Design Optimization of Security-Sensitive Mixed-Criticality Real-Time Embedded Systems

Xia Zhang, Jinyu Zhan, Wei Jiang*, Yue Ma
School of Information and Software Engineering
University of Electronic Science and Technology of China
Email: zhangxia19870317@gmail.com,
{zhanjy, wejiang}@uestc.edu.cn, yue_ma_880131@hotmail.com
*Corresponding Author

Ke Jiang
Department of Computer and Information Science
Linköping University
Email: ke.jiang@liu.se

Abstract—**In this paper we are interested in security-sensitive mixed-criticality real-time systems. Existing researches on mixed-criticality systems usually are safety-oriented, which seriously ignore the security requirements. We firstly establish the system model to capture security-critical applications in mixed-criticality systems. Higher security-criticality protection always results in significant time and energy overhead in mixed-criticality systems. Thus, this paper proposes a system-level design framework for energy optimization of security-sensitive mixed-criticality system. Since the time complexity of finding optimal solutions grows exponentially as problem size grows, a GA based efficient heuristic algorithm is devised to address the system-level optimization problem. Extensive experiments demonstrate the efficiency of the proposed technique, which can obtain balanced minimal energy consumption while satisfying strict security and timing constraints.**

I. INTRODUCTION

Nowadays, more and more applications are integrated into a single platform to meet rapidly increasing cost, power, and thermal constraints [1]. Usually, these integrated platforms are connected to networks and security protections from attacks are required. It's possible that different applications in a platform have different security requirements. For example, in automobiles, the braking application needs more security protections than the infotainment system. The former may cause severe threat to human lives if it fails to resist attacks, while the later would cause less damage. Therefore, these applications are regarded as unequally security-critical, and thus, mixed-criticality aspect is brought into this kind of systems. These systems which can robustly resist against security attacks are referred as Mixed-Security-Critical Distributed Real-time Embedded System (MSCDRES).

Concrete security requirements depend on surrounding environments. Taking the flight control system used in a military unmanned aerial vehicle for example, stronger security protections are required when the system enters enemy territory. That indicates that sometimes stronger protections, which consume much more execution time and energy, are needed. However, these resources in MSCDRES are likely to be very limited, and sometimes insufficient, in handling above situation. In this case, executions of less critical applications are sacrificed to guarantee the required protections of higher critical applications. Therefore, a systematic method for security guarantees for resource limited MSCDRES is desirable. At the same time, energy budget in embedded system is usually limited. So in this paper energy efficiency is a main concern in MSCDRES, where security and real-time constraints are also guaranteed.

Researches on mixed-criticality systems emerge quickly in recent years. In [2], researchers proposed a formal model for presenting certifiable mixed-criticality systems. Authors of [3] derived an effective algorithm called PLRS to schedule mixed-criticality sporadic task system. In [4], researchers presented a Tabu Search based algorithm for designing mixed-critical applications on distributed and cost-constrained architecture. However, only safety-critical applications have been addressed, and security related issues were not covered.

More recently, some efforts have been made in security- and energy-aware task scheduling for distributed real-time systems. The authors of [5] presented a heuristic approach to search for the best system-affordable cryptographic protection for the internal communication messages. In [6], the authors proposed a resource allocation technique for optimizing the security protection in energy constrained mono-processor systems. However, these works ignored the mixed-criticality properties.

In this paper, we are interested to determine an implementation of the mixed-criticality applications on a distributed architecture such that schedulability and security constraints are satisfied and the energy consumption is well balanced. In the implementation task mappings which can achieve energy balancing and security guarantee objective is decided. Firstly, we give a formal system model of MSCDRES, where criticality level of tasks, applications and system behaviors are specified. After that, energy consumption of the system is investigated. Then the energy optimization problem is formulated, and related constraints are addressed. Due to the complexity of the problem, a Genetic Algorithm (GA) based heuristic, i.e. Energy and Security-aware Schedule Algorithm for Mixed-critical Applications (ESAMA), is proposed.

The rest of the paper is organized as follows. In Section II, a mixed-criticality system model is presented. The design optimization problem is formulated in Section III. We describe our energy optimization algorithm in Section IV. Simulations and experiments are conducted and analyzed in Section V. Finally, we conclude the paper in Section VI.

II. SYSTEM MODEL

A. Security Criticality Level

Previously, mixed-criticality systems refer criticality as safety. But security should also be part of the picture [7]. In safety-critical system, safety related applications are certified by Certification Authorities (CA) [8]. Different certification levels are assigned to applications based on prescribed standards [9], such as DO-178B and IEC61508. In security-critical systems, security-related functions should be appropriately certified, and assigned with suitable criticality levels based on existing certification standards, such as FIPS 140-2 [10] and Common Criteria (CC) [11].

Security protections based on various security reinforcement mechanisms are required in MSCDRESs. To give a reasonable security certification, application vulnerabilities, attack probabilities, and damages of system failures should be analyzed. Then the corresponding security requirements can be specified. In this paper, we introduce a Security Criticality Level (SCL) to capture the required security protection levels. Security-related applications must be certified and assigned rational SCL according to their security requirements. We assign four levels to SCL, from SCL1 to SCL4, and the bigger value stands for higher security protections. A task of higher SCL needs stronger security protection, which leads to much more time and energy consumptions. Thereby, its WCET estimation is longer. Separation mechanism must be introduced to prevent tasks of different SCLs from interfering with each other. In this paper, we assume that communication is only allowed between tasks of the same SCL.

B. Security-Sensitive Mixed-criticality Application

In a MSCDRES, SCLs are assigned to tasks according to their security requirements. However, if the system runs in an environment that is dynamically changing over time, these security requirements may change, too. Therefore, concepts of system behaviors corresponding to surrounding environment are employed, which are specified in the following.

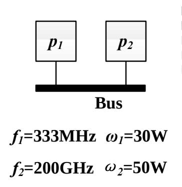
Task behavior: Task behaviors depend on surrounding environment. For example, the system running in an environment with higher attack possibilities needs higher security protection, and thus higher system load. Therefore, a task may exhibit multiple behaviors, to which different SCLs can be assigned according to each behavior's security requirement. In a task, higher SCL implies more security protection and longer WCET. A task t_i is then defined as $t_i = (\lambda_i, \tau_i)$, where λ_i is SCL of the task, which corresponds to the highest security protection of it. $\tau_i = (\tau_i(1), \tau_i(2), \dots, \tau_i(\lambda_i))$ represents WCET corresponding to each task behavior. All the behaviors' SCL assignments and WCET establishments are performed by CA. A task t_i can run at multiple SCL ranging from 1 to λ_i. If t_i runs at level l, we say that it exhibits SCLl behavior, which can be denoted as $t_i(l)$ with WCET $\tau_i(l)$.

Application behavior: Behavior of an application depends on behaviors of its subtasks. Based on the separation mechanism, all tasks of an application have equal SCLs and exhibit equal SCLs at a time. An application of SCLλ has behaviors $G(1)$, $G(2)$, $\dots$, $G(\lambda)$.

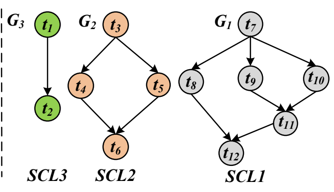
System behavior: A system behavior can be regarded as SCLl, if all the tasks currently executing in the system have SCL higher than or equal to l. In this case, execution of tasks, whose SCLs are lower than l, is sacrificed due to resource constraints. The behavior of a system can be divided in to a series of criticality levels range from 1 to $\lambda_{max}(\lambda_{max} = max\{\lambda_i | t_i \in T\})$. T stands for the tasks in the system. We assume that the system has a security monitor that detects the security threats from the surrounding environments, and then decides the system behaviors.

C. System architecture

In this paper, we focus on heterogeneous distributed MSCDRESs, which consist of multiple processing units and a common bus. For the sake of simplicity, we make several assumptions. (1)The processing capacity, power, and energy supply of these processors are different. (2)Energy consumption during system behavior transition is not considered. (3)The WCETs of message transmission are identical. (4)Transmission delay in a processor can be ignored, but the communication delay on a bus cannot be neglected.



(a) Hardware architecture



(b)Security-sensitive mixed-criticality applications

Task	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
SCL	3	3	2	2	2	2	1	1	1	1	1	1
WCET/1	5	1	2	1	6	5	5	3	2	3	1	5
WCET/2	6	4	3	5	9	9	*	*	*	*	*	*
WCET/3	10	5	*	*	*	*	*	*	*	*	*	*

(c)WCETs of tasks

	SCL 1	SCL 2	SCL 3
p_1	$t_1,t_3,t_4,t_6,t_7,t_8,t_9,t_{11}$	t_1,t_3,t_4,t_6	t_1
p_2	t_2,t_5,t_{10},t_{12}	t_2,t_5	t_2

(d)Task mapping of each system behavior

Fig. 1. A hardware architecture and a set of applications

The set of processors in the system is denoted as $P = \{p_1, p_2, \dots, p_\rho\}$. Vector $F = (f_1, f_2, \dots, f_\rho)$ represents the processing frequencies of these processors, and vector $\Omega = (\omega_1, \omega_2, \dots, \omega_\rho)$ denotes their power. An illustrative hardware platform consisting of two processors is depicted in Fig. 1(a).

There are several periodic applications running in the system. An application, denoted as G, is composed of a set of interdependent tasks and modeled as a Directed Acyclic Graph (DAG), seeing in Fig.1(b). Each node in the DAG represents a task, and edges denote data dependencies, referred as messages. A task can only be scheduled until all the messages from its preceding tasks are received. In this paper, we assume that all the applications have equal period, and release at the same time. For applications of different periods, our proposed framework can be transformed by calculating their hyper-period (LCM of all periods). We denote an application as $\Gamma(T, E, D)$. Where $e_{ij} \in E$ denotes the message from t_i to t_j. If there is a message transmitted from t_i to t_j, then $e_{ij} = 1$, otherwise, $e_{ij} = 0$. D is the common deadline equal to the period of the application set. $T = \{t_1, t_2, \dots, t_n\}$ stands for the task set of the application set. We use an $n \times \lambda_{max}$ matrix C^t to present the WCET of each task on each criticality level, which must satisfy the following property:

$$\forall c_{ij}^t \in C^t, c_{ij}^t = \begin{cases} \tau_i(j) & j \leq \lambda_i \\ 0 & j > \lambda_i \end{cases} \tag{1}$$

The value of WCET (τ_i, c_{ij}^t) is in clock cycles.

All tasks will be mapped to processors, as is seen in Fig.1(d). A 0-1 matrix A sized $\rho \times n$ is used to capture the mapping of tasks to processors. For each $a_{ij} \in A$, if t_j is mapped to p_i, then $a_{ij} = 1$, otherwise $a_{ij} = 0$.

An example of application set is shown in fig.1(b). We assume that SCLs of G_1, G_2 and G_3 is 1, 2 and 3, respectively. WCETs, which are presented in 10^4 clock cycles, of the tasks are shown in fig.1(c). In this table, WCET/1, WCET/2 and WCET/3 stand for WCETs of a task when it exhibit SCL1, SCL2 or SCL3 behavior, respectively. And symbol * means that the task cannot execute at this SCL. Seeing in Fig.1(d), all tasks can be executed and mapped to either processors when the system runs at SCL 1. If the environment getting worse, system behavior level must be raised to handle a more insecure situation. Seeing in the third column of Fig.1(d), when the system runs at SCL 2, G_1 (SCL 1) are eliminated. If environment is getting even worse, the system may exhibit the highest system behavior. Then all the applications except those of the highest SCL are eliminated, as is shown in Fig.1(d). The events of switching from one SCL to another SCL occure at intervals between two successive common periods.

D. Energy Consumption

Usually, energy budget is limited in MSCDRESs, and system failures due to energy depletion could result in serious consequence. Therefore, when scheduling tasks in MSCDRESs, energy consumption must be considered. In this paper, we assume that energy consumption at idle time is very low, and can be ignored. The execution cycles of a processor in a period can be calculated by adding all WCETs of the tasks that are allocated to the processor. Then an $\rho \times \lambda_{max}$ matrix C^r is introduced to denote the execution cycles of each processor corresponding to each system behavior. $c^r_{ij} \in C^r$ stands for the executing cycles of p_i when the system exhibits SCLj behavior. C^r can be calculated using the following function:

$$C^r = A \cdot C^t \tag{2}$$

where A is the task mapping to processors.

In a mixed-criticality system, energy consumption is varying when the system exhibits behaviors of different SCLs, since task sets corresponding to different SCLs, as well as their WCETs, are not the same. We use Y to represent the one-period energy consumption of these processors of different system behaviors. $\gamma_{ij} \in Y$ stands for the energy consumption of p_i when system exhibit SCLj behavior. And Y can be calculated using the following function:

$$\forall \gamma_{ij} \in Y,\ \gamma_{ij} = \frac{c^r_{ij} \cdot w_i}{f_i} \tag{3}$$

III. PROBLEM FORMULATION

In a MSCDRES, all the processors must be available to guarantee completeness and correctness of the system. However, energy budget in embedded systems is usually limited, and required to be effectively utilized. Therefore, a balanced energy-saving design is desirable to prolong the node lifetime[12]. In MSCDRESs, utilization rates corresponding to different system behaviors are diverse. Thus, in order to use energy budget effectively and to prolong system life as much as possible, workload among the computation nodes under each behavior must be balanced [13], while satisfying the real-time constraints of the system.

A. Motivational Example

We firstly present an illustrative example to state our problems. Let's consider the example of Fig.1, where the two processors have the same energy budget of 10J. We assume that the messages consume the same transmitting time, i.e., $20\mu s$. The deadline of the three applications is $1180\mu s$. Then two different one-period scheduling schemes is presented in Fig. 2, and both can be achieved at three SCLs. In the two schemes, different task mappings are generated. Then List Scheduling (LS)[14] algorithm is used to determine the starting time of these tasks statically. And we assume that all of the applications are non-preemptive. Scheme 1 aims at prolonging lifetime, but ignores the mixed-criticality properties. The same scheduling objective is addressed with the consideration of mixed-criticality in scheme 2.

Usually, all the tasks are fully executed, and normal security protections are inherently provided. In this case, the system runs at SCL 1 (Fig.2(a)(d)). One-period energy consumptions of the two processors are $\gamma_1 = 21.6mJ$ and $\gamma_2 = 37.5mJ$, respectively, in both schemes. Their corresponding life time, referred as number of periods, can be calculated using b_i/γ_i, where b_i denotes the energy budget of p_i, and γ_i is its the energy consumption of one period. So, in the two schemes lifetime of p_1 and p_2 is 462 and 266. The two schemes lead to the same lifetime 266 corresponding to SCL1 behavior. When the surrounding security environment is getting worse, stronger protections must be adopted. Then, system behaviors are raised to SCL 2 (fig.2(b)(e)). Execution of tasks of SCL 1 (G_1) is ignored to save more execution time and energy for security reinforcements of higher SCL tasks. WCETs of tasks of SCL 2 and SCL 3 (G_2, G_3) increase, too. We could find that in scheme 1, deadline is already missed. However, when using scheme 2, the tasks are schedulable and the system lifetime is 307. If the security environment keeps getting worse, the strongest protections is needed, and tasks of lower SCL (G_1, G_2) are sacrificed, as seen in Fig.2(c)(f). In this case, all remained tasks will exhibit behaviors of the highest SCL and consume the longest execution time. We could notice that in scheme 1, tasks executions concentrate on p_1, and CPU time on p_2 is wasted. Its system lifetime is 740. When turning to scheme 2, we find that tasks are mapped to processors more evenly, and energy is used more effectively. The corresponding lifetime is 800, which is much longer.

Based on above analysis, we find that scheme 2 can obtain schedulable results and longer system lifetime at each system behavior, comparing to scheme 1. Therefore, to achieve schedulable and energy efficient solutions, system behavior of every SCL must be investigated.

B. Objective

In this paper, our objective is to find a task mapping which can maximize the system lifetime as much as possible, under limited energy budgets. Let $B = \{b_1, b_2, b_3, \ldots, b_\rho\}$ denote the energy budgets of each processor in P. Then, ignoring mixed-criticality property, the lifetime of the system, i.e. the number of iterated execution periods, can be calculated as:

$$lifetime = min\{\frac{b_i}{\gamma_i} | i = 1, 2, \ldots, \rho\} \tag{4}$$

γ_i is one-period energy consumption of processor p_i. Then equation (4) can be transformed as

$$\theta = max\{\frac{\gamma_i}{b_i} | i = 1, 2, \ldots, \rho\} \tag{5}$$

γ_i/b_i can be regarded as weighted one-period energy consumption that needs to be minimized. Therefore, θ must be minimized to achieve the lifetime maximization objective. However, workload reduction on one processor leads to larger workload on other processors. Therefore, our energy minimization objective tends to have balanced workloads for all processors.

In MSCDRESs, energy consumption of each behavior must be studied. And system behaviors tend to switch from one SCL to another when environment changes. In this paper, we use ε_j to denote the occurrence rate of SCLj behavior. This occurrence rate is obtained based on related measurements and experiences in professional fields where the systems are applied. Then the Expectation of One-period Energy Consumption (EOEC) is formulated as follow

$$minimize\ \Theta = \sum_{j=1}^{\lambda_{max}} \varepsilon_j \cdot \theta_j, \quad s.t. \sum_{j=1}^{\lambda_{max}} \varepsilon_j = 1 \tag{6}$$

$$\theta_j = max\{\frac{\gamma_{ij}}{b_i} | i = 1, 2, \ldots, \rho,\ j = 1, 2, \ldots, \lambda_{max}\} \tag{7}$$

Θ is the EOEC that needs to be minimized to achieve energy efficiency. From equation (2), (3) and (6), we find that the value of EOEC depends on task mapping A. Thus, our objective can be described as obtaining the best A which can minimize Θ.

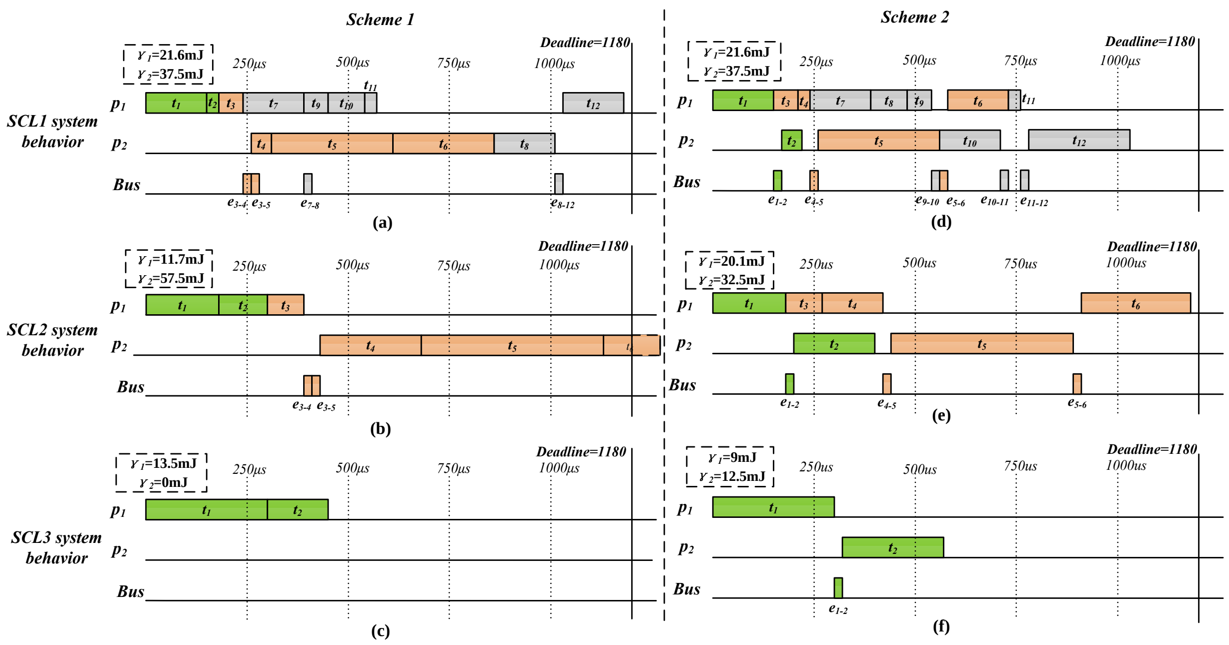


Fig. 2. Schedule examples under different schemes

C. Constraints

In MSCDRESs, where all tasks must satisfy the criticality constraints, LS must be used to determine tasks' starting time corresponding to each system behavior. Using LS, we can obtain schedule length and energy consumption of every behavior, and then check whether the constraints can be satisfied or not.

When achieving the EOEC minimizing objective, following constraints must be satisfied. **Criticality constraints**: In a LS process, only tasks with SCLs higher than SCL of current system behavior can be executed. **Dependency constraints**: Each task is allowed to be executed only after all its preceding tasks (including message transmissions) are finished. **Real-time constraint**: The real-time constraint is then transformed as that all the tasks in the applications should be finished before their deadline.

IV. PROPOSED TECHNIQUE

In this paper, our optimization problem is formulated as obtaining a reasonable task mapping, which can lead to minimum EOEC, under the criticality, dependency and real-time constraints. There are ρ^n different task mapping when a set of applications of n tasks running in the system. Therefore, for a set of applications and a given architecture, there are ρ^n different solutions that grow exponentially as n grow. It is impossible to obtain the optimal solutions for large system designs. Thus, a Genetic Algorithm (GA) based heuristic algorithm ESAMA is proposed to address it. GA is a widely used algorithm based on the emulation of natural selection and evolution, and famous for its robustness and global searching ability [15]. Usually, it is hard to decide if the results of GA are good enough, and this problem is out of the paper's scope because of its complexity. However, at most of the time, GA can obtain good solutions within acceptable execution time. In ESAMA, a task mapping A, which is regarded as a solution to our optimization problem, is formulated as a chromosome. The algorithm starts with a population of task mappings generated by a greedy strategy. The population is evolved from generation to generation towards the best solution via operations such as crossover and mutation. The whole evolution procedure terminates when the solutions are similar to some extent, or the pre-defined maximum number of iterations is reached.

A. Evaluation

In this process, the fitness of each solution is calculated. Solutions with better fitness values have higher chance to survive in the evolution procedure. In regarding that in evolution procedure, solutions with real-time violation have potential ability to evolve into better solutions, solutions with real-time violation should not be simply discarded. Therefore, fitness function must synthetically include the aspects of optimization objective and constraints. In this function, deadline misses are considered as a penalty factor which can impact the fitness values, i.e.

$$fitness(A) = \delta_1 \cdot \Theta + \delta_2 \cdot \Delta d \tag{8}$$

$$\Delta d = \sum_{\lambda=1}^{\lambda_{max}} \Delta d^{\lambda}, \quad s.t. \;\; \Delta d^{\lambda} = \begin{cases} \tau_f^{\lambda} - D & \tau_f^{\lambda} > D \\ 0 & else \end{cases} \tag{9}$$

Two criteria are employed in this function. Θ is the EOEC value corresponding to task mapping A. Δd denotes the violation degree, which is 0 when deadline constraint is guaranteed. According to the scheduling objective, solutions of lower fitness values have higher chance to survive. δ_1 and δ_2 are the weights of the two criteria, and are fine-tuned by extensive experiments to achieve the best performance.

B. Selection

To prevent premature convergence, a random selection approach, namely roulette-wheel selection, is applied. Assuming the population size in each generation is μ, a solution A_i's probability of being selected is:

$$probability(A_i) = \frac{fitness(A_{max}) - fitness(A_i)}{\sum_{j=1}^{\mu}[fitness(A_{max}) - fitness(A_j)]} \tag{10}$$

$$fitness(A_{max}) \geq fitness(A_j) \tag{11}$$

$$A_{max}, A_j \in \Phi(x), \;\; 1 \leq j \leq \mu \tag{12}$$

Algorithm 1 ESAMA

```
1  initialize the application set $\Gamma(T, E, M, D)$;
2  while population size not reach $\mu$ do        (population initialization)
3    randomly generate $A_{initial}$; $A = A_{initial}$ ;
4    while stop criteria is not satisfied do
5      $A' = adjust(A)$;
6      if $\Theta(A') < \Theta(A)$, $\tau_f(A') < D$ then $A = A'$;
7    end while; $A \rightarrow \Phi$;
8  end while;
9  while stop criteria is not satisfied do                    (evaluation)
10   for each individual $A$ in $\Phi$ do $fitness(A) = \delta_1 \cdot \Theta + \delta_2 \cdot \Delta d$;
11   empty $ElitePool$;
12   while individuals in $ElitePool$ not reach $\mu \cdot \theta_r$ do
13     obtain $A_{best} \in \Phi(x) - ElitePool$; $A_{best} \rightarrow ElitePool$;
14   end while;
15   while $MatingPool$ is not full do                    (selection)
16     $A = RouletteWheel(\Phi(x))$; $A \rightarrow MatingPool$;
17   end while; $count = 0$;
18   for each $A$ in $MatingPool$ do                    (crossover)
19     randomly generate $p$ (0$p$1);
20     if $count\%2 == 0$ and $p < \theta_c$ then $count = count + 1$;
21     if $count\%2! = 0$ and $p < \theta_c$ then
22       $perant1 = A$;$perant2 = A$; randomly select crossover point $i$,
23       $offspring1 = perant1 \cdot U_c(i) + perant2 \cdot U_c'(i)$,
24       $offspring1 = perant2 \cdot U_c(i) + perant1 \cdot U_c'(i)$,
25       offspring replace parents;
26   end for;
27   for each $A$ in $MatingPool$ do                    (mutation)
28     randomly generate $p$ (0$p$1);
29     if $p < \theta_m$ then $A' = A \cdot (i) + u^T(j)u(i)$, $A'$ replace $A$;
30   end for;
31   for each $A$ in $ElitePool$ do
32     obtain $A_{worst} \in MatingPool$, $A$ replace $A_{worst}$;
33   $A \in MatingPool$; $A \rightarrow \Phi$;
34 end while;
35 obtain optimal solution $A^*$;
```

$\Phi(x)$ denotes the population of the x-th generation. In this procedure, solutions with higher probabilities tend to have more copies, while those with lower probability are likely to have less or even no copies.

C. Crossover and Mutation

In this operation, each chromosome is selected with a crossover rate θ_c to mate and produce new offspring. Here single point crossover is applied. Assuming A_1 and A_2 are two mated chromosomes that are about to undergo crossover operation. Firstly, a crossover point i is randomly selected. Then the columns behind the i-th column of A_1 and A_2 are swapped to form offspring A_3 and A_4. The crossover operation can be formulated as follows:

$$\begin{cases} A_3 = A_1 \cdot U_c(i) + A_2 \cdot U_c'(i) \\ A_4 = A_2 \cdot U_c(i) + A_1 \cdot U_c'(i) \end{cases} \quad s.t.\ U_c(i) + U_c'(i) = U \tag{13}$$

Where U is an unit matrix, and

$$U_c(i) = \begin{bmatrix} U_{i\times i} & 0 \\ 0 & 0 \end{bmatrix} \quad U_c'(i) = \begin{bmatrix} 0 & 0 \\ 0 & U_{(n-i)\times(n-i)} \end{bmatrix} \tag{14}$$

Then one point mutation is applied, and the mutation rate is θ_m. Assuming i is the mutation point, which is actually the sequence number of the randomly selected task. A randomly selected j is the sequence number of the new processor that t_i is mapped to. Then the mutation operation is represented as follows:

$$A' = AU_m(i) + u^T(j)u(i) \tag{15}$$

$U_m(i)$ is an $n \times n$ matrix having elements $u_{11} = u_{22} = \ldots = u_{(i-1)(i-1)} = u_{(i+1)(i+1)} = \ldots = u_{nn} = 1$, and other elements are all 0. $u(i)$ and $u(j)$ are unit vectors whose i-th and j-th elements are 1, respectively.

D. Enhancement techniques

Before evolution procedures, initial population must be generated. In this algorithm, a greedy searching procedure is introduced to speed up the convergence. Firstly, a population of task mappings is generated randomly. Then each task mapping is adjusted to explore better solutions by randomly reallocating a task to another processor step by step. After a number of iterations, some solutions are kept as the initial population.

After crossover and mutation operations, a new population is generated. Due to the randomness of GA, some good solutions may be lost in the evolution procedure and slow down the convergence speed. Therefore, elite strategy is introduced in each generation, which is implemented by replacing the poorest solutions in current generation with the best solutions from the prior generation.

E. ESAMA algorithm

The pseudocode of ESAMA is presented in Algorithm 1. First, an application set is input into the algorithm. Then, a group of initial population, generated through greedy searching, treated as the starting point of evolution, as seen in lines 2 to 8. Lines 9 to 34 is the evolution procedure. Evaluation, selection, crossover and mutation procedures are implemented in lines 10-14, 15-17, 18-26, 27-33, respectively. Lines 12-14 and 31-33 are where elite strategy is implemented. After a number of iterations, population converges and the optimal solution A^* is obtained. Like traditional GA, ESAMA is sensitive to the value of parameters, such as μ, iterations, θ_c, θ_m and θ_r. Therefore, these parameters must be fine-tuned trough extensive experiments to achieve the best performance.

V. EXPERIMENT RESULTS

In this section, we present the experimental results that were obtained from the evaluation of our proposed technique. All the simulations are implemented using C#, and performed on a Windows machine having a dual-core Intel Pentium CPU with 2.22 GHz frequency and 2GB RAM. Three other algorithms (NESAMA, OESAMA, Greedy) are also studied for comparison. NESAMA and OESAMA are two other heuristic algorithms based on GA. NESAMA does not have enhancement technique, and OESAMA does not consider mixed-criticality aspects. The Greedy approach explores better solutions step by step in a greedy fashion. Two groups of experiments are carried out to evaluate the efficiency of the proposed algorithm in two different aspects. All of the algorithms require long executions because of their complexity. The enhancement techniques in ESAMA lead to longer execution time, but produce better solutions. We have implemented our algorithm on architectures that have 2 to 5 processing nodes with different computation capacities and power budget. DAGs are generated randomly as the inputs of the experiments. The WCET of tasks are randomly assigned within 1×10^4 to 20×10^4, and message transmission times are all equal to 1ms. Each application set includes applications of at least two SCL. And at most four SCL can exist in an application set.

Firstly, instances of different size are evaluated. In this set of experiments, ten instances of 10 to 50 tasks (in 2 to 5 applications) are generated randomly, and the corresponding deadlines are also determined. The applications are mapped to 2 to 5 processors. The simulation results of EOEC and schedule length is depicted in Fig.3(a) and (b). From Fig.3(a), we can find that ESAMA is the best in EOEC minimization in most cases. Most of the time, ESAMA shows much more

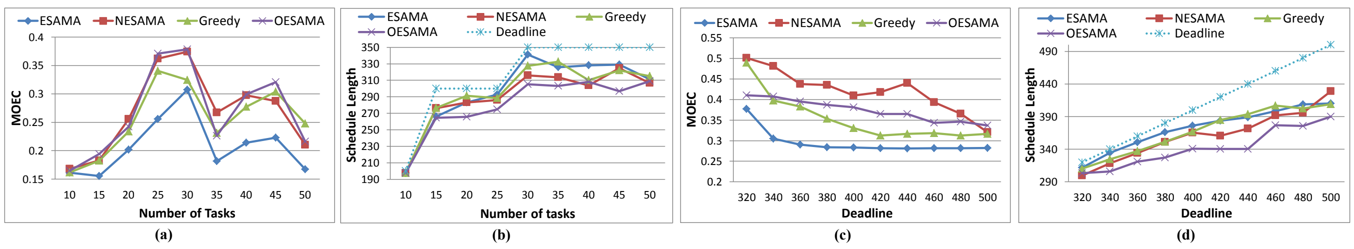


Fig. 3. Performance corresponding to different task numbers and deadline

superiority than the other three algorithms. Greedy performs slightly better than NESAMA in most cases. However, Greedy algorithm consumes much longer execution time because of the limitation of neighborhood search. So, it is reasonable to employ some enhancement techniques in ESAMA. Lacking of mixed-criticality consideration, it is difficult for OESAMA to produce good EOEC. It is worth mentioning that the EOEC in the experiments are increasing with the increased number of tasks firstly. But the EOEC decline rapidly when task number is 35 and 50. This is caused by increasing number of processors. 30-task instances and 35-task instances are mapped to 3 and 4 processors, respectively. With more processors, energy consumption of each processor tends to decrease. We measure the schedule length of all the system behaviors, and obtain the maximum one as the experiment results. From Fig.3(b), we can find that all the algorithms meet the deadline constraints.

Then we evaluate the impacts of different deadlines. The total number of tasks and applications are fixed as 30 and 4. The applications are mapped to 3 processors. Each application has its own SCL level. The results are presented in Fig.3 (c) and (d). With the increase of deadline, EOECs of all algorithms are decreasing. This illustrates that, when the time restrictions are loose, the performances of all algorithms become better. We can see from the figures that ESAMA always exhibit the highest performance. NESAMA tends to obtain the worst results, while Greedy is a little better than OESAMA. When the deadline is longer, the results of ESAMA is nearly the same, and the corresponding schedule lengths become stable. Therefore, if deadline is extended to some extent, the deadline impacts can be ignored. EOECs of OESAMA and Greedy decrease, and tend to be stable with the increase of deadline, but still worse than ESAMA. From Fig.3(d), we find that all the algorithms can satisfy the deadline constraint.

VI. Conclusion

In this paper we have addressed a system-level design optimization for security-sensitive mixed-criticality real-time systems. For these systems, security situation of surrounding environment is changing overtime, and thus the concrete security requirements are changing, too. We introduce a security-related mixed-criticality system model to capture these security requirements. SCL and system behavior is identified according to tasks' inherent security requirements and surrounding environment. Higher security-criticality, requiring higher security protections, always incurs more time and energy overhead. Thus, a good task-to-processor mapping has to be obtained, then energy consumption can be reduced, and system lifetime can be prolonged. Due to the large complexity of finding optimal task mappings, a GA based design optimization algorithm together with an enhancement technique is presented for solving the problem efficiently. Experiments are conducted on the proposed techniques besides three other approaches. Experiential results demonstrate the efficiency of the proposed framework.

References

[1] J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Pau-nicka, P. Sarathy, and J. Scoredos, "White paper: A research agenda for mixed criticality systems," *In CPS Week 2009 Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, 2009.

[2] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2010, pp. 13–22.

[3] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," in *Proc. IEEE Real-Time Systems Symposium (RTSS)*, Apr. 2011, pp. 13–23.

[4] D. Tămas-Selicean and P. Pop, "Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures," in *Proc. IEEE Real-Time Systems Symposium (RTSS)*, Nov. 2011, pp. 24–33.

[5] K. Jiang, P. Eles, and Z. Peng, "Optimization of message encryption for distributed embedded systems with real-time constraints," in *Proc. of Design and Diagnostics of Electronic Circuits & Systems*, Apr. 2011, pp. 243–248.

[6] W. Jiang, K. Jiang, and Y. Ma, "Resource allocation of security-critical tasks with statistically guaranteed energy constraint," in *Proc. Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug. 2012, pp. 330–339.

[7] B. Triquet, "Mixed criticality in avionics," *available online at http://cordis.europa.eu/fp7/ict/embedded-systems-engineering /presentations/triquet.pdf*, 2012.

[8] N. Storey, *Safety critical computer systems*. Addison-Wesley Longman, 1996.

[9] J. Rushby., "Just-in-time certification," in *12th IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS)*, Jun. 2007, pp. 15–24.

[10] "Fips pub 140-2, security requirements for cryptographic modules," *available online at http://csrc.nist.gov/groups/STM/cmvp/standards.html#02*.

[11] "Common criteria for information technology security evaluation," *available online at http://www.commoncriteriaportal.org/cc/*.

[12] Y. Tian and E. Ekici, "Cluster-based information processing in wireless sensor networks: an energy-aware approach: Research articles," *Wireless Communications & Mobile Computing*, 2007.

[13] Y. Jin, J. Jin, A. Gluhak, K. Moessner, and M. Palaniswami, "An intelligent task allocation scheme for multi-hop wireless networks," in *IEEE Transactions on Parallel and Distributed Systems*, Apr. 2011, pp. 444–451.

[14] P. Eles, Z. Peng, P. Pop, and A. Doboli, "Scheduling with bus access optimization for distributed embedded systems," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 8, no. 5, pp. 472–491, 2000.

[15] D. E. Golgberg, *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.