A Multi Variable Optimization Approach for the Design of Integrated Dependable Real-Time Embedded Systems[*]

Shariful Islam and Neeraj Suri

Department of CS
TU Darmstadt, Germany
{ripon,suri}@cs.tu-darmstadt.de

Abstract. Embedded systems increasingly encompass both dependability and responsiveness requirements. While sophisticated techniques exist, on a discrete basis, for both *dependability/fault-tolerance* (FT) and *real-time* (RT), the composite considerations for FT+RT are still evolving. Obviously the different objectives needed for FT and RT make composite optimization hard. In this paper, the proposed *Multi Variable Optimization* (MVO) process develops integrated FT+RT considerations. We introduce dependability as an initial optimization criteria by confining error propagation probability, i.e., limiting the interactions. Subsequently, quantification of interactions together with RT optimization by minimizing scheduling length is developed. A simulated annealing approach is utilized to find optimized solutions. We provide experimental results for our approach, showing significant design improvements over contemporary analytical initial feasibility solutions.

1 Introduction and Paper Objectives

Embedded real-time systems with implications on system dependability[1] are being employed in diverse applications such as flight, drive and process control. More and more functionality is being integrated into such systems, invariably leading to a heterogeneous environment consisting of applications of different *criticality* (both safety critical (SC) and non-SC), each with associated *responsiveness* requirements. Each application introduces system level constraints such as software (SW) complexity, cost, space, weight, power and multiple other realization constraints making the overall system composition a complex resource optimization task.

Thus, efficient system design strategies are needed to integrate these diverse applications across limited hardware (HW) resources while considering the interplay of fault-tolerance (FT) and real-time (RT) objectives. *Mapping* of mixed criticality/responsiveness applications onto shared resources is a crucial step for

[*] This work has been partly supported by the EU IST FP6 DECOS.
[1] The terms Dependability and FT will be used synonymously in the paper.

such a system design strategy. A mapping is defined as: *(i)* assignment of jobs[2] to suitable HW nodes such that platform resource constraints and dependability requirements are met (*resource allocation*) and *(ii)* ordering job executions in time (*scheduling*). This design step faces new challenges under resource constraints and needs careful attention such that FT and RT requirements are not compromised. Moreover, design optimization involves simultaneous consideration of several incompatible and often conflicting objectives.

Recently, bi-criteria objectives have attracted attention by researchers, e.g., in [1],[2,3]. The first two papers consider the trade-off between system reliability and scheduling length, while the third considers minimization of energy consumption using checkpointing to recover from faults. The complexity of the design endeavor becomes apparent when considering the huge design space of possible solutions on one hand, and the many competing design objectives on the other [4]. Overall, better design methodologies are needed to handle such complex problems.

We propose a generic optimization framework considering different design *variables*[3] both from Dependability/FT and RT perspectives. The approach is called *Multi Variable Optimization (MVO)*, which takes into account the satisfaction of various constraints as well as optimization of multiple competing variables. Since dependability is a primary design objective for safety-critical systems, the proposed framework puts emphasis on FT through *replication* of highly critical jobs. Dependability is then enhanced by providing error-containment mechanisms. If an error is present in a job, it is possible for this error to propagate to other jobs and cause multiple failures[4]. In ultra-dependable systems even a very small correlation of failures of the replicated units can have a significant impact on the overall dependability [5]. Thus, highly interacting jobs are assigned onto the same node, to prevent the spread of errors across HW nodes, i.e., enhance dependability by *error confinement* [6]. We also minimize the scheduling length while satisfying job precedence and deadline constraints and minimize the utilization of the network.

The contributions of our work are: *(i)* development of a generic framework which systematically guides the optimized system level design, *(ii)* quantification of application *interactions* and techniques to constrain the propagation of errors, *(iii)* combining interactions with scheduling length and bandwidth utilization that enables us to solve the MVO problem and *(iv)* application of an existing optimization algorithm within the approach, enabling a quantitative evaluation. For a representative target study, our evaluation shows significant design improvements for the considered variables. From a RT perspective, minimizing the scheduling length also gives the basis for maximizing the CPU utilization.

The paper is organized as follows. Section 2 discusses the related work. System models and problem statement are introduced in Section 3. Section 4 presents

[2] Applications are further decomposed into smaller executable units called jobs.

[3] A single variable refers to optimization of a single objective.

[4] The failure of a module (can be an application, a job or a node) due to the failure of another module is called a cascading failure.

the generic MVO framework and Section 5 provides the quantification of the variables. For the evaluation of the approach, we employ simulated annealing described in Section 6. The experimental evaluation and results are given in Section 7. Section 8 concludes the paper.

2 Related Work

Usually, FT is applied to an existing scheduling principle such as rate-monotonic or static off-line either by using task replication [7] or task re-execution [8],[9]. Satisfying RT constraints, in [10], the authors minimize the total completion and communication times. Maximization of the probability of meeting job deadlines is considered in [11]. A scheduling approach for distributed static systems is presented in [12], where the authors minimize the jitter for periodic tasks using simulated annealing. Satisfaction of multiple constraints (timing, dependability) and optimization of a single variable (bandwidth utilization) is presented in [13]. All these approaches either consider dependability as constraint or optimize a selected operational variable from a RT perspective.

Though optimizing one variable is straightforward, optimization of multiple variables is considerably more difficult. Several mapping techniques exist but few are concerned with optimizing dependability/FT and RT issues together. In [14], the authors propose that the combination of active replication and re-execution can provide an optimized design from the scheduling length point of view. In [15], the authors discuss multiple objectives such as minimizing communication, load balancing and minimizing the maximum lateness. [6] specifically addresses dependability (focuses on minimizing interaction) and presents heuristics for conducting the mapping. However, the focus is to aid integration between design stage SW objects. Overall, in design optimization, there is a dearth of work that addresses dependability as an optimization criterion. Commonly, dependability is considered directly as a constraint to be satisfied. Instead, we consider dependability and RT, both as constraints and as optimization criteria.

3 System Model and Problem Statement

Our system design framework is based on the following models: the SW and HW models, constraints model and the fault model. The SW model describes the functional and *extra-functional* (dependability, responsiveness, etc.) requirements of jobs and the HW model is the physical execution platform for those jobs. The fault model depicts the types of faults and their causes, whereas constraints restrict the possible solutions. The rest of this section details various important aspects and characteristics of the different models.

SW Model: The SW model consists of applications of varied criticality. Applications are further decomposed into a set of jobs $(j_1, j_2, ..., j_n)$. A *job* represents the smallest executable SW fragment, with basic communication capabilities for exchanging information with other jobs. We consider a job to have specific properties as required inputs to the mapping process, namely: *(i)* job name - each job has a unique name; *(ii)* timing requirements (earliest start time-EST, computation time-CT, deadline-D); *(iii)* volume of data for inter job communication in terms of bytes; and *(iv)* dependability/FT requirements - the degree of replication dc_i necessary for the i^{th} job to provide the required level of FT. dc_i is specified by the system user.

HW Model: We assume a network topology allowing every HW node to communicate with all other nodes $(n_1, n_2, ..., n_k)$. A HW node is a self-contained computational element (single- or multiprocessor) connected to the network (e.g., using a bus topology) through a communication controller. We assume that the computing basis of all node processors is similar. HW nodes may also contain additional resources, e.g., sensors, actuators etc. The communication controller controls the exchange of messages with other nodes.

Constraints Model: Constraints define the conditions that limit the possible mappings from a dependability, RT or resource perspective. A set of constraints need to be satisfied for a mapping to be valid [16]. We consider the following constraints: *(i)* binding constraints - jobs that need to be allocated onto specific nodes due to the need of certain resources (e.g., sensors or actuators), *(ii)* FT constraints - separation of replicas to different nodes, *(iii)* schedulability - maintaining RT constraints and *(iv)* computing constraints - such as the amount of memory available for jobs.

Fault Model: We consider both SW and HW faults, therefore a fault can occur in any job, HW node or communication link. The consequence of a fault is an error which can propagate from a source module to a target module via a corrupted message or via a shared resource. In the case of communication links, only transient faults are considered. A single fault, either a transient or a crash [17] impacting a shared resource, is likely to affect several or all of the jobs or replicas running on that node.

Problem Statement: The set of all possible mappings for a given set of jobs and nodes is called the *problem space (X)*, shown in Figure 1. A mapping is either feasible or infeasible. A feasible mapping is a solution which satisfies all constraints. If some constraint is not satisfied, the mapping is infeasible. A *point* x in the problem space X represents a mapping of jobs onto nodes. The *neighborhood* space $N(x) \subseteq X$ of a point x is the set of all points that are reachable by performing a *move* operation (e.g., relocating a job to a different node). We employ a transformation operator (Γ) to perform move operations (see Section 6.3 for details). The *value* of a point is a measure of the suitability of the mapping represented by that point. The function $f(x)$ is used to measure the value of a point of the problem space. For an optimization problem, which minimizes the value of variables (v), good mappings have low values. Hence, the task is to find a mapping $x^* \in X$ with the lowest function value for multiple variables, i.e., $f(x^*) \leq f(x)\ \forall x \in X$. x^* is the best mapping from a global search space (X).

In this work, a feasible mapping is provided as an input to the algorithm and feasibility is maintained throughout the quest by an external function call, therefore the problem space remains in the feasible region $X^{'} \in X$ (set of all

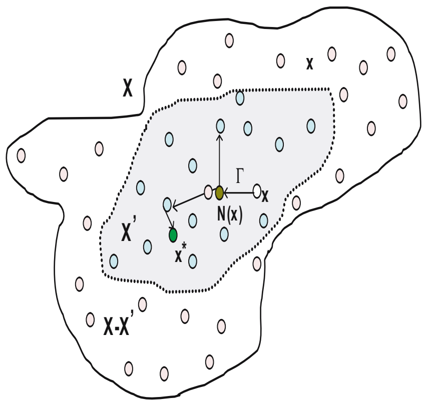

Fig. 1. Search space for the mapping

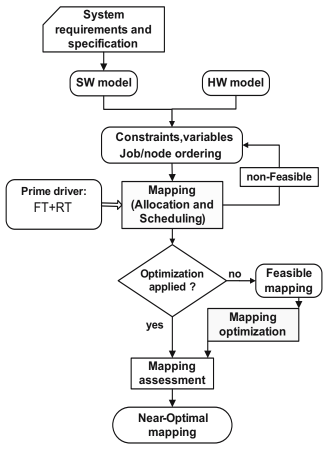

Fig. 2. System design optimization

admissible solutions), which reduces the search space considerably. We strive for finding the mapping $x^* \in X^{'}$, where $f(x^*) \leq f(x)\ \forall x \in X^{'}$. We use an MVO function ($MVO(v)$) to represent the mapping (see Section 6.1 for details).

4 The MVO Approach

The proposed MVO framework systematically guides the FT+RT driven mapping towards an optimized solution. In this section we discuss this generic design framework on the basis of the models presented in the previous section. The system design optimization flow and the corresponding steps are depicted in Figure 2 and in Algorithm 1 respectively. The design process starts with characterizing the SW and HW model. The properties of the model are extracted from the system requirements and specification document. In *Step* 2 (Algorithm 1), constraints are modeled, which need to be satisfied during the mapping. Design variables are defined in *Step* 3, which are employed in the *mapping optimization* phase shown in Figure 2. Variables are used for capturing the design criteria and they strongly depend on the objectives of the system design and on the considered system model [18].

Mapping algorithms need heuristics to achieve good performance. Of particular importance are job ordering and node ordering that decide which job to assign next and what node to assign that job onto [16]. Job and node ordering are described in *Step* 4. A crucial issue that arises at this design stage is the mapping of jobs onto suitable nodes. An initial mapping is created in *Step* 5 where allocation and scheduling is performed off-line in the early design phase. The result of this step is a feasible mapping. However, this mapping is likely to be very inefficient from a system design perspective. The purpose of the rest of the steps is to find a better mapping by using the proposed optimization framework. A candidate mapping from the set of possible solutions is generated in *Step* 6.

Algorithm 1. Generic framework for system level design optimization

```
1: derive the system model
2: extract design constraints
3: define design variables
4: ordering of jobs and nodes
5: generate an initial current mapping - apply heuristics
6: generate candidate mapping - exploring neighborhoods
7: a) compare candidate mapping with the current mapping
   b) go back to Step 6 until stoping criteria is met
8: define minimum requirements to select the mapping (the aspiration values)
9: assess the mapping and return the good mapping (a near-optimal one)
```

In order to select better designs, the candidate mapping is compared with the current mapping in *Step* 7. If a better mapping is found, the current mapping is updated. This step is iterative so that the comparison can be made with all the possible solutions. A detailed description of *Step* 6 and *Step* 7 is also given in Section 6.2. In *Step* 9, the mapping is assessed to ensure that it satisfies the minimum system requirements defined in *Step* 8. Essentially, we are interested in finding a near-optimal mapping meeting FT+RT design objectives.

5 Quantification of Design Variables

In this section we *quantify* the set of variables. This includes how to estimate variables, and how to formulate them in terms of function minimization. The primary objective is to enhance dependability by design, where our focus is to minimize interactions, i.e., to confine the propagation of errors between nodes. The second and third considerations are the scheduling length and the bandwidth utilization respectively that are important in terms of resource utilization and consequently lead to designs with lower cost. In the subsequent sections, we provide details of the variables used to quantify these objectives.

5.1 Interactions

Interaction is the probability of error propagation from a source to a target. This variable refers to how well the errors are contained within a single node. Low interaction values between nodes implies good error containment. Assigning highly interacting jobs on the same node reduces the error propagation probability across nodes. Below we describe two potential ways in which interactions between a source and a target could take place.

Case 1: Errors occur in the source and propagate to the target via message passing or shared resources. If a job is affected by an error of the node it is running on, it might propagate errors to jobs on other nodes with which it communicates or shares a resource. Such interactions risk the failure of multiple nodes and are undesirable.

Case 2: Messages sent over the network can be lost or erroneous due to transmission errors. Erroneous messages can propagate to different nodes and may cause unexpected behavior.

Estimating interactions: The interactions as shown in Figure 3 consists of three phases, namely: *(1)* an error occurring in a module or in a communication link, *(2)* propagation of the error to another module and *(3)* the propagating error causing a cascaded error in the target module. In order to measure interactions, let's assume P_e is the probability of error propagation from source to target considering no corruption over the network and P_l is the probability of message corruption over the network. The probability of error propagation from a source (s) to a target (t) is denoted by $P_{s,t}$ and defined as follows: $P_{s,t} = p\{error\ propagation|no\ corruption\ over\ the\ network\}\cdot p\{no\ corruption\ over\ the\ network\} = P_e \cdot (1 - P_l)$, where, $P_e = P_s \cdot P_t$. The probability that s outputs an error and sends it to the input of t is P_s and P_t is the probability that an error occur in t due to the error received from s. P_s indicates how often s allows errors to propagate and P_t indicates how vulnerable t is to errors propagating from s. Considering both $P_{s,t}$ and P_l, the interaction is calculated as follows: $I_{s,t} = P_{s,t} + P_l$.

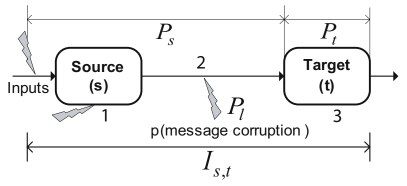

Fig. 3. Error propagation

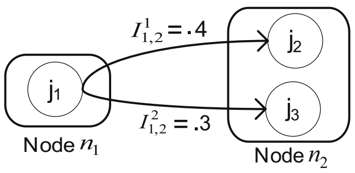

Fig. 4. Combining interactions

$I^o_{s,t}$ is the overall interactions between the set of jobs assigned together on a node and interacting jobs allocated on different nodes, which is expressed by the following equation:

$$I^o_{s,t} = 1 - \prod_{\rho}(1 - I^{\rho}_{s,t}) \tag{1}$$

Where ρ is the number of interactions paths between two nodes. For example, the overall interaction of node n_1 to n_2 as shown in Figure 4, will be $I^o_{n_1,n_2} = 1 - [(1-0.4)\cdot(1-0.3)] = 0.42$. Interactions are assumed to be zero for jobs which are assigned on the same node. However, it is not possible to assign all interacting jobs onto a single node due to the constraints. Also replicas need to be placed on different nodes which might have interactions with other jobs. Hence, there will be jobs interacting across nodes. We strive to minimize these interactions as much as possible for a mapping, such that dependability is enhanced by design. Values for error occurrence probabilities can be obtained, for example, from field data, by fault injection or from system specification [19]. The computation of the system level interactions $\hat{I}$ is expressed as follows, where k is the number of nodes:

$$\hat{I} = \sum_{i,j=1}^{k} I^o_{i,j} \tag{2}$$

5.2 Scheduling Length

This variable represents the total completion and communication time for a set of jobs on a node. As we use replication as the FT scheme, this results in more jobs needed to be scheduled and this naturally incurs a overhead on scheduling. The goal is to minimize overall scheduling length ($\hat{S}_l$) on a node satisfying precedence and deadline constraints. Minimizing scheduling length is important from the viewpoint of the uses of a set of processors, since it leads to maximization of the processor utilization. In every scheduling, gap may remain between two consecutive jobs executing on the same node due to precedence relations and communication cost. We define this gap as in-between slack (IBS). Slack can be used for future upgrading of jobs and also for energy savings.

We have developed a schedulability analysis in our previous work [16] and we employ that strategy in this work as well. The scheduling length for a candidate mapping is calculated using the following equation:

$$\hat{S}_l = \forall k \; max \left[\sum_{i,j=1}^{n} (M_{i,k} \cdot CT_{i,k} + IBS_{i,j}) \right] \tag{3}$$

where, n is the number of jobs, $CT_{i,k}$ is the computation time of the i^{th} job in the k^{th} node, $IBS_{i,j} = EST_j - LET_i$, where i is the job executed before j on the same node, and $M_{i,k} = 1$, if j_i is assigned to the k^{th} node and 0 otherwise. LET_i is the latest ending time of job i.

5.3 Bandwidth Utilization

In an integrated system design, jobs of different criticality and from different applications may be assigned onto a single node and jobs from a single application may be assigned onto different nodes. Therefore, good utilization of shared communication links is necessary. Bandwidth utilization ($\hat{B}_w$) is the ratio between the total bandwidth required by the system and the available bandwidth of the network (B_T) defined as follows:

$$\hat{B}_w = \sum_{i,j=1}^{k} b_{i,j}/B_T \tag{4}$$

where k is the number of nodes and $b_{i,j}$ is the total bandwidth requirements in terms of message size between nodes i and j. Minimizing this variable may allow for the use of a slower but cheaper data communications bus [15].

6 The Algorithm - Employing MVO

In the prior sections, we described the MVO framework and the quantification of considered variables. Next, we apply an existing optimization algorithm within our framework. For this purpose we have chosen simulated annealing (SA) [20].

SA is an algorithm, which converges to the global minima while solving an MVO problem (MVO-SA). SA is a long established effective metaheuristic with an explicit schema for avoiding local minima [12],[13],[20],[21]. Alternative approaches such as Genetic algorithm, Tabu search [21] were also investigated as options. However, the global minima possibility with SA makes it attractive. The overall optimization process is shown in Algorithm 2, which differs from usual single objective SA. We have adapted SA for multiple objectives, which returns the best values of variables together with the best mapping found so far.

6.1 The MVO Function

$MVO(v)$ is a function, which returns a natural number that corresponds to the overall quality of a given mapping. We construct the $MVO(v)$ function as a weighted sum of the variables, which is a widely used method for this class of problem [18],[22]. The value of the function is determined by using the values of variables $\hat{I}, \hat{S}_l,, \hat{B}_w$ and the trade-off factors ψ_i, ψ_s and ψ_b.

$$MVO(v) = \psi_i \cdot \hat{I} + \psi_s \cdot \hat{S}_l + \psi_b \cdot \hat{B}_w \tag{5}$$

The individual values of the variables are represented in a matrix form: $M[v] \equiv M[\hat{I}, \hat{S}_l, \hat{B}_w]$. After performing a move, the function is denoted as $MVO(v^{'})$ and the matrix as $M[v^{'}]$.

6.2 Application of SA

The MVO-SA algorithm requires the following inputs: *(i)* the set of jobs and nodes including their properties to create the initial mapping, *(ii)* variables, the $MVO(v)$ function and the trade-off factors, *(iii)* the Γ operator to change the mapping and *(iv)* SA parameters - initial temperature, the cooling schedule and the cooling factor for lowering the temperature. The output of the algorithm represents the optimized mapping of jobs onto nodes.

After setting the initial heating temperature T_h (Algorithm 2), the initial feasible mapping is created. The feasibility of the mapping is maintained throughout the search by an external function call, i.e., the best feasible mapping is sought. The values of all variables are set in the MVO function (Equation 5) and $MVO(v)$ is computed in *Step* 4. In order to generate the candidate mapping, neighborhoods are explored in *Step* 6. We apply the *transformation operator (Γ)* to explore neighborhoods. While applying this operator the feasibility of the mapping is checked. In *Step* 9, the candidate mapping $MVO(v')$ is evaluated in order to compare it with the current best mapping. If the difference $\delta v = MVO(v') - MVO(v)$ is less than zero (minimization) then we choose the candidate mapping. If δv is greater or equal to zero, then the candidate mapping is accepted with a certain probability, called the acceptance probability (a_p). One of the commonly used acceptance probability functions is $a_p = e^{-\delta v / T_h}$ [12],[13].

The technique used by SA to not get stuck at a *local optima* is to accept some worse moves as the search progresses. For larger δv, i.e., when the candidate mapping is extremely undesirable, the probability of acceptance diminishes.

Algorithm 2. MVO algorithm - SA based

```
1: initialization $T$; heating temperature $T_h$
2: generate an initial mapping
3: create the matrix $M[v] \equiv M[\hat{I}, \hat{S}_l, \hat{B}_w]$ for this mapping
4: evaluate the initial mapping $MVO(v)$
5: repeat
6:    explore neighborhood of the current mapping using $\Gamma$
7:    generate candidate mapping
8:    create matrix $M[v'] \equiv M[\hat{I}', \hat{S}'_l, \hat{B}'_w]$
9:    evaluate candidate mapping $MVO(v')$ for the new matrix
10:   calculate $\delta v = MVO(v') - MVO(v)$
11:   if $\delta v < 0$ then
12:      $M[v] = M[v']$ and $MVO(v) = MVO(v')$
13:   else
14:      calculate acceptance probability $a_p = e^{-\delta v/T_h}$ and
         generate $r = random[0, 1]$
15:      if $a_p \geq r$ then
16:         $M[v] = M[v']$ and $MVO(v) = MVO(v')$
17:      else
18:         restore the current mapping, i.e., keep $M[v]$ and $MVO(v)$.
19:      end if
20:   end if
21:   reduce the temperature $T_h$ by using a cooling schedule $T_{h-1} = c_f \cdot T_h$.
22: until some stopping criterion is met
23: return the best matrix M[v] and corresponding mapping $MVO(v)$
```

The initial temperature (T) is set to a sufficiently high value to accept the first few candidate mappings. However, the a_p decreases as T_h decreases. If an acceptance criteria is met, the candidate mapping is chosen, otherwise the current mapping is restored and the process is continued. T_h is reduced according to the cooling scheduling $T_{h-1} = c_f \cdot T_h$, which is the most commonly used in the literature [12],[22], where c_f is the cooling factor. We perform several iterations at the same T_h (so called Metropolis Monte Carlo attempts [20]) to cover a larger search space. The algorithm returns the best mapping found so far when the temperature is reduced to a certain value.

6.3 The Transformation Operator Γ

As mentioned before, the operator Γ performs the changes/moves to the mapping in order to generate a candidate mapping. Specifically, Γ generates the move to perform the *local search*, i.e., to explore the neighborhood. Three commonly used moves [22] are discussed below: *(a) relocate* a job to a different node, *(b) swap* the nodes between two jobs and *(c) interchange* the allocated jobs between two nodes. A move is accepted when it satisfies all the constraints defined in Section 3. After a successful move, the candidate mapping is evaluated in *Step* 9 (Algorithm 2).

7 Evaluation of the MVO Framework

In this section we first present the experimental setting. Based on this we evaluate the effectiveness of the MVO approach and discuss the results. Results show significant improvements in terms of *interactions, scheduling length, bandwidth utilization, CPU utilization and FT overhead.*

7.1 Experimental Settings

For the evaluation of our approach, we use randomly generated mixed-criticality sets of 40, 60 and 80 jobs denoted as $J40$, $J60$ and $J80$ respectively. All jobs, along with their replicas, are to be assigned in an optimized way onto the available nodes. All job properties are uniformly distributed within the following ranges: Replication factor $\in \{2, 3\}$, Interaction $\in [.04, .52]$, $EST \in [0, 50]$ ms, $CT \in [2, 17]$ ms, $D \in [15, 200]$ ms, Memory size $\in [4, 10]$ MB, Message size $\in [30, 120]$ bytes. Sensors and actuators are attached to arbitrary nodes. The message transmission delay time (size of the exchanged messages divided by transmission speed of the link) between communicating jobs executing on different nodes are subtracted from the deadlines. The HW model comprises 8 nodes, which are connected to a communication link with a speed of $150kbps$. The memory capacities of nodes were arbitrarily chosen as $100, 150$ and 250 MB; nodes n_2 and n_3 have sensors and n_5 and n_7 have actuators attached to them.

As used in literature [20],[22] and after investigating different runs of our algorithm with various configurations, we tune the SA parameters as follows: the value of the initial temperature (T) was set to 50000, the cooling factor was set to 0.98, and the used trade-off factors were $\psi_i = 1500$, $\psi_s = 20$ and $\psi_b = 500$ respectively. In order to generate the candidate mapping we have performed two types of moves (random reallocation and swapping - 50% each of Monte Carlo iteration) at the same temperature to cover a larger search space. The third type of move is not relevant in our case study. Experiments showed that applying both types of moves together gives a better solution than only using a single type of move.

7.2 Experimental Results

Performance Evaluation: We first observe the convergence of MVO-SA. Figure 5 shows that after a certain number of iterations with decreasing temperature the MVO function reaches a minimum. At higher temperatures, more states have been visited by the operator Γ to cover the search space. Given the proper selection of the parameters and the problem size, SA gives the global solution by construction [20],[21]. Nevertheless, we performed several experiments to evaluate if the MVO algorithm converges to a single point. Even though the algorithm is started with different feasible mappings ($Feas1$, $Feas2$ as shown in Figure 5), MVO-SA converges towards a solution every time. However, the convergence points may differ negligibly, as shown in Figure 5 in case of $J60$. A good performance test of a mapping algorithm is to take a solvable problem and add resources [13], the algorithm should return a mapping no worse than the result of the original problem. We added two more nodes to the configuration of $J40$ and the resulted mapping displayed better performance. The convergence is shown in Figure 5 marked as $J40$ (10 nodes). To show the effectiveness of

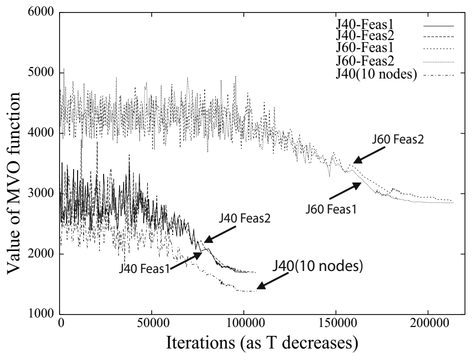

Fig. 5. Performance evaluation of MVO-SA

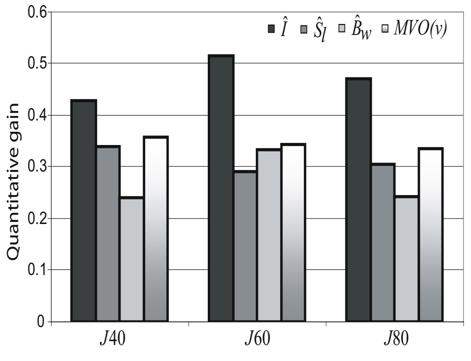

Fig. 6. Mapping M_{PF}

starting the optimization with a feasible mapping, we also ran the algorithm starting from an infeasible mapping. Though this can converge to an improved solution, it is slower than starting from a feasible solution (time for the creation of feasible mapping is included).

Quantitative Gain: We are interested in evaluating the *quantitative gain* compared to a contemporary initial solution. As this gain depends on the value of the initial mapping, we performed experiments using different initial feasible mappings. Figure 6 depicts the mapping performance profile (M_{PF}) for $J40$, $J60$ and $J80$ in terms of $\hat{I}, \hat{S}_l$, $\hat{B}_w$ and $MVO(v)$. M_{PF} is shown as relative gain with respect to the initial mapping. We observe that the gain is higher in case of $\hat{I}$, which ensures FT driven design. In our case studies, on average, our approach found 35% *better solutions* (composite FT+RT gain), which leads to significantly better designs for dependable real-time embedded systems.

CPU Utilization and FT Overhead: Figure 7 shows the computation utilization by different node's processors for jobs set $J40$, $J60$ and $J80$, which is about equally distributed among CPUs, i.e., *a proper load balancing* is maintained by the approach. It is calculated by $UF = \frac{\sum_{i=1}^{n}(M_{i,k} \cdot CT_{i,k})}{\hat{S}_l}$. We observe the FT overhead both for initial and optimized mapping in terms of scheduling length. We varied the replication factor (Replication factor = # jobs after replication/# jobs) from 1 to 3. On average, the quantitative gain is 34.33%. Obviously, scheduling length has increased due to increasing the replication factor. Therefore, a design trade-off

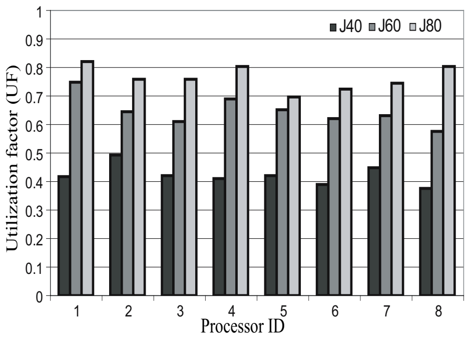

Fig. 7. CPU utilization

between RT properties and the level of FT is necessary. The quantitative gain shows that the overhead is reduced significantly by the optimized mapping, which provides an FT design with reduced scheduling length.

8 Conclusions

We have presented a generic Multi Variable Optimization (MVO) framework for designing embedded systems. The experimental results show the effectiveness of the approach and a significant improvement of the FT+RT system design compared to a straightforward solution where optimizations have not been applied. Particularly, we emphasize the following preeminent benefits of our approach: *(i)* FT is provided and then it is enhanced by restricting the possible nodes from correlated faults, *(ii)* RT requirements are met and the scheduling length is minimized, which increases the overall system performance and *(iii)* bandwidth utilization is reduced, which allows the use of a slower but cheaper bus. The generic framework also allows more variables to be considered, e.g., power.

References

1. Assayad, I., Girault, A., Kalla, H.: A Bi-Criteria Scheduling Heuristic for Distributed Embedded Systems under Reliability and Real-Time Constraints. In: DSN, pp. 347–356 (2004)
2. Dogan, A., Özgüner, F.: Biobjective Scheduling Algorithms for Execution Time-Reliability Trade-off in Heterogeneous Computing Systems. Comput. J. 48(3), 300–314 (2005)
3. Melhem, R., Mosse, D., Elnozahy, E.: The Interplay of Power Management and Fault Recovery in Real-Time Systems. IEEE Trans. Comput. 53(2), 217–231 (2004)
4. Eisenring, M., Thiele, L., Zitzler, E.: Conflicting Criteria in Embedded System Design. IEEE Design and Test 17(2), 51–59 (2000)
5. Bouyssounouse, B., Sifakis, J.: Embedded Systems Design: The ARTIST Roadmap for Research and Development. Springer, Heidelberg (2005)
6. Suri, N., Ghosh, S., Marlowe, T.: A Framework for Dependability Driven Software Integration. In: ICDCS, pp. 406–415 (1998)
7. Oh, Y., Son, S.H.: Enhancing Fault-Tolerance in Rate-Monotonic Scheduling. Real-Time Syst. 7(3), 315–329 (1994)
8. Ghosh, S., Melhem, R., Mossé, D.: Enhancing real-time schedules to tolerate transient faults. In: RTSS, pp. 120–129 (1995)
9. Kandasamy, N., Hayes, J.P., Murray, B.T.: Tolerating Transient Faults in Statically Scheduled Safety-Critical Embedded Systems. In: SRDS, pp. 212–221 (1999)
10. Lo, V.M.: Heuristic Algorithms for Task Assignment in Distributed Systems. IEEE Trans. Comput. 37(11), 1384–1397 (1988)
11. Hou, C.-J., Shin, K.G.: Allocation of Periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems. IEEE Trans. Comput. 46(12), 1338–1356 (1997)
12. Natale, M.D., Stankovic, J.A.: Scheduling Distributed Real-Time Tasks with Minimum Jitter. IEEE Trans. Comput. 49(4), 303–316 (2000)

13. Tindell, K., Burns, A., Wellings, A.: Allocating Hard Real-Time Tasks: An NP-Hard Problem Made Easy. Real-Time Syst. 4(2), 145–165 (1992)
14. Izosimov, V., Pop, P., Eles, P., Peng, Z.: Design Optimization of Time-and Cost-Constrained Fault-Tolerant Distributed Embedded Systems. In: DATE, pp. 864–869 (2005)
15. Ekelin, C., Jonsson, J.: Evaluation of Search Heuristics for Embedded System Scheduling Problems. In: Constraitnt Programming, pp. 640–654 (2001)
16. Islam, S., Lindström, R., Suri, N.: Dependability Driven Integration of Mixed Criticality SW Components. In: ISORC, pp. 485–495 (2006)
17. Laprie, J-C., Randell, B.: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Secur. Comput. 1(1), 11–33 (2004)
18. Keeney, R.L., Raiffa, H.: Decisions with Multiple Objectives: Preferences and Value Tradeoffs. Cambridge University Press, Cambridge (1993)
19. Jhumka, A., Hiller, M., Suri, N.: Assessing Inter-Modular Error Propagation in Distributed Software. In: SRDS, pp. 152–161 (2001)
20. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. J. of Science 220(4598), 671–680
21. Blum, C., Roli, A.: Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Comput. Surv. 35(3), 268–308 (2003)
22. Silva, J.L.: Metaheuristic and Multiobjective Approaches for Space Allocation. University of Nottingham, UK, PhD thesis (2003)