Design of Fault Tolerant Distributed Cyber-Physical Systems for Smart Environments

Luca Cassano, *Member, IEEE*, Antonio Miele, *Senior Member, IEEE*, Francesco Mione, Nicola Tonellotto, Carlo Vallati

***Abstract*—Cyber-Physical Systems are more and more employed to implement smart environments also in safety-critical scenarios. We here propose a novel system-level design approach capable at considering two relevant aspects of such systems: i) elaborations together with sensing and actuation need to be placed in the zones where cyber-physical interactions take place, and ii) fault tolerance mechanisms have to be incorporated to tolerate device failures. The proposed design approach identifies the optimal instantiation of the system architecture and deployment of the applications to minimize the monetary cost of the solution while guaranteeing resource requirements and fault tolerance. Experimental results show that the proposed approach reduces up to 20% the solution cost w.r.t. a straightforward hardening baseline with a computationally viable execution time.**

I. Introduction

Nowadays the pervasiveness of mobile and embedded systems and the growing processing and communication capabilities of modern devices have led to a paradigm shift in the Internet of Things (IoT) scenario. Namely, data processing is not offloaded in Cloud anymore, but it is performed through the Edge/Fog computing [1], where computations are performed directly in edge nodes of the distributed Cyber-Physical System (CPS). This trend has increased the possibilities in employing IoT applications in several scenarios to design smart environments, including those having realtime and safety-critical requirements.

The design of CPSs for safety-critical applications presents two main issues. On the one hand, the installation of edge nodes requires an accurate planning for defining the optimal positions in the environment. Indeed, the applications have to interact with the surrounding environment; therefore, they need to be executed on edge nodes specifically located in the position where the cyber-physical interaction should take place (e.g., a person detection application has to be executed in the room where video surveillance is required). On the other hand, fault tolerance mechanisms have to be introduced to allow the system to provide the service even in case of failures of edge nodes. Indeed, nodes may fail for several reasons, spanning from battery discharges or lack of connectivity to damages caused by the surrounding environment (e.g., bad weather conditions or accidental human activity). However, even if there is a large body of literature on Fog/Edge computing [1], [2], very few approaches have considered the planimetry-aware CPS design [3], [4] and the hardening of CPSs [5], [6]. Indeed, we claim that these two requirements have to be jointly considered and in a cost-effective way by instantiating and configuring a distributed computing platform with the minimal number of the cheapest edge nodes.

Given these motivations, in this paper we propose a system-level design approach to implement fault tolerant CPSs for smart environments. The approach is obtained as an extension of the Mixed Integer Linear Programming (MILP) formulation presented in [4] by introducing application-level fault tolerance requirements. Our approach instantiates a CPS architecture, composed of edge nodes, on a instrumented and cabled physical planimetry, and maps a distributed application on such an architecture taking into account processing and communication requirements, cyber-physical interactions and fault tolerance requirements. Application-level fault tolerance is achieved by hardening the critical components of an application through the active-standby technique [7]. The goal of the proposed approach is to identify the best system configuration minimizing the monetary cost of the selected architecture while guaranteeing resource requirements and fault tolerance. A systematic evaluation on a large set of problems have demonstrated the effectiveness of the proposed solution; the proposed approach reduces up to 20% the solution cost w.r.t. a considered hardening baseline with similar execution time of the MILP solver.

The text is organized as follows. Sec. II presents the system model. Sec. III and IV describe the proposed approach and the optimization engine, respectively. Sec. V discusses the experimental results, and Sec. VI draws conclusions.

II. Fault tolerant CPS model

We model a CPS in three layers (as in Figure 1): the *environment*, the *computing architecture* and the *application*.

A. Environment

The environment is modeled by means of its floor plan as shown in the bottom-most part of Figure 1. The environment is assumed to be already wired to install edge nodes in specific positions referred to as *spots*, each one characterized by the coordinates and the available connection (up/downlink bands)[1]. Moreover, the environment is assumed to be divided in *zones* where cyber-physical interactions (sensing or actuation) are

L. Cassano and A. Miele are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy. F. Mione, N. Tonellotto and C. Vallati are with the Information Engineering Department, University of Pisa, Italy.
E-mail: {first_name.last_name}@polimi.it, {first_name.last_name}@unipi.it
Manuscript received April 19, 2005; revised August 26, 2015.

[1]For the sake of brevity, a single connection type is considered but the system model can be extended to support several types in each spot.

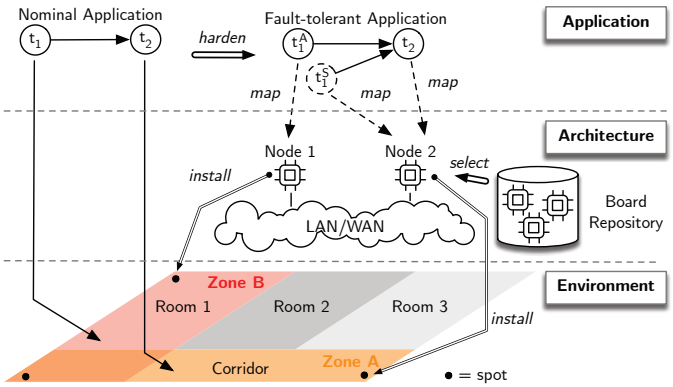

Fig. 1: System model and synthesis.

required to be performed. Each zone is defined in terms of a polygon on the planimetry, and based on its shape and position will include a certain number of spots.

B. Computing Architecture

The computing architecture (middle part of Figure 1) is composed of a number of distributed edge *nodes*. Each node is a commercial *board*, retrieved by a given vendor repository and installed in a specific spot. The board is characterized in terms of processing/storage capabilities (i.e. amount of CPU, RAM and disk), cyber-physical peripherals (list of sensors and actuators) and monetary cost. Based on the specific installation spot, each node is able to interact with the surrounding area and to communicate with the rest of the distributed platform according to the available connection band.

Fault tolerance aspects: Nodes may experience failures causing interruption of service due to several reasons (e.g. battery discharge, break out, network disconnection, etc.). Thus, we here adopt the single node *fail-silent* fault model [8] since it effectively represents failures in the considered scenario; the node either provides the correct service, or provides no service at all. Finally, we assume a single node to fail at a time; this assumption is totally realistic if we consider that in common working conditions (i.e. no radioactive or harsh environments, as at the ground level) failures are assumed to be i) independent and ii) far (in time) from each other so that upon the occurrence of a second failure, the system has already recovered from the previous one [7].

C. Application

The application (or the set of applications) to be executed by the system is organized as a set of distributed and cooperating tasks, which is classical of the IoT domain. The application is therefore modeled with a direct acyclic graph, called a *task graph*, where nodes represent tasks and edges represent data exchange from a task to another one. Each task is characterized by the resources required for its execution: i) processing and storage (i.e. CPU, RAM and disk quotas), and ii) cyber-physical (i.e. the list of sensors and actuators). Moreover, each task is also annotated with the specific zone it has to interact with (for sensing/actuation purposes), if any. On the other hand, each edge is characterized by the required uplink/downlink band. As a note, at the considered abstraction level, specific application time requirements can be expressed by translating delay constraints into computing and communication requirements, thus to ensure task completion and data transmission within a certain deadline.

Fault tolerance aspects: In a CPS, specific tasks implemented by the application may expose mission- or safety-critical requirements. On the other hand, other tasks may not be relevant from the safety point of view. Therefore, tasks are labeled as *critical* or *non-critical*; the former ones are mandatory to be executed while the latter ones can be skipped on necessity.

III. Fault Tolerant CPS Synthesis

Given as input i) an application task graph (or a set of task graphs), ii) a board repository, and iii) an environment floor plan, the nominal system synthesis problem is defined as the identification of the optimal solution in terms of

- *architecture instantiation* – the selection of a set of edge nodes from the board repository and their deployment in the selected spots, and
- *application mapping* – for each application task, the selection of the node that will have in charge its execution,

such that the monetary cost of the instantiated architecture is minimized and all cyber-physical constraints are satisfied:

- at most one node can be installed in a spot,
- each task is mapped on one and only one node providing all required sensors/actuators,
- each task is mapped on a spot that is comprised in the specific zone needed by the cyber-physical requirements of the task,
- each node provides processing and storage resources greater or equal to the sum of the requirements of all the mapped tasks,
- each node is mapped on a spot that provide communication band greater or equal to the sum of the requirements of all the mapped tasks.

Fault tolerant synthesis: To guarantee fault tolerance under the single fail-silent fault model, we adopted the *active-standby* technique [7]. This technique replicates each critical task to have a standby counterpart, mapped on a different node, that is executed only in case the active counterpart cannot work due to a node failure. From a global point of view, when a node fails, the tasks mapped on it are distributed on an alternative set of nodes in the architecture. The task graph is also enhanced so that the edges incorporate the communication requirements due to the restoring of the task state from the failed node to the one executing the standby counterpart. Thus, the synthesis problem is extended as follows:

- each critical task in the nominal application, together with its incoming and outcoming edges, is duplicated to obtain an active task and a standby one,
- the standby task must be mapped on a different node w.r.t. the active task.

On the other hand, non-critical tasks are not hardened since it is not mandatory to guarantee their completion.

TABLE I: MILP variables, sets and parameters.

Sets	
S	the set of available spots s_k
T_{nc}	the set of non-critical tasks t_j
T_c	the set of active (A) and standby (S) tasks; for each critical task j we have $t_j^{\mathsf{A}}, t_j^{\mathsf{S}} \in T_c$
B	the set of edge node b_i
R	the set of resource types, i.e., $R = \{\mathsf{cpu}, \mathsf{ram}, \mathsf{disk}\}$
C	communication directions, i.e., $C = \{\mathsf{up}, \mathsf{down}\}$
W	the set of replica types, i.e., $W = \{\mathsf{A}, \mathsf{S}\}$
Parameters	
$Bcost_i$	cost of edge node b_i
$Bres_{i,r}$	number of resources on edge node b_i of type $r \in R$
$Scom_{k,c}$	bandwidth of spot s_k in direction $c \in C$
Pre-Processing Parameters	
$m_{i,j}, m^w_{i,j}$	binary parameter set to 1 if and only if edge node b_i has all sensors and actuators required by task t_j, t_j^w
$z_{j,k}, z^w_{j,k}$	binary parameter set to 1 if and only if spot s_k is included in all zones with which task t_j, t_j^w requires to interact
$res_{i,j,r,k}$	number of resources of type r required by the *non-critical* task t_j on edge node b_i installed in spot s_k; $+\infty$ if task t_j cannot be executed on edge node b_i
$res^{\mathsf{A}}_{i,j,r,k}$	number of resources of type r required by the *active* task t_j^{A} on edge node b_i installed in spot s_k; $+\infty$ if task t_j cannot be executed on edge node b_i
$com_{j,c}$	required bandwidth of the *non-critical* task t_j in direction $c \in C$
$com^{\mathsf{A}}_{j,c}$	required bandwidth of the *active* task t_j^{A} in direction $c \in C$
Variables	
$x_{i,k,j}$	binary variable set to 1 if and only if the non-critical task $t_j \in T_{nc}$ is hosted on edge node b_i installed in the spot s_k
$x^w_{i,k,j}$	binary variable set to 1 if and only if the $t_j^w \in T_c$ task is hosted on edge node b_i installed in the spot s_k, where $w \in W$
$y_{i,k}$	binary variable set to 1 if and only if the edge node b_i is installed in spot s_k

IV. Optimization Engine

The synthesis process discussed in the previous section defines a large space of possible solutions due to the many alternative choices in i) the architecture instantiation, both in terms of the selection of the edge nodes from the board repository and of nodes deployment in the available spots, and ii) in the mapping of the tasks on the instantiated architecture by fulfilling all requirements (cyber-physical, processing and communication, and fault tolerance requirements). To automate such design space exploration we exploited a Mixed Integer Linear Programming (MILP) formulation, that we then solved with a commercial optimization engine. The MILP formulation, presented in Tables I and II, has been obtained as an extension of our previous work [4], where fault tolerance issues had not been taken into account. Briefly, the model inherits binary parameters for sensor-to-node (m), task-to-zone (z), and edge-to-spot (y) mappings, and same constraints (C2-C5) to enforce task allocation, edge node placement and CPS requirements, as in [4]. In the following, we introduce the novel fault tolerance-related aspects of the MILP formulation, highlighted in grey in the tables. In details, we divide the original set of tasks into two sets: the set of non-critical tasks T_{nc} and the set of critical tasks T_c. For each critical task, T_c contains both the active and the standby replicas. Both active and standby task replicas are characterized in terms of resource and communication requirements by the same parameters $res^{\mathsf{A}}_{i,j,r,k}$ and $com^{\mathsf{A}}_{i,c}$; moreover, each active and standby task replica is modelled by its own allocation binary variable $x^{\mathsf{A}}_{i,k,j}$ and $x^{\mathsf{S}}_{i,k,j}$, respectively, as illustrated in Table I.

TABLE II: MILP model constraints and optimization function.

Cost function	
C1	$\min \sum_{b_i \in B} \sum_{s_k \in S} Bcost_i \cdot y_{i,k}$
Task allocation	
C2a	$\sum_{b_i \in B, s_k \in S} x_{i,k,j} = 1 \quad \forall t_j \in T_{nc}$
C2b	$\sum_{b_i \in B, s_k \in S} x^w_{i,k,j} = 1 \quad \forall t_j^w \in T_c$
Edge node placement	
C3	$\sum_{b_i \in B} y_{i,k} \leq 1 \quad \forall s_k \in S$
C4a	$y_{i,k} \leq x_{i,k,j} \quad \forall b_i \in B, \forall s_k \in S, \forall t_j \in T_{nc}$
C4b	$y_{i,k} \leq x^w_{i,k,j} \quad \forall b_i \in B, \forall s_k \in S, \forall t_j^w \in T_c$
Cyber-physical requirements	
C5a	$(m_{i,j} + z_{j,k}) \cdot x_{i,k,j} = 0 \quad \forall b_i \in B, \forall s_k \in S, \forall t_j \in T_{nc}$
C5b	$(m^w_{i,j} + z^w_{j,k}) \cdot x^w_{i,k,j} = 0 \quad \forall b_i \in B, \forall s_k \in S, \forall t_j^w \in T_c$
Processing requirements	
C6	$\sum_{t_j \in T_{nc}} res_{i,j,r,k} \cdot x_{i,k,j} + \sum_{t_j^w \in T_c} res^{\mathsf{A}}_{i,j,r,k} \cdot x^w_{i,k,j} \leq$ $\leq Bres_{i,r} \quad \forall b_i \in B, \forall s_k \in S, \forall r \in R, \forall w \in W$
Communication requirements	
C7	$\sum_{t_j \in T_{nc}} com_{j,c} \cdot x_{i,k,j} + \sum_{t_j^w \in T_c} com^{\mathsf{A}}_{j,c} \cdot x^w_{i,k,j} \leq$ $\leq Scom_{k,c} \quad \forall b_i \in B, \forall s_k \in S, \forall c \in C, \forall w \in W$
Fault tolerance	
C8	$x^{\mathsf{A}}_{i,k,j} + x^{\mathsf{S}}_{i,k,j} = 1 \quad \forall b_i \in B, \forall t_j^{\mathsf{A}}, t_j^{\mathsf{S}} \in T_c, \forall s_k \in S$

New modeling constraints due to fault tolerance requirements are introduced in Table II. Specifically, the constraints C6 and C7 now take into account the resource and communication requirements for all active and standby task replicas in T_c, since, in the worst case scenario, all standby task replicas will end up replacing their active counterparts. Moreover, the requirement on the mapping of the standy task replica on different nodes w.r.t. the corresponding active tasks is modeled by the new constraint C8. The remaining part of the formulation handles the nominal architecture instantiation and application mapping as in [4].

V. Experimental Results

We performed a set of experimental evaluations of the proposed MILP formulation by means of the IBM ILOG CPLEX optimization solver [9]. To have a large set of test cases, we implemented a synthetic problem generator similar to the one in [4] defining realistic planimetries and applications and related characterizations to describe the overall smart environment. In particular, each problem is defined with a planimetry of approximately 100 zones, 200 spots and 300 tasks. In each problem, we have generated n different applications, each one having a task graph similar to the one defined in [4], and we have partitioned the zones into subsets, each one associated to a single application. Each task graph has been defined to have a number of tasks with cyber-physical requirements proportional to the number of zones associated to the application. We varied n in $\{5, 10, 20\}$ by associating to each application approximately $\{20, 10, 5\}$ zones, respectively. In this way, we have obtained large planimetries with a variable internal complexity in terms of applications dimension and cyber-physical requirements. The board repository and the board parameters have been borrowed from [4].

We tagged the generated problems several times, each time with different fault tolerance requirements. In particular, these

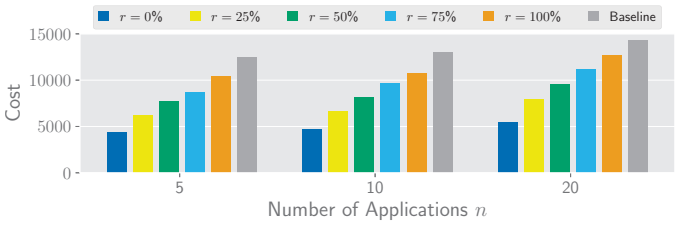

Fig. 2: Synthesis cost when varying the number of applications n and the criticality level r.

requirements have been modeled with a *criticality level* r, representing the amount of critical tasks in the considered application. We varied the value of r as follows: 0%, 25%, 50%, 75%, and 100%. In particular, when $r = 0\%$ there is no fault tolerance requirement (as in [4], while $r = 100\%$ the case where entire application requires fault tolerance mechanisms. Overall, for each number of application n and criticality level r, we generated 10 different problems.

To analyze the effectiveness of our approach, we defined a baseline in which all applications requires 100% fault tolerance and the zones in the planimetry do not share spots (similarly to the baseline in [4]). This represents the common practice of a designer that, to dominate the complexity of the synthesis problem, manually defines a solution to the fault tolerance-agnostic problem acting on each zone separately, and subsequently introduces fault tolerance by duplicating nodes and mapped tasks in each zone with a straightforward method.

Figure 2 reports the experimental results, in terms of the average costs of the various solutions for different values of the number of applications n and criticality levels r, together with the baseline costs for each application scenario. For each value of n, it is possible to notice how the cost of the solutions decreases with the lowering of the criticality level in an almost linear way spanning from $0.90\times$ when $p = 75\%$ down to $0.43\times$ when $p = 0\%$ w.r.t. $p = 100\%$: this confirms that the proposed approach is able to tailor the hardening process and the architecture instantiation based on the specific needs of the specific problem, without any over-provisioning.

At the same time, if we compare the results obtained with a different number of applications n, we notice that a larger number of applications results in higher costs; varying n from 5 to 20 results in an average $+23\%$ cost increase across different r values. Indeed, with a larger value of n, the defined problem generator reduces the number of tasks associated to each applications since the planimetry size is kept almost constant; in turn, there is a decrease of the number of spots shared across different applications that can be exploited by the approach to reduce the number of installed boards.

The comparison against the baseline shows how our synthesis strategy outperforms the manual common practice also in the scenario with $p = 100\%$, ranging from $1.13\times$ with $n = 20$ up to $1.20\times$ with $n = 5$. Differently from the baseline, our approach is able to opportunistically exploit spots placed in the intersection among zones to to pack together tasks having cyber-physical interactions with the various involved areas; in this way, a lower number of boards may be installed to host both active and standby tasks. This observation is also confirmed by the fact that the baseline presents a cost that is higher than $2\times$ the case without any fault tolerance requirements ($p = 0\%$), even if the adopted hardening technique is the task duplication. This is because our approach is capable of accommodating multiple applications on the same spot; therefore, by placing redundant tasks on boards installed on spots in zones that are of interest for multiple tasks, our approach allows to reduce the overall costs.

To compare the performance of the MILP engine, the baseline problem solution computation requires approximately from 1 to 3 minutes on average, i.e., at most $2.29\times$ the time required to solve the problem with no fault tolerance as in [4]. In the worst-case scenario, i.e., with $p = 100\%$, our solution is computed approximately from 3 to 9 minutes, with at most a $3.47\times$ slowdown. Thus, we may assume the proposed solution to be computationally affordable.

VI. Conclusions

This paper has presented a novel system-level design approach for implementing CPSs for smart environments having fault tolerance requirements. The MILP engine aims at minimizing the monetary cost of the solution by properly i) selecting devices composing the architecture, ii) installing them in the environment planimentry, iii) duplicating and deploying the application tasks on them while guaranteeing both cyber-physical and fault tolerance requirements. Experimental results show that our proposed solution is computationally viable and can reduce the total cost of the synthesis up to 20% compared to straightforward hardening baseline. As future work, we will extend the approach to handle multiple faults.

Acknowledgments

Work partially funded by the Italian Ministry of University and Research in the framework of the CrossLab project (Departments of Excellence).

References

[1] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions," *IEEE Access*, vol. 6, pp. 47 980–48 009, 2018.

[2] C.-H. Hong and B. Varghese, "Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms," *ACM Comput. Surv.*, vol. 52, no. 5, 2019.

[3] E. Fraccaroli, F. Stefanni, R. Rizzi, D. Quaglia, and F. Fummi, "Network Synthesis for Distributed Embedded Systems," *IEEE Trans. on Computers*, vol. 67, no. 9, pp. 1315–1330, 2018.

[4] G. Tanganelli, L. Cassano, A. Miele, and C. Vallati, "A methodology for the design and deployment of distributed cyber-physical systems for smart environments," *Future Generation Computer Systems*, vol. 109, pp. 420–430, 2020.

[5] A. Kouloumpris, M. K. Michael, and T. Theocharides, "Reliability-Aware Task Allocation Latency Optimization in Edge Computing," in *Proc. Intl. Symp. on On-Line Testing and Robust System Design (IOLTS)*, 2019, pp. 200–203.

[6] A. Kouloumpris, T. Theocharides, and M. K. Michael, "Cost-Effective Time-Redundancy Based Optimal Task Allocation for the Edge-Hub-Cloud Systems," in *Proc. Symp.on VLSI (ISVLSI)*, 2020, pp. 368–373.

[7] D. K. Pradhan, *Fault Tolerant Computer System Design*. Prentice Hall Inc., 1996.

[8] A. S. Tanenbaum and M. Van Steen, *Distributed Systems*. Pearson Education, 2013.

[9] IBM Corporation, "CPLEX Optimizer," https://www.ibm.com/analytics/cplex-optimizer, accessed: 2019-09-16.