Evaluation of Fault Tolerant Online Scheduling Algorithms for CubeSats

Petr Dobiáš
Univ Rennes, Inria, CNRS, IRISA
F-22305 Lannion, France
petr.dobias@irisa.fr

Emmanuel Casseau
Univ Rennes, Inria, CNRS, IRISA
F-22305 Lannion, France
emmanuel.casseau@irisa.fr

Oliver Sinnen
PARC Lab, University of Auckland
Auckland, New Zealand
o.sinnen@auckland.ac.nz

Abstract—**Small satellites, such as CubeSats, have to respect time, spatial and energy constraints in the harsh space environment. To tackle this issue, this paper presents and evaluates two fault tolerant online scheduling algorithms: the algorithm scheduling all tasks as aperiodic (called ONEOFF) and the algorithm placing arriving tasks as aperiodic or periodic tasks (called ONEOFF&CYCLIC). Based on several scenarios, the results show that the performances of ordering policies are influenced by the system load and the proportions of simple and double tasks to all tasks to be executed. The "Earliest Deadline" and "Earliest Arrival Time" ordering policies for ONEOFF or the "Minimum Slack" ordering policy for ONEOFF&CYCLIC reject the least tasks in all tested scenarios. The paper also deals with the analysis of scheduling time to evaluate real-time performances of ordering policies and shows that ONEOFF requires less time to find a new schedule than ONEOFF&CYCLIC. Finally, it was found that the studied algorithms perform well also in a harsh environment.**

Index Terms—**CubeSats, Fault Tolerance, Online Scheduling, Real-Time Systems, Multiprocessors**

I. Introduction

The idea of CubeSats dates back to 1999 and its aim was to provide affordable access to space by defining standard dimensions to reduce costs and time [1]. At present, CubeSats become more and more popular, their number of launches increases and they are built not only at universities but also by companies and space agencies [2].

CubeSats are small satellites consisting of several units (e.g. 1U, 2U, 3U or 6U) where each unit (1U) is a $10\,cm$ cube, which can weigh up to $1.3\,kg$ [1]. CubeSats are composed of several systems, such as on-board computer, electrical power system, attitude determination and control system, communication system, and payload. Their missions are aimed at scientific investigations, like studying urban heat islands [3].

CubeSats operate in the harsh space environment, where they are exposed to charged particles and radiations. These phenomena cause both transient effects, such as single event upsets, and long-term effects, e.g. total ionising dose [4]. Consequently, it is necessary for CubeSats to be robust against faults to achieve their mission.

Our aim is to provide CubeSats with fault tolerance. As there are several systems aboard CubeSats and most of them has its own processor, we present a solution gathering all processors on one board. This modification will reduce space and weight and improve the system resilience. First, a shielding against radiation will be easier to put into practice [5]. Second, a CubeSat will remain operational even in case of a permanent processor failure because processors are not dedicated to one system (as it is done in current CubeSats) and each processor can execute any task. Although this implementation choice may seem considerable, it was successfully realised on board of ArduSat, which counts 17 processors on one board [6].

Once all processors are gathered on one board, we intend to use the proposed scheduling algorithms dealing with all tasks (no matter the system) on board of any CubeSat or any small satellite. These algorithms schedule all types of tasks (periodic, sporadic and aperiodic), detect faults and take appropriate measures to provide correct results. They are executed online in order to promptly manage occurring faults and respect real-time constraints. They are mainly meant for CubeSats based on commercial-off-the-shelf processors, which are not necessarily designed to be used in space applications and therefore more vulnerable to faults than radiation hardened processors.

The contributions of this paper are as follows:

- we assess the algorithm performances using the rejection rate (which represents the ratio of rejected tasks to all arriving tasks and which we try to minimise) for different scenarios; whenever possible the results are compared to the optimal solution provided by CPLEX solver;
- we analyse the scheduling time of ordering policies for two studied algorithms;
- we evaluate how the algorithms deal with faults;
- based on the algorithm performances, we suggest which algorithm should be used on board of the CubeSat.

The remainder of this paper is organised as follows. Section II sums up the related work on fault tolerance in CubeSats and Section III presents our system, task and fault models. The algorithms are described in Section IV. Section V then introduces the experimental framework and the results are analysed in Section VI. Section VII concludes this paper.

II. Fault Tolerance in CubeSats

This section summarises how the fault tolerance is put into practice on board of CubeSats.

First of all, we stress that not all CubeSats are fault tolerant mainly due to financial or time constraints [7]. If a CubeSat is made more robust, the fault tolerance is implemented rather in hardware than in software. Though, a usual hardware technique is redundancy of several or all components [8], 43% of CubeSats do not use any redundancy due to budget, time or space constraints [9]. Other fault tolerant techniques aboard CubeSats are for example watchdog timers [10] or data protection techniques [11]. It is also possible to analyse error reports, scan important parameters, like power consumption or temperature, in order to detect abnormal behaviour and send appropriate commands if necessary [8], [11], [12].

Even though software techniques are not common in CubeSats, they are widely used in other applications, e.g. creation of several task copies [13], [14] or task rescheduling [15].

III. System, Fault and Task Models

Similarly to the model in [16], the studied system consists of P interconnected identical processors[1]. It handles all tasks on board of the CubeSat. These tasks are mostly related to housekeeping (like sensor measurements), communication with ground station and storing or reading data from memory.

The task model distinguishes aperiodic and periodic tasks. An *aperiodic task* is characterised by arrival time a_i, execution time et_i, deadline d_i and task type tt_i, which will be defined in the next paragraph. A *periodic task* has several instances and has four attributes: ϕ_i (which is the arrival time of the first instance), execution time et_i, period T_i and task type tt_i. We consider that the relative deadline equals the period. For both, aperiodic and periodic tasks, a task must be executed before deadline or beginning of the next period, respectively.

As for the fault model, it considers both transient and permanent faults and it distinguishes two task types: simple (S) and double (D) tasks depending on the fault detection. For both task types, we differentiate two types of task copies: *primary copy* (PC) and *backup copy* (BC). The former copies are necessary for task execution in a fault-free environment. If a primary copy is faulty, the corresponding backup copy is scheduled. *Simple tasks* have only one PC because a fault is detected by timeout, no received acknowledgment or failure of data checks. By contrast, the fault detection for *double tasks* requires the execution of two PCs[2] and then their comparison because fault detection techniques for simple tasks may not be sufficient to detect a fault.

Our objective is to minimise the task rejection rate subject to real-time and reliability constraints, which means maximising the number of tasks being correctly executed before deadline even if a fault occurs.

IV. Presentation of Algorithms

This section describes two algorithms meant for global scheduling on multiprocessor systems. First of all, it starts with several general principles applicable for both of them.

All tasks arriving to the system are ordered in a task queue using different policies. The policies for aperiodic tasks are as follows: Random, Minimum Slack (MS) first, Highest ratio of et_i to (d_i-t) first, Lowest ratio of et_i to (d_i-t) first, Longest Execution Time (LET) first, Shortest Execution Time (SET) first, Earliest Arrival Time (EAT) first and Earliest Deadline (ED) first; and the ones for periodic tasks are as reads: Random, Minimum Slack (MS) first, Longest Execution Time (LET) first, Shortest Execution Time (SET) first, Earliest Phase (EP) first and Rate Monotonic (RM).

A preemption is not authorised but the task rejection is allowed. A task t_i is rejected at time t and removed from the task queue if its task copies do not meet its deadline, i.e. $t + et_i > d_i$ for the aperiodic task or $t + et_i > \phi_i + k \cdot T_i$ for the k^{th} instance of periodic task. We remind the reader that a simple task t_i has one PC (denoted by PC_i), whereas a double task t_i has two PCs (labeled respectively $PC_{i,1}$ and $PC_{i,2}$) in a fault-free environment.

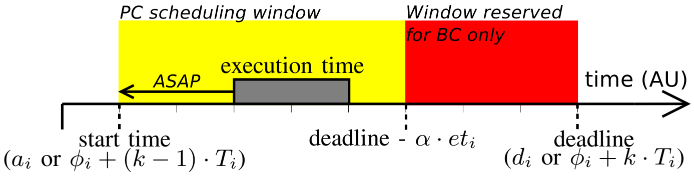

PC scheduling window
Window reserved
for BC only
execution time
ASAP
time (AU)
start time
$(a_i$ or $\phi_i + (k-1) \cdot T_i)$
deadline - $\alpha \cdot et_i$
deadline
$(d_i$ or $\phi_i + k \cdot T_i)$

Fig. 1: Principle of scheduling task copies

As Fig. 1 shows, all primary copies are scheduled as soon as possible to avoid idle processors just after the task arrival and possible high processor load later. As our goal is to minimise the task rejection, the algorithm reserves a certain time of the task window to place a backup copy if the PC execution is faulty. The end of the PC scheduling window is defined as $d_i - \alpha \cdot et_i$ for the aperiodic task and $\phi_i + k \cdot T_i - \alpha \cdot et_i$ for the k^{th} instance of periodic task (with $\alpha \geqslant 1$). In this paper, we consider without lost of generality that $\alpha = 1$. If the algorithm finds out that a primary copy was faulty, the corresponding backup copy is scheduled and can start its execution immediately, i.e. even during the PC scheduling window, because its results are necessary.

As for the processor allocation, we call a *slot*, a time interval on processor schedule. The algorithm starts to check the first free slot on each processor and then, if a solution was not found, it continues with next slots (second, third, ...) until a solution is obtained or all free slots on all processors tested. The principle of the search is illustrated in Fig. 2, where xC_i stands for primary or backup copy of task t_i.

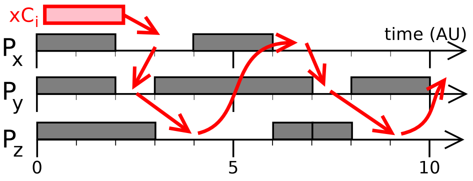


Fig. 2: Principle of algorithm search for a free slot

A. Mathematical Programming Formulation

We define the mathematical programming formulation of the studied scheduling problem as follows:

[1] To simplify, a system presented in this paper is composed of homogeneous processors sharing the same memory. Nevertheless, this model can be easily extended to a system with heterogeneous processors, like in [17].

[2] Two task copies of the same task t_i can overlap each other on different processors but it is not necessary. However, they must not be executed on one processor in order to be able to detect a faulty processor.

$$\max \sum\limits_{i}^{\text{Set of tasks}} t_i \text{ is accepted}$$

subject to

$$\begin{cases} \textbf{1)}\ a_i \leqslant start(PC_i) < end(PC_i) \leqslant d_i - et_i \\ \textbf{2)}\ \begin{cases}\text{For simple tasks: } PC_i \in P_x \Rightarrow BC_i \notin P_x \\ \text{For double tasks: } PC_{i,1}\in P_x \Rightarrow (PC_{i,2}\notin P_x \text{ and} \\ \qquad\qquad BC_i \notin P_x) \text{ and } PC_{i,2}\in P_y \Rightarrow BC_i\notin P_y\end{cases} \\ \textbf{3)}\ (xC_i \text{ and } xC_j)\in P_x \Rightarrow \\ \quad end(xC_i)\leqslant start(xC_j) \text{ or } end(xC_j) \leqslant start(xC_i) \\ \textbf{4)}\ \text{For double tasks: } PC_{i,1} \text{ scheduled } \Leftrightarrow PC_{i,2} \text{ scheduled}\end{cases}$$

The objective function is to maximise the number of accepted tasks, which is equivalent to minimise the task rejection rate. The first constraint is related to the PC scheduling window depicted in Fig. 1 and the second one forbids task copies of the same task to be scheduled on the same processor. The third constraint stands for no overlap among task copies xC (i.e. PC or BC) on one processor, i.e. only one task copy can be scheduled per processor at the same time. The last constraint requires that both primary copies of double tasks are scheduled.

*B. Online Scheduling Algorithm for All Tasks Scheduled as Aperiodic Tasks (*OneOff*)*

The online algorithm scheduling arriving tasks as aperiodic ones is called OneOff in this paper. When it is used, all tasks are considered as aperiodic, which means that each instance of periodic task is transformed into an aperiodic task[3].

The principle of OneOff is summarised in Fig. 3.

First (Line 1), the algorithm is triggered if (i) a processor becomes idle, (ii) a processor is idle and a task arrives, or (iii) a fault occurs.

If there is neither task arrival nor fault occurrence and a processor becomes/is idle (i.e. Case (i)), a new search for a schedule is not necessary and task copies are committed using an already defined schedule (Lines 2-6).

Otherwise (Lines 7-19), new task copies (PC(s) for new task and BC for task impacted by fault) are added to the task queue. Then, the algorithm removes all task copies, which have not yet started their execution, it orders tasks in the queue using the chosen ordering policy and it searches for a new schedule. Finally, the task copies starting at time t are committed.

The complexity for one search for a schedule where N is the number of tasks in the task queue and P is the number of processors is as follows. The complexity to order a task queue is $O\left(N \log(N)\right)$ and the one to add a task in an already ordered queue is $O\left(N\right)$. Then, it takes $O\left(P \cdot N \cdot \textit{(\# task copies)}\right)$ to map and schedule tasks from the task queue and $O\left(1\right)$ to commit a task copy. If we consider that the task queue is always ordered, the overall worst-case complexity is $O\left(N + P \cdot N \cdot \textit{(\# task copies)} + 1\right)$.

Input: Mapping and scheduling of already scheduled tasks, (task t_i)
Output: Updated mapping and scheduling

```
 1: if there is a scheduling trigger at time $t$ then
 2:     if a processor becomes idle and there is neither task arrival
        nor fault occurrence then
 3:         if an already scheduled task copy starts at time $t$ then
 4:             Commit this task copy
 5:         else
 6:             Nothing to do
 7:     else ▷ processor is idle and task arrives and/or fault occurs
 8:         if a (simple or double) task $t_i$ arrives then
 9:             Add one or two $PC_i$ to the task queue
10:         if a fault occurs during the task $t_k$ then
11:             Add $BC_k$ to the task queue
12:         Remove task copies having not yet started their execution
13:         Order the task queue
14:         for each task in the task queue do
15:             Map and schedule its task copies (PC(s) or BC)
16:         if an already scheduled task copy starts at time $t$ then
17:             Commit this task copy
18:         else
19:             Nothing to do
```

Fig. 3: Principle of online algorithm scheduling all tasks as aperiodic tasks (OneOff)

*C. Online Scheduling Algorithm for All Tasks Scheduled as Aperiodic or Periodic Tasks (*OneOff&Cyclic*)*

The online algorithm scheduling arriving tasks as aperiodic or periodic tasks is called OneOff&Cyclic. It is aware that there are not only aperiodic tasks but also periodic ones. Therefore, there are two task sets: one for periodic tasks and one for aperiodic ones.

The principle of OneOff&Cyclic is summed up in Fig. 4.

First (Line 1), the algorithm is triggered (i) if a processor becomes idle, and/or if there is (ii) an arrival of aperiodic task(s), (iii) an arrival/withdrawal[4] of periodic task(s), or (iv) a fault during task execution.

In the case a processor becomes/is idle (Case (i)), a new search for a schedule is not carried out and task copies are committed using an already defined schedule (Lines 2-6). As there is no modification in task sets, the schedule of one hyperperiod, which is the least common multiple of task periods, is repeated until one of Cases (ii)-(iv) happens.

Otherwise (Lines 7-23), the task sets of periodic and aperiodic tasks are updated and all task copies, which have not yet started their execution, are removed from the former schedule. Afterwards (Lines 15-19), tasks are ordered and the algorithm schedules aperiodic tasks and periodic ones. Finally (Lines 20-23), the task copies starting at time t are committed.

Similarly to OneOff, we denote N_{aper} as the number of aperiodic task in the task queue and N_{per} as the number of task instances per hyperperiod of periodic tasks in the task queue. The overall worst-case complexity is $O\left(N_{aper} + P \cdot N_{aper} \cdot \textit{(\# task copies)} + N_{per} + P \cdot N_{per} \cdot \textit{(\# task copies)} + 1\right)$.

[3] The arrival time a_i equals $\phi_i + (k-1) \cdot T_i$ and the deadline d_i is computed as $a_i + T_i$. The execution time et_i and the task type tt_i are not modified.

[4] A possibility to add and withdraw a periodic task from the task set allows us to model sporadic tasks related to the communication between a CubeSat and a ground station. More details are presented in Section V.

```
Input: Mapping and scheduling of already scheduled tasks, (task $t_i$)
Output: Updated mapping and scheduling
if there is a scheduling trigger at time $t$ then
    if a processor becomes idle and there is neither arrival/withdrawal of periodic task nor arrival of aperiodic task nor fault occurrence then
        if an already scheduled task copy starts at time $t$ then
            Commit this task copy
        else
            Nothing to do
    else    ▷ processor is idle and there is a change in set of periodic or aperiodic tasks and/or a fault occurs
        if a periodic task $t_i$ arrives or is withdrawn then
            Add/withdraw one or two $PC_i$ to/from the queue of periodic tasks
        if an aperiodic task $t_i$ arrives then
            Add one or two $PC_i$ to the queue of aperiodic tasks
        if a fault occurs during the task $t_k$ then
            Add $BC_k$ to the queue of aperiodic tasks
        Remove task copies having not yet started their execution
        Order the task queues
        for each task in the task queue of aperiodic tasks do
            Map and schedule its task copies (PC(s) or BC)
        for each task in the task queue of periodic tasks do
            Map and schedule its task copies (PC(s) or BC)
        if an already scheduled task copy starts at time $t$ then
            Commit this task copy
        else
            Nothing to do
```

Fig. 4: Principle of online algorithm scheduling all tasks as periodic or aperiodic tasks (OneOff&Cyclic)

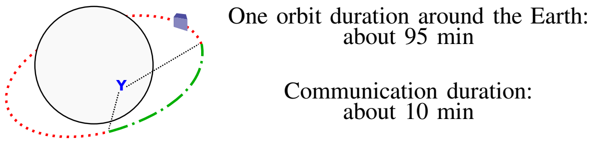


Fig. 5: Communication phase (green dot-and-dash line) and no-communication phase (red dashed line)

V. Experimental Framework

When a CubeSat orbits the Earth, two main phases can be identified from the scheduling point of view: *communication* and *no-communication phases*. During the no-communication phase (marked by red dashed line in Fig. 5), there is no communication between a CubeSat and a ground station and the CubeSat mainly executes periodic tasks associated with for example telemetry, reading/storing data or checks. If there is an interrupt due to an unexpected or asynchronous event, it is considered as an aperiodic task. When a communication with a ground station is possible, i.e. during the communication phase, periodic tasks related to the communication are executed in addition to the previously mentioned tasks.

The data used in our experimental framework are based on real CubeSat data provided by the Auckland Program for Space Systems (APSS)[5] and by the Space Systems Design Lab (SSDL)[6]. These data were gathered by functionality and generalised in order to generate more data for our simulations. They are respectively called Scenario APSS and Scenario RANGE and summarised in Tables I and II, where U denotes a uniform distribution and one hyperperiod is the least common multiple of task periods.

TABLE I: Set of tasks for Scenario APSS

Periodic tasks

Function	Task type	Phase ϕ_i	Period T_i	Execution time et_i	# tasks
Communication	D	U(0; T)	500 ms	U(1ms; 10ms)	2
Reading data	S	U(0; T)	1000 ms	U(100ms; 500ms)	10
Telemetry	D	U(0; T)	5000 ms	U(1ms; 10ms)	2
Storing data	S	U(0; T)	10000 ms	U(100ms; 500ms)	7
Readings	D	U(0; T)	60000 ms	U(1ms; 10ms)	2

Sporadic tasks related to communication

Function	Task type	Phase ϕ_i	Period T_i	Execution time et_i	# tasks
Communication	S	U(0; T)	500 ms	U(1ms; 10ms)	46

Aperiodic tasks

Function	Task type	Arrival time a_i	Execution time et_i	# tasks
Interrupts	D	U(0; 100000ms)	U(1ms; 10ms)	1

TABLE II: Set of tasks for Scenario RANGE

Periodic tasks

Function	Task type	Phase ϕ_i	Period T_i	Execution time et_i	# tasks
Kalman filter	D	U(0; T)	100 ms	U(1ms; 30ms)	1
Attitude control	D	U(0; T)	100 ms	U(10ms; 30ms)	1
Sensor polling	D	U(0; T)	100 ms	U(1ms; 5ms)	5
Telemetry gathering	S	U(0; T)	20000 ms	U(100ms; 500ms)	1
Telemetry beaconing	S	U(0; T)	30000 ms	U(10ms; 100ms)	2
Self-check	D	U(0; T)	30000 ms	U(1ms; 10ms)	5

Sporadic tasks related to communication

Function	Task type	Phase ϕ_i	Period T_i	Execution time et_i	# tasks
Communication	S	U(0; T)	500 ms	U(1ms; 10ms)	10

Aperiodic tasks

Function	Task type	Arrival time a_i	Exec. time et_i	# tasks
Interrupts, GPS	D	U(0; 10000ms)	U(1ms; 50ms)	10

In order to further analyse the algorithm performances (see Section VI), we also modified Scenario APSS. This scenario is called Scenario APSS-modified. Its tasks are the same as for Scenario APSS but the periods of 500 ms were prolonged to 1000 ms and periods longer than 5000 ms were shortened to 5000 ms. The number of tasks, whose periods were modified, per period were computed pro rata. Thus, the system load and the proportion of simple and double tasks for Scenarios APSS and APSS-modified are the same.

To model dynamic aspect, although task sets are defined in advance for simulations, they are unknown to the algorithms until discrete simulation time equals the arrival time (for aperiodic tasks) or phase (for periodic and sporadic tasks).

To evaluate the algorithms, 20 simulations of 2 hyperperiods were realised and the obtained values were averaged.

To compare our results, resolutions carried out in CPLEX solver[7] (described in Section IV-A) were computed based on

[5] https://space.auckland.ac.nz/auckland-program-for-space-systems-apss/

[6] http://www.ssdl.gatech.edu/

[7] https://www.ibm.com/analytics/cplex-optimizer

the same data set. To model real-time aspect (i.e. dynamic task arrival) in CPLEX solver, at each task arrival, the main function updates data (arrival/withdrawal of periodic task and/or arrival of aperiodic task) and launches a new resolution using the current data set. Due to computational time constraints in this case, only results for ONEOFF&CYCLIC were obtained and no fault was injected.

For simulations with fault injection, we take into account that the worst estimated fault rate in the real space environment is 10^{-5} fault/ms [18]. Therefore, we inject faults at the level of task copies with fault rate for each processor between $1 \cdot 10^{-5}$ and $1 \cdot 10^{-3}$ fault/ms in order to assess algorithm performances not only using the real fault rate but also its higher values. For the sake of simplicity, we consider only transient faults and that one fault can impact at most one task copy.

Regarding the metrics, we make use of the *rejection rate*, which is the ratio of rejected tasks to all arriving tasks, and the *system throughput*, which counts the number of correctly executed tasks. In a fault-free environment, this metric is equal to the number of tasks minus the number of rejected tasks. The *task queue length* stands for the number of tasks in the task queue, which are about to be ordered and scheduled. The algorithm run-time is measured by the *scheduling time*, which accounts for the time elapsed during one scheduling search.

VI. RESULTS

First, we compare the rejection rate for both algorithms. Second, we analyse the scheduling time of each ordering policy for both algorithms. Third, we evaluate the algorithm performances in the presence of faults.

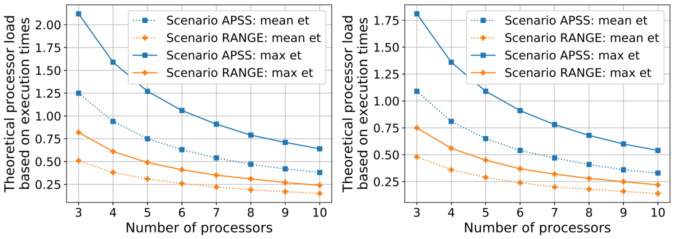
A. *Rejection Rate of* ONEOFF *and* ONEOFF&CYCLIC

We compare different ordering policies for three scenarios to choose which policy is the best in terms of the rejection rate. Before analysing separately the performances of ONEOFF and ONEOFF&CYCLIC, we focus on the system load and task proportions for each scenario in a fault-free environment.

Based on Tables I and II, we computed the theoretical processor load when considering both maximum and mean execution times of each task. We remind the reader that a simple task has one primary copy and a double task has two primary copies. The results are depicted in Fig. 6 representing the processor load respectively for both communication phases as a function of the number of processors.

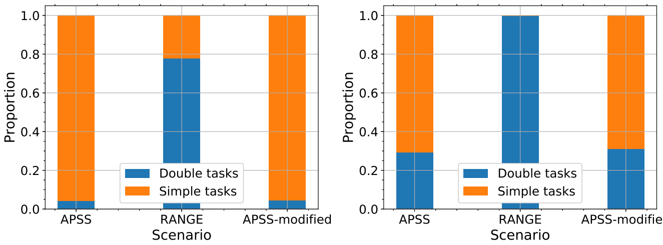
Scenario RANGE has lower theoretical processor load than other two scenarios no matter the communication phase. Theoretically, it means that all tasks for Scenario RANGE can be scheduled (maximum theoretical processor load is between 22% for 10-processor systems and 82% for 3-processor systems) while it is not always possible for Scenarios APSS and APSS-modified because the maximum theoretical processor load exceeds 100% when a CubeSat has only a few processors.

Regarding the proportion of simple and double tasks, they are represented in Fig. 7. It can be observed that during the communication phase the percentage of double tasks for Scenarios APSS and APSS-modified is low (about 4%) while



(a) Communication phase (b) No-communication phase

Fig. 6: Theoretical processor load when considering maximum and mean execution times (et) of each task



(a) Communication phase (b) No-communication phase

Fig. 7: Proportions of double and simple tasks

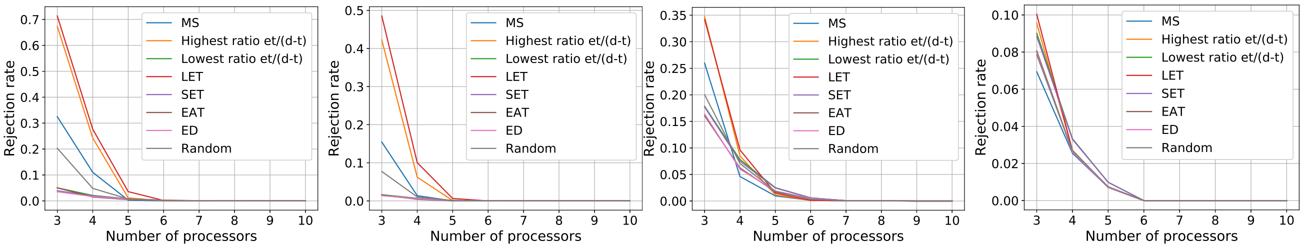
the task set for Scenario RANGE consists of 78% double tasks. During the no-communication phase, the percentage of simple tasks is almost negligible (0.02%) for Scenario RANGE and it is about 30% for other two scenarios.

To conclude, our experimental framework makes use of two very different sets of scenarios. On the one hand, Scenarios APSS and APSS-modified have high system load and high proportion of simple tasks compared to double tasks. On the other hand, Scenario RANGE contains mainly double tasks and has lower system load.

1) Analysis of ONEOFF*:* Fig. 8 show the rejection rate of Scenarios APSS and APSS-modified for both communication phases as a function of the number of processors. Scenario RANGE is not presented because the rejection rate is 0 regardless of ordering policy and communication phase. This is due to the task data set, which has rather low system load. We notice that the ”Earliest Deadline” or ”Earliest Arrival Time” techniques overall reject the least tasks.

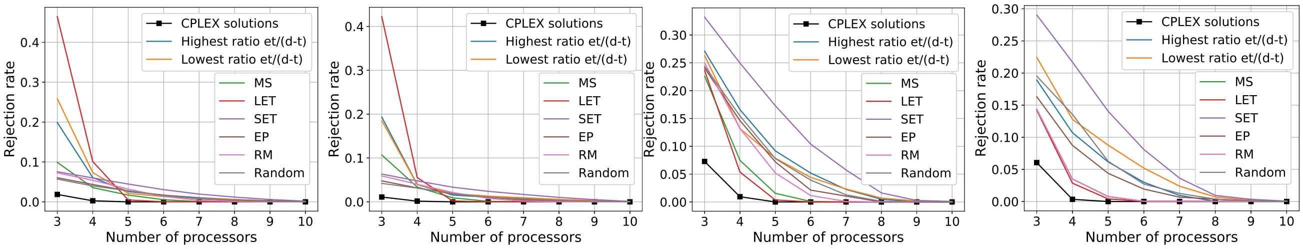
2) Analysis of ONEOFF&CYCLIC*:* Fig. 9 depict the rejection rate of Scenarios APSS and APSS-modified for both communication phases as a function of the number of processors. In these figures, we plot not only studied ordering policies but also a curve presenting the optimal solution provided by CPLEX solver. In general, the algorithm using the ordering policy achieving the lowest rejection rate has its competitive ratio of 2 or 3, which are rather good results taking into account that our search is not exhaustive compared to the search for optimal solution.

Scenario RANGE is again not presented because the rejection rate of all ordering policies no matter communication phase is close or equal to 0 (in general the rejection rate is less than 1%) due to lower system load. As for depicted scenarios,



(a) Scenario APSS; com. phase (b) Scenario APSS-mod.; com. ph. (c) Scenario APSS; no-com. phase (d) Scenario APSS-mod.; no-com. ph.

Fig. 8: Rejection rate of OneOff as a function of the number of processors



(a) Scenario APSS; com. phase (b) Scenario APSS-mod.; com. ph. (c) Scenario APSS; no-com. phase (d) Scenario APSS-mod.; no-com. ph.

Fig. 9: Rejection rate of OneOff&Cyclic as a function of the number of processors

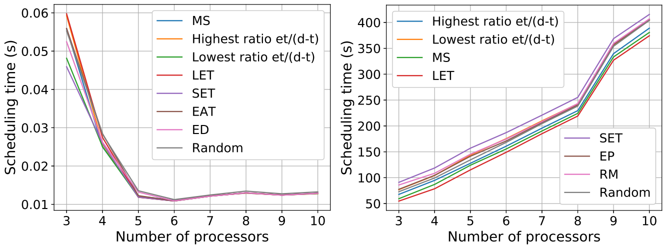
it is not so straightforward to determine one policy, which always performs well. A reasonable choice is the ”Minimum Slack” or ”Earliest Phase” during the communication phase and the ”Minimum Slack” or ”Longest Execution Time” during the no-communication phase. Altogether, the ”Minimum Slack” policy performs well regardless of communication phase. Nevertheless, the rejection rate of OneOff&Cyclic is in general higher than the one of OneOff.

3) Comparison of Different Scenarios: The performances of a given ordering policy are influenced by the system load and the task proportions. The influence of the former factor is illustrated by Scenario RANGE, which has much lower (or none) rejection rate than other two scenarios. The impact of the latter factor is demonstrated by the difference in the rejection rate for Scenarios APSS and APSS-modified. For several ordering policies, the rejection rate is higher during the no-communication phase than during the communication one despite the fact that there are less tasks during the no-communication phase. Actually, there are 29.4% double tasks during the no-communication phase against 4.2% double tasks during the communication phase.

In order not to oversize the system, it is useless to consider more than 6 processors because, when an ordering policy is well chosen, no task is rejected.

B. Comparison of Scheduling Time

In this section, we compare the scheduling time of OneOff and OneOff&Cyclic. First, we will analyse Scenario APSS and then Scenario APSS-modified. Scenario RANGE is not presented because the results of OneOff&Cyclic are qualitatively similar to the ones of Scenario APSS. As for the results of OneOff, there are several variations since the task queue length does not have significant differences for different ordering policies as for Scenario APSS.

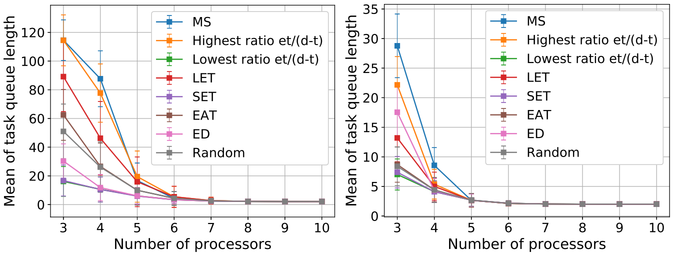


(a) OneOff (b) OneOff&Cyclic

Fig. 10: Scheduling time as a function of the number of processors (Scenario APSS; no-communication phase)

Fig. 10 represent the scheduling time of **Scenario APSS** for OneOff and OneOff&Cyclic during the no-communication phase as a function of the number of processors. The scheduling time during the communication phase is qualitatively similar to the ones in Fig. 10 but approximately 4 times longer for OneOff&Cyclic and 2 times longer for OneOff (when there is less than 5 processors). The communication phase takes more time to find a schedule than the no-communication phase because there are more tasks.

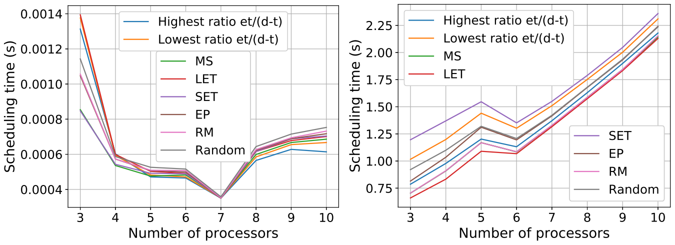
Moreover, there is no significant difference among ordering policies for OneOff while there is one for OneOff&Cyclic. The ordering policies that achieve the lowest scheduling time for OneOff are the ”Shortest Execution Time”, ”Lowest ratio of $et/(d-t)$” and ”Earliest Deadline”. As for OneOff&Cyclic, we point out the ”Longest Execu-



(a) Communication phase

(b) No-communication phase

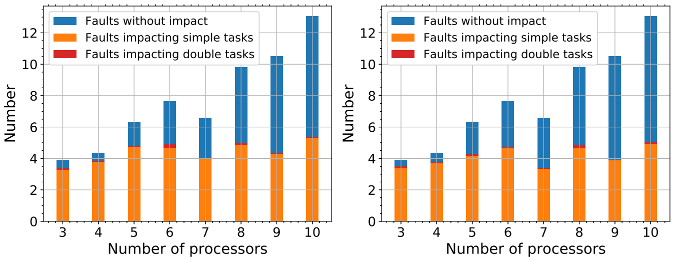
Fig. 11: Mean value of the task queue length with standard deviations as a function of the number of processors (OneOff; Scenario APSS)



(a) OneOff

(b) OneOff&Cyclic

Fig. 12: Scheduling time as a function of the number of processors (Scenario APSS-modified; no-communication phase)



(a) Communication phase

(b) No-communication phase

Fig. 13: Number of faults (injected with fault rate $1 \cdot 10^{-5}$ fault/ms) and their proportion respectively impacting simple and double tasks (OneOff; Scenario APSS)

tion Time”, ”Minimum Slack” and ”Highest ratio of $et/(d-t)$” techniques as the best ordering policies and the ”Shortest Execution Time” and ”Rate Monotonic” techniques as the worst ones in terms of the scheduling time.

The scheduling time is related to the algorithm complexity, which is defined in Sections IV-B and IV-C for OneOff and OneOff&Cyclic, respectively. One of the terms standing for the complexity is the number of tasks in the task queue. To show the trend of the task queue length, Fig. 11 depict the mean value of the task queue length with standard deviations during both communication phases for OneOff and Scenario APSS. We notice that the higher the number of processors, the shorter the task queue and that ordering policies have significant differences in the number of tasks in the task queue when a system has a low number of processors.

Consequently, the scheduling time of OneOff decreases with the higher number of processors because the task queue is shorter owing to more scheduling triggers. Regarding the scheduling time of OneOff&Cyclic, it raises when the number of processors increases even though the number of tasks is almost constant (the set of periodic tasks remains the same for a given phase and there is only one arrival of aperiodic task). The increase is due to more possibilities to be tested when a system has more processors.

Nonetheless, as the results are based on simulations (since real experiments are not easily feasible), the scheduling time in our experiments does not significantly change as the task queue length could foresee. This difference is due to the additional complexity related to our simulation framework (handling of arrays in time standing for schedules on processors), which will not be present in reality and the real scheduling time will be shorter.

Finally, the scheduling time of OneOff&Cyclic is roughly 5 orders of magnitude greater than the one of OneOff. This huge gap is mainly due to the significant difference in task periods: between 500 ms and 60000 ms. To better evaluate this impact on scheduling time, we modified Scenario APSS to Scenario APSS-modified, as described in Section V.

Fig. 12 represent the scheduling time of **Scenario APSS-modified** for OneOff and OneOff&Cyclic during the no-communication phase as a function of the number of processors. The trend of scheduling time during the communication phase is again similar to the ones in Fig. 12 and the values are multiplied by a number within the range from 5 to 10 for OneOff&Cyclic and by 2 for OneOff (when a system has less than 6 processors).

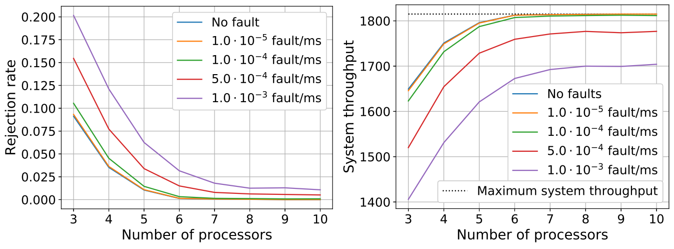
The scheduling time of OneOff&Cyclic is roughly 3 orders of magnitude greater than the one of OneOff. We conclude that the idea to reduce the substantial difference in the task periods accelerates the scheduling time. Therefore, we suggest to teams building CubeSats to avoid tasks with very short and very long periods to be scheduled together.

C. *Fault Injection*

In this section, we evaluate the fault tolerance of both algorithms for Scenario APSS. We consider the ”Earliest Deadline” policy for OneOff and the ”Minimum Slack” policy for OneOff&Cyclic.

Fig. 13 represent the number of faults (injected with fault rate $1 \cdot 10^{-5}$ fault/ms, which corresponds to the worst estimated fault rate in the real space environment [18]) and their proportion respectively impacting simple and double tasks as a function of the number of processors. Albeit only values for OneOff are shown, the ones for OneOff&Cyclic are similar. We remind the reader that presented results were computed as an average of 20 simulations and thereby they may not be integers.

The number of impacted tasks remains almost constant and there is no significant difference between two algorithms nor between communication phases. Furthermore, double tasks are



(a) Rejection rate (b) System throughput

Fig. 14: System performances at different fault injection rates as a function of the number of processors (ONEOFF; Scenario APSS; no-communication phase)

rarely impacted, which is due to their shorter execution time when compared with simple tasks. We also studied other fault rates (graphs not shown in this paper). As expected, the higher the fault rate, the more faults. Nonetheless, the proportions of impacted simple and double tasks remain the same.

Fig. 14 depict the rejection rate and system throughput for no-communication phase as a function of the number of processors. Qualitatively similar results were obtained for ONEOFF during the communication phase and for both phases of ONEOFF&CYCLIC. The higher the number of processors, the lower the rejection rate and the higher the system throughput because the number of tasks to be executed aboard the CubeSat is always the same for a given phase. The rejection rate stands for the schedulability as described in Section IV, i.e. if a fault occurs during a PC execution, the corresponding backup copy is scheduled. Nonetheless, a backup copy may be impacted by a fault too. Since such a task was not correctly executed, it does not contribute to the system throughput.

Moreover, the studied metrics do not change significantly up to $1 \cdot 10^{-4}$ fault/ms, which is higher than the worst estimated fault rate in the real space environment (10^{-5} fault/ms [18]). The same conclusions were made for other two scenarios (RANGE and APSS-modified) as well.

VII. CONCLUSION

This paper evaluated the performances of two online algorithms meant for CubeSats, which operate in the harsh space environment and are vulnerable to faults. To make CubeSats fault tolerant, these algorithms schedule all tasks aboard the CubeSat, detect faults and take appropriate measures in order to deliver correct results.

While the first algorithm (called ONEOFF) considers all tasks as aperiodic tasks, the second one (named ONEOFF&CYCLIC) distinguishes aperiodic and periodic tasks when searching for a new schedule. Each algorithm can use different ordering policies to sort a task queue. The presented results based on two real CubeSat scenarios show that it is useless to consider systems with more than six processors and that ONEOFF performs better than ONEOFF&CYCLIC in terms of the rejection rate and the scheduling time. ONEOFF&CYCLIC can be more efficient in applications where there are only a few changes in the set of periodic tasks. Therefore, we suggest that teams, which design their CubeSats gathering all processors together on one board, put into practice rather ONEOFF.

Finally, it was found that the studied algorithms perform well also in a harsh environment.

As our future work, we are about to further evaluate energy constraints, which play an important role to ensure real-time execution of tasks because a CubeSat spends one third of its orbit in the eclipse with limited power supply.

ACKNOWLEDGMENTS

The authors would like to thank the Auckland Program for Space Systems (APSS) and Space Systems Design Lab (SSDL) for sharing their CubeSat data.

REFERENCES

[1] NASA CubeSat Launch Initiative, "CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers," 2017, https://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf.

[2] Erik Kulu, "Nanosats Database," https://www.nanosats.eu/.

[3] Arizona State University, "Phoenix PDR," Presentation on March 24, 2017 at AMSAT-UK Colloquium 2014, 2017.

[4] K. A. LaBel, "Radiation Effects on Electronics 101: Simple Concepts and New Challenges," Presentation at NASA Electronic Parts and Packaging (NEPP) Webex Presentation, 2004.

[5] D. Burlyaev, "System-level Fault-Tolerance Analysis of Small Satellite On-Board Computers," Master's thesis, Delf University of Technology, 2012.

[6] D. Geeroms *et al.*, "ARDUSAT, an Arduino-Based CubeSat Providing Students with the Opportunity to Create their own Satellite Experiment and Collect Real-World Space Data," in *22nd ESA Symposium on European Rocket and Balloon Programmes and Related Research*, ser. ESA Special Publication, vol. 730, Sep 2015, p. 643.

[7] A. O. Erlank and C. P. Bridges, "Satellite Stem Cells: The Benefits & Overheads of Reliable, Multicellular architectures," in *2017 IEEE Aerospace Conference*, March 2017, pp. 1–12.

[8] K. Laizans *et al.*, "Design of the Fault Tolerant Command and Data Handling Subsystem for ESTCube-1 ," in *Proceedings of the Estonian Academy of Sciences*, 2014, pp. 222–231.

[9] A. Erlank and C. Bridges, "Reliability Analysis of Multicellular System Architectures for Low-Cost Satellites," in *Acta Astronautica*, vol. 147, 2018, pp. 183–194.

[10] W. U. of Technology, "PW-SAT 2 Preliminary Requirements Review: On-Board Computer," 2014.

[11] T. B. Clausen *et al.*, "Designing On Board Computer and Payload for the AAU CubeSat."

[12] L.-W. Chen, T.-C. Huang, and J.-C. Juang, "Implementation of the Fault Tolerance Module in PHOENIX CubeSat," Presentation at 10th IAA Symposium on Small Satellites for Earth Observation, 2015.

[13] S. Ghosh, R. Melhem, and D. Mosse, "Fault-Tolerance Through Scheduling of Aperiodic Tasks in Hard Real-Time Multiprocessor Systems," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 3, 1997, pp. 272–284.

[14] S. Wang *et al.*, "A Reliability-aware Task Scheduling Algorithm Based on Replication on Heterogeneous Computing Systems," in *Journal of Grid Computing*, vol. 15, no. 1, 03 2017, pp. 23–39.

[15] J. Mei *et al.*, "Fault-Tolerant Dynamic Rescheduling for Heterogeneous Computing Systems," in *Journal of Grid Computing*, vol. 13, no. 4, 2015, pp. 507–525.

[16] P. Dobiáš, E. Casseau, and O. Sinnen, "Fault-Tolerant Online Scheduling Algorithms for CubeSats," in *11th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures / 9th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM'20)*, 2020.

[17] Q. Zheng, B. Veeravalli, and C.-K. Tham, "On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs," in *IEEE Transactions on Computers*, vol. 58, no. 3, 2009, pp. 380–393.

[18] R. M. Pathan, "Real-Time Scheduling Algorithm for Safety-Critical Systems on Faulty Multicore Environments," in *Real-Time Systems*, vol. 53, no. 1, 2017, pp. 45–81.