh} \frac{w_u^1 \cdot \lceil UBh \rceil + w_u^2 \cdot UBh + w_u^3 \cdot \left(1 - \left|UBh - \overline{UB}\right|\right)}{card\,(B)} \quad (2)$$

Equation (3) gives the total fitness F, which is computed by the weighted sum of f_t and f_s. A negative time fitness denotes the violation of a deadline and therefore it is an invalid solution. In this case, the total fitness is only the time fitness. The weights w_t and w_s have to sum 1.

$$F = \begin{cases} w_t \cdot f_t + w_s \cdot f_s, & \text{if } f_t \geq 0 \\ w_t \cdot f_t, & \text{if } f_t < 0 \end{cases} \quad (3)$$

H. Stop Condition

We define a optimization tendency function O_c that stops the genetic algorithm when the optimization process does not have the minimum rate required to find a valid solution within the configured maximum generations. Being g_{max} and g_c the maximum and current generation of the genetic algorithm, and f_t^i and f_t^c the time fitness of the best initial and current candidate solutions, the optimization tendency is $O_c = (g_{max}/g_c - 1) \cdot (1 - \left|f_t^i/f_t^c\right|)$. This factor is only used when $f_t^c < 0$ and the algorithm stops when $O_c < O_c^{min}$.

V. Experiments

Scenario: The experiments of the priority assignment and network segmentation problems are approached over four system types: small system with loose deadlines (SL), small system with tight deadlines (ST), large system with loose deadlines (LL), and large system with tight deadlines (LT). Some logical architectures are randomly generated for each one of those four types of systems using the following parameters. The number of processors Q_P and transactions Q_A is 4 and 6, respectively, in small systems, and 8 and 12, respectively, in large ones. The number of tasks in each transaction Q_T^{Aj} is $random[2, Q_P]$ and the number of messages $Q_M^{Aj} = Q_T^{Aj} - 1$. The worst-case execution time of the tasks C_i is $random[10, 50]$ ms. and the length of the messages L_i

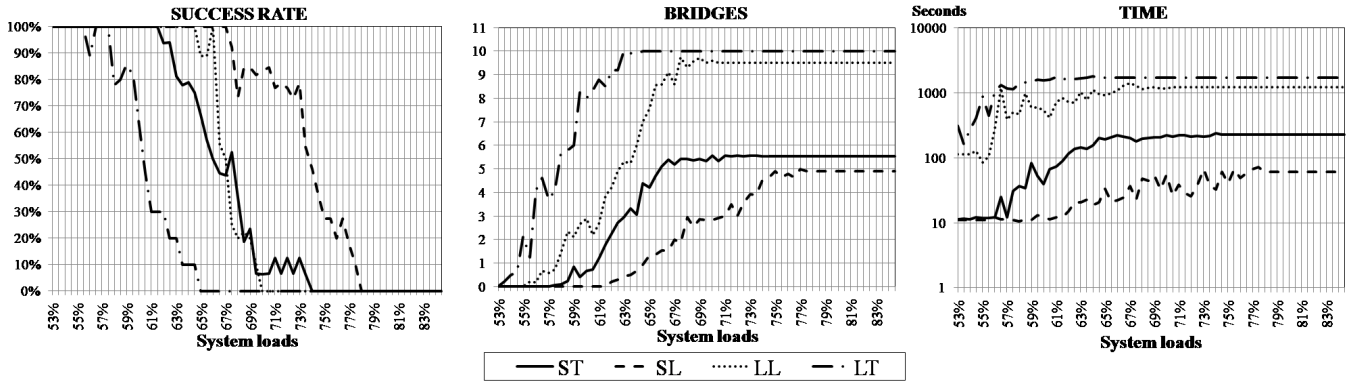


Figure 3. Average results of the experiments

is $random[1000, 5000]$ bits. The period of the transactions is $\left\lceil random\,[2.0, 4.0] \cdot \left(\sum_{Ti\ in\ A_j} C_i + \sum_{Mi\ in\ A_j} \frac{L_i^{pck}}{WB}\right)\right\rceil$ ms. and the deadline is $\left\lceil X \cdot \left(Q_T^{A_j} + Q_M^{A_j}\right) \cdot T_j\right\rceil$ ms., being $X = 0.5$ in systems with tight deadlines and $X = 1$ in systems with loose deadlines. The network parameters are $L_{pck} = 125$ bits, $L_{pay} = 64$ bits and $WB = 1000$ bits/ms. The mapping of tasks to processors is carried out in the following way: processors are iteratively traversed, and a task whose computation requirement fits in the current processor is randomly selected, repeating the steps until all the tasks are mapped.

Process: The average system load -*system load* now onwards- is computed adding the utilization of the processors and the network. It is computed as if it were only one segment, and each transmitted message is considered only once even though it crosses a bridge. In each random scenario, the initial system load is gradually rised by increasing the lengths of the messages a random number between $[2000, 2500]$ bits, and the genetic algorithm is executed with each one of those system loads, storing the obtained solutions. Each execution of the genetic algorithm is based on the following process. First, the maximum number of segments is configured to 1 and the algorithm starts searching schedulable priority assignments. If it stops before having found a valid solution, the maximum number of segments is reconfigured to 2 and the algorithm starts again. And so on, until obtaining a valid solution or reaching the maximum number of segments set by the user. With this method, the scheduling of the system is started over the potentially most optimized candidate solutions (1 segment and 0 bridge) and gradually loses optimality (adding bridges) if valid priority assignments are not found. The described technique allows to reduce the probability of not checking the fitness of a more optimal (fewer bridges) candidate solution.

Configuration: The genetic algorithm is configured with a population of 50 individuals. The initial population is pseudoaleatorily scheduled by executing HOPA a random integer number of iterations in the $[3, 5]$ interval and with a random rational number in the $(0, 5]$ interval selected independently for k_a and k_r for each individual. The maximum number of generations is 100. The minimum optimization tendency is $O_c^{min} = 0.8$. The selection method is a tournament of 2 individuals. Values between 0.7 and 1 for the crossover probability [9], [10] and between 0.001 and 0.05 for the mutation probability [10] are configurations that work well in almost any problem. We use 0.8 for the crossover probability and 0.005 for the mutation probability. Generational replacement with elitism of 1 individual is used. The fitness weights are configured as $w_t = 0.01$ and $w_s = 0.99$. This way, the minimization of the bridges becomes the main objective whenever the worst-case response times of the transactions meet their deadlines. On the other hand, segments can be added gradually to a maximum of 4 in the small scenarios and 5 in the large ones. The worst-case response times are computed with the holistic schedulability analysis [18] implementation of the MAST [19] tool suite, because it is a fast method that facilitates massive experiments. However, the elitist candidate solution is reanalyzed with the offset-based technique [2] to check with more accuracy whether it meets deadlines. The experiments are done on a dual-core Intel i5-650 processor running at 3.2GHz with 4GB of memory, multithreading the schedulability analysis.

Results: *Success Rate* graph of Figure 3 shows the average proportion of schedulable priority assignments found by the genetic algorithm. As the scenarios are larger and deadlines tighter, the success rate of the genetic algorithm decreases. Moreover, it can be seen that LL has a success rate of 100% with larger system loads than ST. It seems logic, since the greater network utilization of the large scenario is partially neutralized by the segmentation, which has less power to overcome the drawback of deadline tightness. Anyway, this graph shows that the genetic algorithm can find schedulable priority assignments in different distributed real-time systems. *Bridges* graphs of Figure 3 show the

average number of bridges that uses the genetic algorithm in each system load. As can be seen, the network segmentation allows to increase significantly the loads in which a distributed real-time system is schedulable, specially in LL scenarios. Note that in LL scenarios without bridges, the genetic algorithm has a success rate of 100% up to system loads of 55%, whereas the segmentation allows the genetic algorithm to keep the success rate of 100% up to system loads of 64.5%. This fact falls within the logic. On the one hand, large scenarios have more messages, i.e. greater utilization of the network, and consequently, the segmentation may provide them more noticeable benefits. On the other hand, the segmentation may sometimes make some transactions to have larger worst-case response times, so the fulfillment of the time constraints may be easier if they have loose deadlines. *Time* graph of Figure 3 shows the average execution times of the genetic algorithm in a logarithmic scale. As can be seen, the times are reasonable for a complex system design process.

VI. Conclusions and Future Work

This paper proposes a genetic algorithm with a permutational solution encoding that solves the holistic priority-based scheduling aided by the optimized network segmentation of distributed real-time systems. The experimental results show that our genetic algorithm can find good solutions for complex distributed real-time architectures in reasonable times. Further, this genetic algorithm could also be used for the simultaneous mapping and scheduling of tasks and messages by adding more candidate values to the genes and including some weighted factors to the fitness function.

References

[1] K. Tindell, A. Burns, and A. Wellings, "Allocating hard real-time tasks: an NP-hard problem made easy," *Real-Time Systems*, vol. 4, no. 2, pp. 145–165, 1992.

[2] J. Carlos Palencia and M. González Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proceedings of the 19th IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 26–37.

[3] J. Javier Gutiérrez and M. González Harbour, "Optimized priority assignment for tasks and messages in distributed hard real-time systems," in *Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems*. IEEE Computer Society, 1995, pp. 124–132.

[4] S. Samii, Y. Yin, Z. Peng, P. Eles, and Y. Zhang, "Immune genetic algorithms for optimization of task priorities and flexray frame identifiers," in *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2009, pp. 486–493.

[5] E. Azketa, J. P. Uribe, M. Marcos, L. Almeida, and J. Javier Gutiérrez, "Permutational genetic algorithm for the optimized assignment of priorities to tasks and messages in distributed real-time systems," in *8th IEEE International Conference on Embedded Software and Systems*, 2011.

[6] M. Richard, P. Richard, and F. Cottet, "Allocating and scheduling tasks in multiple fieldbus real-time systems," in *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1, sept. 2003, pp. 137–144.

[7] P. Pop, P. Eles, Z. Peng, and T. Pop, "Analysis and optimization of distributed real-time embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, pp. 593–625, June 2004.

[8] W. Zheng, Q. Zhu, M. D. Natale, and A. S. Vincentelli, "Definition of task allocation and priority assignment in hard real-time distributed systems," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, 2007, pp. 161–170.

[9] Y. Monnier, J.-P. Beauvais, and A.-M. Deplanche, "A genetic algorithm for scheduling tasks in a real-time distributed system," in *Proceedings of the 24th Euromicro Conference*, vol. 2, aug 1998, pp. 708 –714 vol.2.

[10] J. Oh and C. Wu, "Genetic-algorithm-based real-time task scheduling with multiple goals," *Journal of Systems and Software*, vol. 71, no. 3, pp. 245–258, 2004.

[11] R. Dick and N. Jha, "MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 920–935, 2002.

[12] B. Dave, G. Lakshminarayana, and N. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 7, no. 1, pp. 92–104, 2002.

[13] H. Oh and S. Ha, "Hardware-software cosynthesis of multi-mode multi-task embedded systems with real-time constraints," in *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, 2002, pp. 133 –138.

[14] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, "Mapping applications to tiled multiprocessor embedded systems," in *Seventh International Conference on Application of Concurrency to System Design*, 2007, pp. 29–40.

[15] M. Glaß, M. Lukasiewyc, J. Teich, U. Bordoloi, and S. Chakraborty, "Designing heterogeneous ECU networks via compact architecture encoding and hybrid timing analysis," in *46th ACM/IEEE Design Automation Conference*, 2009, pp. 43–46.

[16] J. Holland, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–72, 1992.

[17] L. Davis, *Handbook of genetic algorithms*. Arden Shakespeare, 1991.

[18] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and microprogramming*, vol. 40, no. 2-3, pp. 117–134, 1994.

[19] M. González Harbour, J. Javier Gutiérrez, J. Carlos Palencia, and J. M. Drake, "Mast: Modeling and analysis suite for real time applications," in *Proceedings of 13th Euromicro Conference on Real-Time Systems*. IEEE Computer Society, 2001, pp. 125–134.