Fault Detection and Co-design Recovery for Complex Task within IoT Systems

Radia Bendimerad[1], Kamel Smiri[2] and Abderrazek Jemai[3]

[1] *Université de Tunis El Manar, Faculté des Sciences de Tunis, Laboratoire LIP2, 2092, Tunis, Tunisie*

[2] *Université de Carthage, Ecole Polytechnique de Tunisie, Laboratoire SERCOM, ISAM, 2010, Manouba, Tunisie*

[3] *Université de Carthage, Ecole Polytechnique de Tunisie, Laboratoire SERCOM, INSAT, 1080, Tunis, Tunisie*

Keywords: Internet of Things (IoT), Complex Task, Sub-task, Fault Detection, Co-design Recovery.

Abstract: Internet of things (IoT) allows the implementation of embedded devices that guarantee multiple application processing. A device contains a processor core (CPU) but also hardware accelerator (FPGA) to permit a hardware-software (HW-SW) partitioning depending on its complexity. However, these devices have limited capacities, so they are exposed to faults. Thus, the system needs to be adapted in such a way to detect device faults, continues to function normally and perform tasks to produce correct output. In this paper, a holistic approach dealing with fault detection and recovery for complex tasks within the IoT was outlined. It offers a technique to diagnose the state of processing elements. Then it combines task scheduling in HW and SW parts (Co-design) in such a manner to ease the process of recovery when a system fault is detected. The proposed work ensures a better performance for the entire system. An experimental study validates the effectiveness of the present strategy without impacting system performances thanks to the contributions defined in this paper.

1 INTRODUCTION

IoT is a combination of the physical and digital world (Min et al., 2014) with features as the integration of heterogeneous devices essentially based on multiprocessors system on chip (MPSoC). The main performance issue in IoT is to guarantee data processing within a given time and constraints. However, in these systems, data requirements are ensured by embedded devices that are exposed to faults. The replacement of these components in the execution time is costly in deployment in case of fault. (Su et al., 2014)In some proposed approaches, researchers rely on the sensor's fault because they have less risk of fault, but other devices should not be excluded(Zieliski et al., 2019).

In this paper, we are dealing with problems related to performance of the IoT like the deduction of faults as early as possible and recovery strategy with run-time solutions to enable a gain of time.

The rest of this paper is organized as follows. In Section 2, we present the related works. In Section 3, we formalize the IoT system and concepts associated with the IoT application. Section 4 discusses the contributions for faults detection and recovery strategy. Section 5 tests and evaluates the proposed solutions. Section 6 concludes with some perspectives.

2 RELATED WORKS

Over the last decades, many researchers have aimed to work on various topics related to the IoT. In many IoT systems, there is an important number of works related to fault detection and system recovery. Roberto et al proposed in (Casado-Vara et al., 2019) a new strategy based on game theory and prediction of errors that could be made in the future to enhance fault-tolerant tracking using control algorithm.

Andreas S. Spanias integrates into (Spanias, 2017) a smart monitoring device (SMD) associated with vision algorithm in individual solar panel. As a result, these panels behave as nodes in an internet of things system for fault detection purposes.

According to the search reported in (Lee, 2017), detected signals by IoT device are analyzed by the cloud server to identify the relationship between the signals and defects.

Also, a method called DICE for faulty IoT devices detection with context extraction is developed in (Choi et al., 2018). The DICE is a system installed in a gateway that is connected to a cloud server. The DICE system records the IoT environment to analyze information. The information is sent to the cloud through the gateway in order to identify faults.

Bendimerad, R., Smiri, K. and Jemai, A.
Fault Detection and Co-design Recovery for Complex Task within IoT Systems.
DOI: 10.5220/0009869304840491
In *Proceedings of the 15th International Conference on Software Technologies (ICSOFT 2020)*, pages 484-491
ISBN: 978-989-758-443-5

(Zieliski et al., 2019) uses the MM method for the conjunction of all possible results of comparison tests with a comparator node that orders his neighboring the same task and compares between results.

To avoid fault occurrence in the future, (Kannan et al., 2019) presents Minimal Relevant Feature Extraction-based Class-Specific Support Vector Machine (CS-SVM) methodology to extract the relevant features and classify the faults accordingly within the IoT service in the cloud platform.

(Power and Kotonya, 2019) provides a hybrid architecture in terms of including preemptive migration and self-healing and reactive policies. The output of a node is confirmed by an acceptance test according to execution time checking. Otherwise, the task will be assigned to another node. (Tsai et al., 2019)built up a detection system architecture to avoid the abnormality among the sensors by exploring the correlation between sensors based on machine learning, classify faults on two groups and propose a fault recovery for each type by providing prediction value, adding the bias read by the faulty sensor, or find out the degradation rate of fault sensor.

In (Di Modica et al., 2019) a layered architecture is used in fog environment for fault detection and recovery. The idea is that if a given layer is faulty, the upper layer is responsible for handling faults that may occur while the IoT application runs and recovers by acting upon the entity that generated the fault.

After the conducted studies on related works, we have made several notes. As aforementioned, there are many research works focused on faults detection and system recovery. Generally, the previous works were based on periodic component test but none of them thought about the consequences such as time-consuming while the component may work well. For example, (Chudzikiewicz et al., 2015)applies a comparative test between neighbor nodes before each task processing. In our paper, testing is triggered by an abnormal device behavior. On another note, researchers turn over prediction using machine learning. It helps to monitor the system but it's not certain for real-time systems. Also, the provided solution is based on a controller that could be centralized or decentralized. The latter consists of dividing the network into sub-networks and deployed many control-head that are continuously surveyed by a master-control. However, it is better to solicit the control station only in an anomaly case.

In our approach, we propose a suspicious behavior detection to trigger the main controller in order to reduce the required response time. Also, we propose a recovery solution depending on hardware acceleration to keep the system working.

3 REQUIREMENTS FORMALIZATION

In this section, we describe the structure of an IoT system by modelling formally all used elements. Internet of things system (IoT_S) is expected to be an interrelated embedded system devices and wireless sensor to transfer data over a network. Figure 1 illustrates an example of a small IoT_S.

3.1 System Model

An IoT system includes a set of devices D and set of network link L. The devices are composed of processing units (CPU) and may contain a material accelerator (FPGA) that communicate together via a bus communication. The architecture is heterogeneous with different characteristics and capacities, i.e.:

- $S_{IoT}=(D,L)$.
- $D = SPC \bigcup MA \bigcup STATE(D_i)$ where:
  i) SPC is the software processing core,
  ii) MA is the material accelerator when the FPGA exists,
  iii) $STATE(D_i)$ is the state of a task running on a device D_i.The state could be "exec" or "exit", i.e.,

$$STATE(D_i) = \begin{cases} exec, \ if \ a \ task \ is \ running \\ exit, \ if \ a \ task \ is \ stopped \end{cases}$$

- L is a set of network links l defined by the bandwidth Bd.

3.1.1 Graph Modeling

A complex task is assigned to a device. The device shares sub-tasks and awaits a result from other devices in the system by selecting components in the proximity since there is a risk of loss of information from them. For this purpose, we will consider all devices in the system as a graph $Gr = (Vd, Ed)$. The components are the nodes V_d in the graph and edge E_d is information shared between the one responsible for distributed sub-tasks and near components. In graph theory, we can associate Laplacian matrice.

Laplacian matrice: is a symmetric matrix with one row and a column for each node. It is formed by the combination of a degree matrix and adjacency matrix. The first one is a diagonal matrix and refers to the number of edges related to each node. In the system, it indicates how many neighbors a component has. The second one is a square matrix to denote if two nodes are adjacent or not. In the system, it means if two components are neighbors or not.

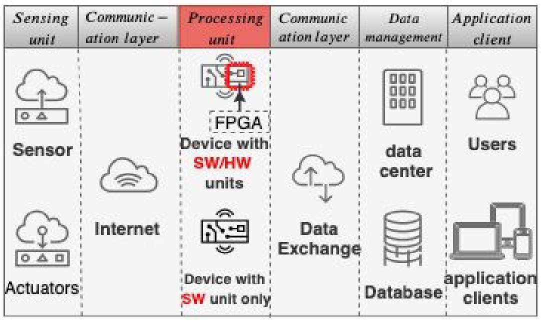


Figure 1: IoT architecture.

3.2 Application

An application in IoT is composed of a set of sub-applications. In this paper, each application is considered as combination of simple task and a complex task and it is formalized by a graph such that G=($C_{tk_i}, ST_i, E_{dges_i}, Dl_i$) where: (i) C_{tkt_i}: nodes expressing a complex task (composed of sub-tasks), ii) ST_i refers to simple task (composed of one task only), (iii) E_{dges_i}: edges to allow information exchange between nodes, (iv) Dl_i: the deadline of application.

Hence, we can define the application to be the union of complex tasks and simple tasks as in equation (1):

$$G = C_{tk_i} \bigcup ST_i \bigcup E_{dges_i} \bigcup Dl_i \qquad (1)$$

3.2.1 Complex Task

A complex task (Bradley and Strosnider, 1998) CT_i, $i = 1,..,m$ is represented by a sub directed acyclic graph DAG, $CT_i = (S_{tk_i}, E_i, CDl_i)$, where: (i) S_{tk_i} is a set of nodes that refers to sub-tasks, (ii) E_i is a set of arcs which outlined the connection between sub-tasks and (iii) CDI_i is the deadline relating to complex task which is the deadline of the leaf nodes of sub-task.

3.2.2 Sub-tasks of Complex Task

We define a sub-task to be an elementary entity denoted by S_{tk_i} with $i = 1,..,n$. We assume that sub-tasks can be of three types:

- hardware sub-task ($HW_{S_{tk}}$) are sub-tasks mapped on device with MA,
- software sub-task ($SW_{S_{tk}}$) are sub-tasks mapped on SW part of the same device,
- and other tasks that are assigned on the remaining devices containing SPC only.

A sub-task is formalized by $S_{tk_i} = (ET_i, Pr_{static_i})$ where: (i) $ET_{S_{tk_i}}$ is the required time for the execution of a sub-task, defined by the equation (2). it is equal to ET_s if the sub-task is mapped on a SW unit of a device with MA and it is equal to ET_h if the sub-task is allocated to a HW part of a device with MA. Otherwise, it is equal to ET_o if the sub-task is allocated to a device incorporating a SPC only i.e.,

$$ET = \begin{cases} ET = ET_s, \ if \ S_{tk_i} \in alloc(SW) \\ ET = ET_h, \ if \ S_{tk_i} \in alloc(HW) \\ ET = ET_o, \ Otherwise \end{cases} \qquad (2)$$

where alloc(SW), alloc(HW) expressed, respectively, two functions that group sub-tasks assigned to software part and sub-tasks assigned to hardware part. (ii) Pr_{static_i} is the priority between sub-tasks to be computed firstly as reported in (Radia Bendimerad, 2019).

3.2.3 SW-HW Partitioning (Co-design)

A sub-task can be affected to the SPC or mapped to the MA (FPGA). This concept is well known as co-design. A SW sub-task is defined by its software required time $ReqT_s$. A HW sub-task is defined by the sum of its required hardware time $ReqT_h$ and reconfiguration time RT multiplied by $\omega(ST_{ij})$ where $\omega(ST_{ij})$ is the percentage of sub-task, that will be mapped on HW part, need to be reconfigured. They are given respectively in equation (3):

$$\begin{aligned} ET_s &= ReqT_s + WT_s \\ ET_h &= ReqT_h + WT_h + RT * \omega(ST_{ij}) \end{aligned} \qquad (3)$$

where WT_h expressed the waiting time for a task to be assigned to a device that is WT_s plus the waiting time for the bus communication between SW part and HW part.

3.3 Shared Data

A function denoted by $\phi(S_{tk_i}, S_{tk_j})$ represents the rate x of shared data between two sub-tasks: a sub-task S_{tk_i} and its successors in order to regroup similar sub-tasks. For example if $\phi(S_{tk_3}, S_{tk_4}) = 60\%$ that means that sub task S_{tk_3} and sub-task S_{tk_4} shares 60% of similar data.

3.4 Scheduling Tables

Two types of tables are defined for scheduling:

- *Scheduling table for detection*: is denoted $SchedT(D_i)$ and allocates tasks to devices according to occasionned mutexes that confers to tasks a dynamic priority noted $Pr\_Sched(S_{tk_{ij}})$

- *Scheduling table for recovery*: is denoted Γ. The function $insert(S_{tk_i})$ has the role of ordering sub-tasks according to their arrived time AT. The scheduler table is associated to ranked devices and it includes all sub-tasks to be computed, i.e

$$\Gamma_j = \{S_{tk_i} \in CT_i, j \in [0,1,2]\} \quad (4)$$

where 0, 1, 2 are the rank of device that will ensure sub-tasks processing. Every device performs the execution of sub-task recorded in the table according to the order of the sub-task position and the rank of device.

3.5 System Constraints

The verification of constraints allows to guarantee the efficient operation of each component of the system:

- *Device state constraint*: The execution state of all devices in the system have to be maintained. This condition is defined by:

$$\forall\, D_i \in S_{IoT},\ STATE(D_i) == exec \quad (5)$$

- *Task priority constraint*: According to the scheduler table $SchedT(D_i)$, priority of tasks $Pr\_Sched(S_{tk_{ij}})$ have to be conserved. The condition is verified by:

$$\begin{aligned}&\forall\, ((S_{tk_{ij}}),(S_{tk_{ij+1}})) \in CT_i,\\ &PSchd(S_{tk_i}) \leq PSchd(S_{tk_{i+1}}) \implies True\end{aligned} \quad (6)$$

- *Time execution constraint* A device that works properly means, also, that the completion time is respected. This implies that the waiting time in addition to requested time must not be exceeded by the execution time result as in equation (7):

$$ET(S_{tk_{ji}}) \leq WT_{S_{tk_{ji}}} + ReqT_{S_{tk_{ji}}} \quad (7)$$

3.6 Problems

We consider the assignment of a complex task composed of n subtasks $S_{tk_1},...,S_{tk_n}$ on a set of m devices $D_1,...,D_m$. The purpose is to:

- Respect the constraints relating to sub-tasks execution time.
- Respect the priority between tasks.
- Check the state of task running on device.
- The violation of the constraints or state checking leads to trigger the controller station. Then, the strategy must enabling the execution of task with the remaining devices.

4 STRATEGY OVERVIEW

In this section, we present an overview about the proposed methodology. It is divided on 2 parts for fault detection and recovery solution.

4.1 Methodology for Fault Detection in IoT System

The basic idea is to empower a master device to handle a complex task, distribute sub-tasks on slave devices over the system and trigger the control station when an abnormal behavior on a device occurs according to the defined constraints. The control station isolates the suspicious device as it can propagate to others and test it in order to specify the kind of fault. The proposed methodology is given in figure 2:

- Device selection and sub-tasks distribution: It includes mainly two inputs. The first type of input is a DAG: $CT_i = (S_{tk_i}, E_i, CDl_i)$ representing a complex task with sub-task features $S_{tk_i} = (ET_i, Pr_{static_i})$ as explained previously. The second input for our methodology is the devices D_i existing in the IoT system for task execution. To start, a sub-graph constituted of a set of sub-tasks is attributed to a device that we call a master device. As soon as the master device receives the DAG, it determines the neighbor devices, by using a Laplacian matrix, called slave nodes to contribute in the execution operation. The master emits to slaves sub-tasks nodes.
- Constraints verification: each slave device node sends first to master the STATE(D_i) of the current sub-task that should be equal to exec to continue the verification of the rest of constraints. A schedule table $SchedT(D_i)$ is drawn up where the scheduled sub-tasks depending on dynamic priority $Pr\_Sched(S_{tk_{ij}})$, must be verified. When priority constraint is respected the sub-task execution time is checked and should be less than estimate value. Finally, the results are sent to the master. Using these informations, the master device decides whether or not constraints are violated and which device has anomalous behavior to trigger the controller.
- Base station control: when an abnormal event has occurred on the device, the controller is triggered and a diagnose of the suspicious device has to be performed. It applies a series of tests to specify the malfunction origin. The test is a vector denoted $Vec[D_i]$ where the device D_i is the row and the column refers to a test. The test may be related to a hardware or software problem.

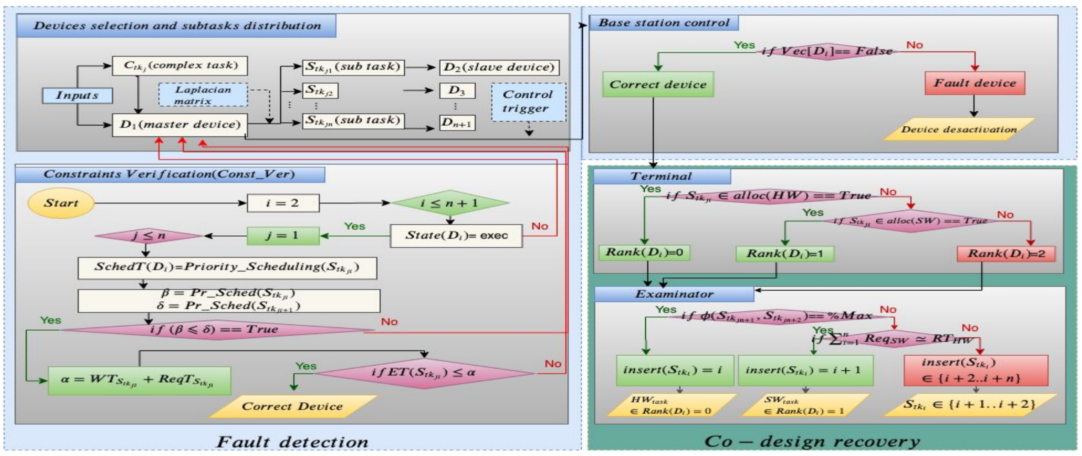


Figure 2: Strategy overview.

Vector is a Boolean value that can be as follow:

$$Veq[D_i] = \begin{cases} 1, & \text{if } device\ is\ fault \\ 0, & \text{otherwise.} \end{cases} \qquad (8)$$

As output, the fault devices are determined in the IoT system.

4.2 Methodology for System Recovery in IoT System

The key point here is to find an alternative solution in response to the isolated component. For this purpose, an available hardware part of the functional device will be reconfigured equally to the deficient one. Three modules with policy collaborate together to ensure the safe running of the system: Terminal, examinator, and scheduler.

- Terminal: considers the output of fault detection and generates a list containing available devices operating properly. It applies a categorization method to separate between the devices including material accelerator and the device without MA. This method allows to obtains a list where the devices are ranked by the function Rank(D_i) where $Rank(D_i)$ =

$$\begin{cases} 0, & for\ the\ HW\ part\ of\ devices\ including\ MA \\ 1, & for\ the\ SW\ part\ of\ devices\ including\ MA \\ 2, & for\ other\ devices\ that\ include\ SPC\ only \end{cases}$$

- Examinator: The main function of the examinator is to take into account the list of available ranked devices produced by the terminal, then to map tasks to the SW part or HW part. It receives the sub-tasks arrived at time AT and analyses each of them according to the similarity between tasks. In fact, to map a task to the SW part, it needs only a computation time, unlike the HW part which requires supplemental time in addition to computation time consisting of the configuration. However, the reconfiguration operation is time-consuming and acceleration resources are limited(Jeong et al., 2000). In line with this concern, the trick is to minimize the time spent in reconfiguration by comparing between tasks and grouping those with the same type as the similarity between data reduces the total reconfiguration. A function denoted by $\phi(S_{tk_i}, S_{tk_j})$ represents the rate x% of data sharing between two tasks. The maximum value is the one that will be taken, i.e : $Max\phi(S_{tk_i}, S_{tk_j})$.
- Scheduler: The output of previous modules are the input in the scheduler. This process permits to tasks to be assigned to the right ranked devices. the goal of the scheduler is to obtain a scheduling table Γ where are inserted tasks according to their priority in the relevant column. The table Γ contains a number of tasks arrived at time AT_i. For a given sub-task ST_i, the insertion in the table Γ will be as follow :

$$\begin{cases} insert(S_{tk_i}) = 1^{st}\ column,\ if\ \Gamma = \emptyset, \\ insert(S_{tk_i}) = j^{th}\ column,\ \text{otherwise.} \end{cases} \qquad (9)$$

For this, we admit that table columns contains sub-tasks with priority Pr_i. They will be allocated to ranked resources denoted by 0, 1 or 2.

– Tasks allocated to 0 are tasks with highest priority. It corresponds to tasks with the maximum value $\phi(S_{tk_i}, S_{tk_j})$ computed in previous module for similar data of tasks. The task selected will be moved to the i^{th} column of the table.

– Tasks allocated to 1 are tasks with second priority. For this we study the estimate's execution time, then we compare it to time used for reconfiguration. In fact a HW computation in material accelerator (FPGA) and a SW computation in CPU cannot run concurrently. Hence we associate a HW reconfiguration to a SW computation. As a consequence the execution time of one or more software task will be approximately equal to the reconfiguration time, i.e, $\sum_{S_{tk}=1}^{n} Req_{SW} \simeq RT_{HW}$. The task selected is shifted to the $i+1^{th}$ column of the table.

– Tasks allocated to 2 are tasks with lowest priority. It concerns devices not equipped with an accelerator. Thereupon, the estimated time execution is checked, then the task is placed in the table according to available matching device, i.e, $[i+2,.., i+n]$.

5 EXPERIMENTATION

5.1 Case Study

We consider the surveillance area application. It is composed of a group of cameras attached to the Raspberry Pi device through the PiCamera module, capable of capturing images and high definition videos. RaspberryPi provides access to the internet allowing connection with devices for automatic execution functionality. The system is essentially composed of a kit of heterogeneous devices as well as Raspberry Pi, Arduino (system on chip(SoC) with SW part only). Also, the system contains devices of a Zedboard type (SoC with SPC associated with an MA area. The simulation was done by using Python that supports the code to run on any kind of computer.

5.1.1 Device Fault Detection

The data recorded by each camera is considered as sub-application that will be supported by a master device noted D_1 and choose the nearby devices via the Laplacian matrix called slave devices and noted from D_2 to D_4. Let's take the case where a problem occurs with D_3. We suppose that the dynamic priority of the ordering table $SchedT(D_i)$ has not been respected, i.e. $Pr\_Sched(S_{tk_{ij}}) \leq Pr\_Sched(S_{tk_{ji+1}})$ and that the device is still executing the task $S_{tk_{ij}}$. The report is then sent to the D_1, which suspends D_3 and triggers the controller (Figure 3). The control base applies a series of tests. If the value of the result is equal to 1, the device is deactivated. Otherwise, the problem is considered as temporary failure and the device will be reset.

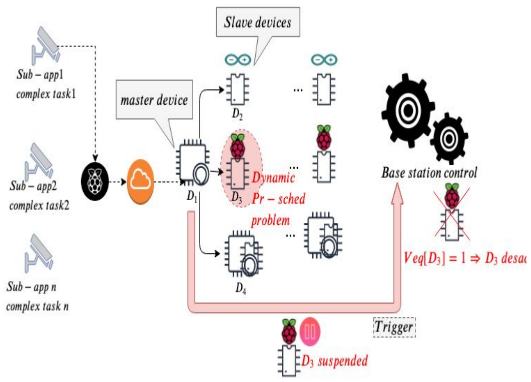


Figure 3: Device fault detection.

5.1.2 Co-design Recovery

- the second part concerns the allocation of sub-tasks on the remote devices. Let's suppose that D_3 is deficient. The rest of devices: D_1, D_2 (slave device with SW part only) and D_4 (slave device with SW/HW parts) have the respective ranks in scheduler table Γ (0, 1 and 2), i.e: $rank(D_4) = 0$ for MA in the i^{th} column, $rank(D_4) = 1$ for SW part in the $i+1^{th}$ column and $rank(D_2) = 2$ in the $i+2^{th}$ column. We applied the proposed Co-Design recovery strategy to a set of HEVC standard tasks (Alvarez-Mesa et al., 2012). Table 1 shows some examples of HEVC tasks arrived at time AT_i and the measured ϕ expressing the ratio of shared data between every pair of tasks.

- The maximum value of measured rate will be the one selected and will be inserted in the i^{th} column of Γ that corresponds to $rank(D_4) = 0$ and means a HW reconfiguration. According to the Table 1, the max value of *phi* is taken which is 0.4% (pink cell) corresponds to IT and entropy tasks (gray cells) as in equation (10):

$$\begin{cases} Entropy\ encoding \in alloc(HW) \\ IT \in alloc(HW) \end{cases} \tag{10}$$

- For tasks for which $insert(S_{tk_i}) = i+1^{th}$ column, Table 2 indicates in millisecond the estimated SW execution time for tasks. One task or more could be chosen as long as $\sum_{i=1}^{n} Req_{SW} \simeq RT_{HW}$ that corresponds to $rank(D_4) = 1$. According to Table 2 $Req_{SW}(IQ) + Req_{SW}(ALF) = 210 + 40 = 250ms$. In another hand $RT_{HW}(IT) + RT_{HW}(entropy) = 197 + 63 \leq 260ms$. This implies that $\sum_{S_{tk}=IQ}^{ALF} Req_{SW} \simeq RT_{HW}$. In this study we neglected the time spent in communication between the SW and HW part.

- For tasks that do not meet the requirements (SAO), the insertion will be in the $i+2^{th}$ column

Table 1: Values of ϕ for HEVC standard tasks.

	Entr encod	IQ	IT	SAO	ALF
Entr encod	1	0.1	0.03	0.05	0.15
IQ	0.25	1	0.22	0.23	0.26
IT	0.4	0.01	1	0.16	0
SAO	0.38	0.09	0.05	1	0.2
ALF	0.01	0.27	0.3	0	1

Table 2: Estimation for execution time for reconfiguration and required software time (ms).

	RT_{HW}	Req_{SW}
Entr encod	63	71
IQ	190	210
IT	197	160
SAO	432	500
ALF	29	40

and fits with $rank(D_2) = 2$. Table 3 summarizes the obtained results.

5.2 Performace Evaluation

The proposed strategy impacts the overall performance of the IoT system in terms of efficiency and robustness. it applies a supervising method for faults devices and keeps the system running with fewer devices. For this study, we select two metrics: detected fault rate and recovery time gained.

- ***Detected Fault Rate:*** Denoted by $\Psi(IoT_S)$ and represents the total detected faults counted compared with the total number of given faults. The formula is given in (11):

$$\Psi(IoT_S) = \frac{Nbr.\ of\ faults\ recorded}{Nbr.\ of\ total\ faults} \qquad (11)$$

  For the case study, the Detected fault rate $\Psi(IoT_S) = 82\%$

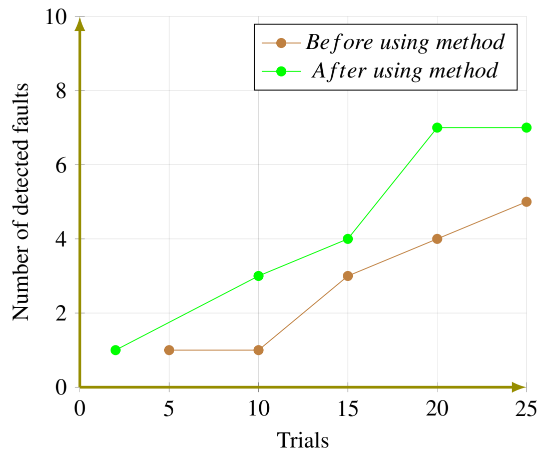
- ***Total Gained Time in Recovery:*** - Denoted by $\Omega(IoT_S)$. It relates to the time gained throughout the recovery step by using material accelerator MA comparing with using software processing core SPC only. The formula is given in (12):

$$\Omega(IoT_S) = \frac{ET_{system}}{2 * ET_{SW}} \qquad (12)$$

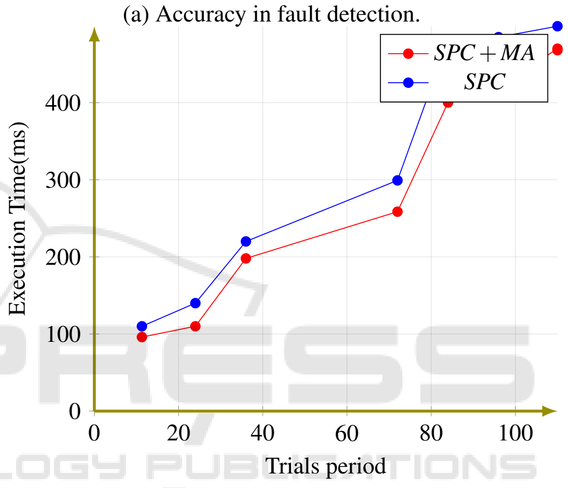
  Where ET_{system} is the ET computed by the sum of ET_{HW} and ET_{SW}. In this study $\Omega(IoT_S) = 69\%$ We suppose that the number of faults in the system varies from 1 to 10 faults and generate many trials.

Table 3: Schedular table for HEVC tasks.

$insert(S_{tk_i}) =$	i	i+1	i+2
$Rank(D_i) =$	0	1	2
S_{tk_i}	Entropy, IT	ALF, IQ	SAO



(a) Accuracy in fault detection.



(b) Execution time rate in IOT system.

Figure 4: Performance evaluation in terms of accuracy in fault detection and execution time in IoT system.

Figure 4(a) shows the increase number of detecting fault before and after using the proposed method (in green color) that is more accurate. Figure 4(b) compares in execution time between recovery solution that uses SPC only (in blue) and the one that uses SPC and MA and proves that the second one (in red)consumes less time.

6 CONCLUSIONS

In this paper, we proposed a new methodology to resolve the faults in the IoT system and define a Codesign recovery in order to avoid losing time in execution with defective devices. The experiments show the enhancement of performances with the reduction of execution time and the increase of faults detection due to the proposed solution.

In future works, we intend to differentiate between the multiple faults that can occur all at once by using smart IoT devices. Also, we investigate for reducing time for fault type identification by inserting a database knowledge directive. Finally, we plan to add a policy to manage and actions to handle the overloads devices.

REFERENCES

Alvarez-Mesa, M., Chi, C. C., Juurlink, B., George, V., and Schierl, T. (2012). Parallel video decoding in the emerging hevc standard. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1545–1548. IEEE.

Bradley, K. and Strosnider, J. K. (1998). An application of complex task modeling. In *Proceedings. Fourth IEEE Real-Time Technology and Applications Symposium (Cat. No. 98TB100245)*, pages 85–90. IEEE.

Casado-Vara, R., De la Prieta, F., Rodriguez, S., Sitton, I., Calvo-Rolle, J. L., Venayagamoorthy, G. K., Vega, P., and Prieto, J. (2019). Adaptive fault-tolerant tracking control algorithm for iot systems: Smart building case study. In *International Workshop on Soft Computing Models in Industrial and Environmental Applications*, pages 481–490. Springer.

Choi, J., Jeoung, H., Kim, J., Ko, Y., Jung, W., Kim, H., and Kim, J. (2018). Detecting and identifying faulty iot devices in smart home with context extraction. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 610–621. IEEE.

Chudzikiewicz, J., Furtak, J., and Zielinski, Z. (2015). Fault-tolerant techniques for the internet of military things. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 496–501. IEEE.

Di Modica, G., Gulino, S., and Tomarchio, O. (2019). Iot fault management in cloud/fog environments. In *Proceedings of the 9th International Conference on the Internet of Things*, pages 1–4.

Jeong, B., Yoo, S., Lee, S., and Choi, K. (2000). Hardware-software cosynthesis for run-time incrementally reconfigurable fpgas. In *Proceedings 2000. Design Automation Conference.(IEEE Cat. No. 00CH37106)*, pages 169–174. IEEE.

Kannan, R., Manohar, S. S., and Kumaran, M. S. (2019). Iot-based condition monitoring and fault detection for induction motor. In *Proceedings of 2nd International Conference on Communication, Computing and Networking*, pages 205–215. Springer.

Lee, H. (2017). Framework and development of fault detection classification using iot device and cloud environment. *Journal of Manufacturing Systems*, 43:257–270.

Min, D., Xiao, Z., Sheng, B., Quanyong, H., and Xuwei, P. (2014). Design and implementation of heterogeneous iot gateway based on dynamic priority scheduling algorithm. volume 36, pages 924–931. Sage Publications Sage UK: London, England.

Power, A. and Kotonya, G. (2019). Providing fault tolerance via complex event processing and machine learning for iot systems. In *Proceedings of the 9th International Conference on the Internet of Things*, pages 1–7.

Radia Bendimerad, Kamel Smiri, A. J. (2019). Performance estimation within iot system. In *IINTEC. 2019 international conference on Internet of Things. Embedded Systems and Communications*. IEEE.

Spanias, A. S. (2017). Solar energy management as an internet of things (iot) application. In *2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)*, pages 1–4. IEEE.

Su, P. H., Shih, C.-S., Hsu, J. Y.-J., Lin, K.-J., and Wang, Y.-C. (2014). Decentralized fault tolerance mechanism for intelligent iot/m2m middleware. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 45–50. IEEE.

Tsai, F.-K., Chen, C.-C., Chen, T.-F., and Lin, T.-J. (2019). Sensor abnormal detection and recovery using machine learning for iot sensing systems. In *2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 501–505. IEEE.

Zieliski, Z., Chudzikiewicz, J., and Furtak, J. (2019). An approach to integrating security and fault tolerance mechanisms into the military iot. In *Security and Fault Tolerance in Internet of Things*, pages 111–128. Springer.