# Fault-Tolerant Online Scheduling Algorithms for CubeSats

Petr Dobiáš, Emmanuel Casseau, Oliver Sinnen

# Fault-Tolerant Online Scheduling Algorithms for CubeSats

Petr Dobiáš, Emmanuel Casseau
*Univ Rennes, Inria, CNRS, IRISA*
Lannion, France
petr.dobias@irisa.fr

Oliver Sinnen
*Department of Electrical and Computer Engineering*
*University of Auckland*
Auckland, New Zealand

**Abstract**

CubeSats are small satellites operating in harsh space environment. In order to ensure correct functionality on board despite faults, fault tolerant techniques taking into account spatial, time and energy constraints should be considered. This paper presents a software-level solution taking advantage of several processors available on board. Two online scheduling algorithms are introduced and evaluated. The results show their performances and the trade-off between the rejection rate and energy consumption. Last but not least, it is stated that ordering policies achieving low rejection rate when using the algorithm scheduling all tasks as aperiodic are the "Earliest Deadline" and "Earliest Arrival Time". As for the algorithm treating arriving tasks as aperiodic or periodic tasks, the "Minimum Slack" ordering policy provides reasonable results.

**Index Terms**

CubeSats, Fault-tolerance, Online Scheduling, Multiprocessors

## I. Introduction

CubeSats are small satellites composed of several units (e.g. 1U, 2U, 3U or 6U) where each unit (1U) is a 10 cm cube, which can weigh up to 1.33 kg [1]. Thanks to this standardisation, which reduces costs and time, CubeSats have become popular and their number of launches rapidly increases [2]. CubeSats consist of several systems, such as on-board computer, electrical power system, communication system, attitude determination and control system, and payload. They are mainly used for scientific investigations, for example to detect earthquakes [3] or study urban heat islands [4].

CubeSats operate in harsh space environment and are exposed to charged particles and radiations, which cause transient effects, such as single event upsets, and long-term effects like total ionising dose [5]. Fault rate can be up to $10^{-2}$ fault per second [6]. Besides natural faults, there are also human faults due to for instance a bug in software or wrong commands [7]. All faults can generate errors, which may later cause a system failure [8]. Altogether CubeSats need to deal with faults to be able to achieve their mission.

Our aim is to make CubeSats more fault tolerant. We design scheduling algorithms dealing with all tasks (no matter the system) on board of any CubeSat or any small satellite. They are online real-time algorithms, which schedule all types of tasks (periodic, sporadic and aperiodic), detect faults and consequently take appropriate measures in order to deliver correct results. They are primarily designed for CubeSats making use of commercial-off-the-shelf (COTS) processors because it was shown [9] that the more COTS components in CubeSat, the lower probability of mission success.

This paper presents and compares two such algorithms. Since CubeSats have limited energy resources, we evaluate different simple ordering policies and choose only the one which satisfies the best our objective function, i.e. to minimise the rejection rate accounting for the ratio of rejected tasks to all arriving tasks.

As CubeSats have currently several systems on board and most of them has its own processor, our idea is to put all processors together on one board. This solution will not only reduce space and weight but also improve the fault tolerance. First, a protection against faults from radiation will be easier to put into practice, using for example shielding [10]. Second, as each processor can carry out any computation (which is the opposite of the standard when each system has its own dedicated processor), the designed algorithm can use spatial redundancy and a CubeSat will remain operational in case of permanent processor failure. The idea to put all processors on one board has already been successfully realised on board of ArduSat gathering 17 processors on one board [11].

The remainder of this paper is organised as follows. Section II summarises the related work and Section III introduced our system, task and fault models. Our algorithms are then described in Section IV. The experiment framework is presented in Section V and the results are analysed in Section VI. Section VII concludes the paper.

## II. Fault Tolerance in CubeSats

As there is no closely related work to fault tolerant scheduling for CubeSats, we summarise techniques that are used to make CubeSats fault tolerant. Nevertheless, we point out that not all CubeSats are fault tolerant owing to budget or time constraints [9].

If a fault tolerance is considered in CubeSats at all, it is rather implemented in hardware using redundancy [12]. For example, it was reported [13] that unfortunately 43% of CubeSats do not use any redundancy.

To deal with faults in CubeSats, watchdog timers [14] or data protection techniques, e.g. checksum or Hamming codes [15], are employed. Several CubeSat teams try to anticipate fault occurrence and they analyse error reports, scan important parameters, like power consumption, and then send update commands [7], [12], [15].

Although in other fields, software fault tolerant solutions are commonly used to improve the system reliability, such as checkpointing [16], task replication [17], task rescheduling [18] or task swapping [19], they are relatively rare on board of CubeSats. In our research, we were inspired by task replication, in particular by the primary-backup approach [20], [21] considering primary and backup copies to deal with faults.

## III. System, Fault and Task Models

The system is composed of $P$ interconnected identical processors[1] and it deals with tasks on board of a CubeSat. These tasks are mostly related to housekeeping (like sensor measurements), communication with ground station and storing or reading data from memory. When a task arrives, it is put into a sorted task queue.

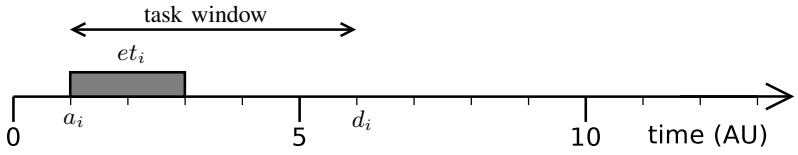



Figure 1: Model of aperiodic task $t_i$

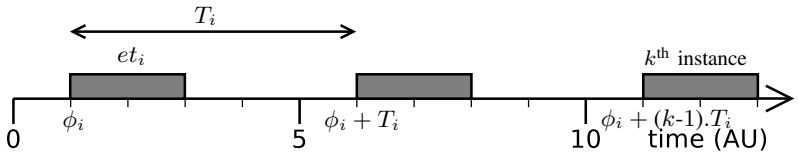



Figure 2: Model of periodic task $\tau_i$ [23]

Regarding our task model, we distinguish aperiodic and periodic tasks. An *aperiodic task* has four attributes: arrival time $a_i$, execution time $et_i$, deadline $d_i$ and task type $tt_i$ (defined in the next paragraph). A *periodic task* has several instances and is characterised by phase $\phi_i$ (i.e. arrival time of the first instance), execution time $et_i$, period $T_i$ and task type $tt_i$. We consider that the relative deadline is equal to the period. Aperiodic and periodic tasks are respectively depicted in Figures 1 and 2.

Our fault model considers both transient and permanent faults. Depending on the fault detection, we distinguish two task types: standard (S) and critical (C) tasks. *Standard tasks* are tasks, which are executed only once since a fault is detected by timeout, no received acknowledgement or failure of data checks. By contrast, a fault detection for *critical tasks* requires the execution of two identical task copies[2] and then their comparison because fault detection techniques for standard tasks may not be sufficient to detect a fault. In both cases, if a fault occurs during a task copy execution, a new task copy (called *backup copy* (BC)) is scheduled. To complete our definitions, task copies necessary to be executed in a fault-free environment are called *primary copies* (PC).

Our objective function 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. Algorithm Approaches

This section presents two scheduling algorithms and it starts with several principles applicable for both.

Arriving tasks to the system are ordered in a list using different policies, which will be listed later for each scheduling algorithm. A task is rejected and removed from the list if it does not meet its deadline. A preemption is not authorised.

As Figure 3 depicts, all task primary copies are scheduled as soon as possible in order to avoid idle processors just after task arrival and possible high processor load later. As our aim is to minimise the task rejection, the algorithm reserves a certain time of task window to place a backup copy if the primary copy execution was faulty. The limit between the primary copy scheduling window and the backup copy one is defined as $d_i$ - $\alpha.et_i$ for aperiodic tasks and $\phi_i + k.T_i$ - $\alpha.et_i$ for periodic tasks (with $\alpha \geqslant 1$). In this paper, we consider without lost of generality that $\alpha = 1$.

[1]For the sake of simplicity, a system presented in this paper consists of homogeneous processors. Nonetheless, the studied model can be extended to system with heterogeneous processors, such as in [22].

[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.

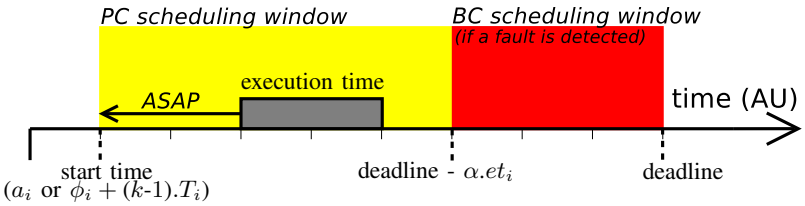



Figure 3: Principle of scheduling task copies

### A. *Online Scheduling Algorithm for All Tasks Scheduled as Aperiodic Tasks (Algorithm 1)*

The online algorithm scheduling arriving tasks as aperiodic ones is called Algorithm 1 in this paper. When it is put into service, all tasks are considered as aperiodic, which means that each instance of periodic task is transformed into an aperiodic task[3].

A search for a *new schedule is triggered* if i) a processor becomes idle, ii) a processor is idle and a task arrives, or iii) a processor is idle and a fault occurs. Then, all task copies, which have not yet started their execution, are removed from the former schedule. Finally, the algorithm schedules tasks.

As it is illustrated in Figure 4, our algorithm tests several policies to order task list and evaluates performances of each schedule. The schedule achieving the best performances in terms of the objective function, i.e. minimising the rejection rate, is chosen. The algorithm distinguishes accepted, rejected and pending tasks. A *pending task* is the task which is neither accepted nor rejected at time $t$ since there is no free slot to schedule it at time $t$ and there is still enough slack.

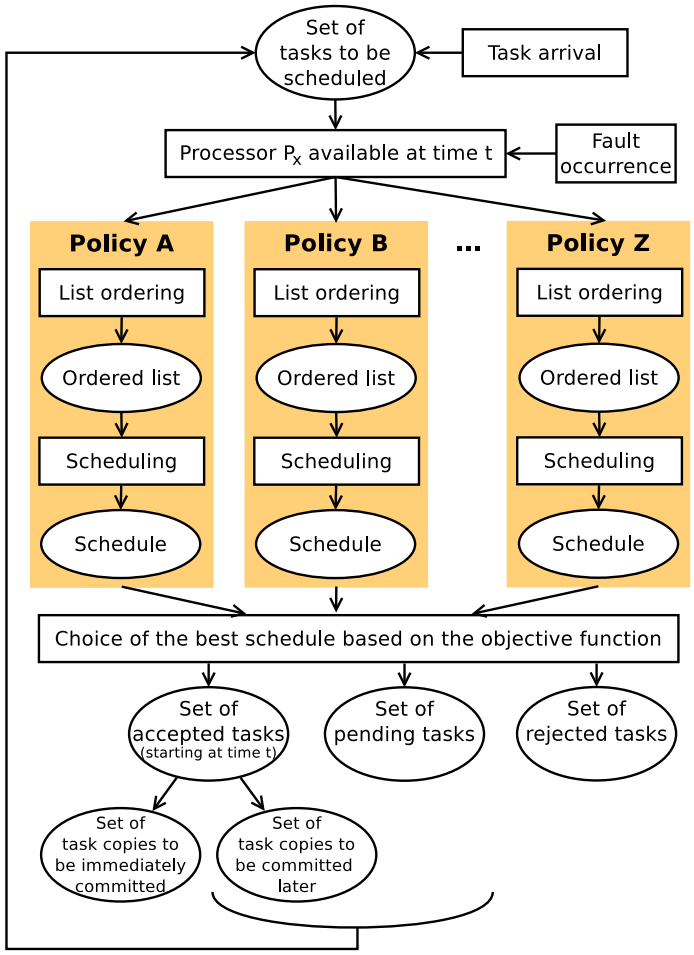



Figure 4: Principle of online algorithm scheduling all tasks as aperiodic tasks (Algorithm 1)

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

The ordering policies tested by this algorithm 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.

It can be shown that the worst-case complexity for one scheduling attempt is $P + OP.P.$*(# tasks).(# task copies)*$+OP.$*(# tasks)* where $P$: number of processors and $OP$: number of ordering policies.

### *B. Online Scheduling Algorithm for All Tasks Scheduled as Aperiodic or Periodic Tasks (Algorithm 2)*

The online algorithm scheduling arriving tasks as aperiodic or periodic tasks is depicted in Figure 5 and called Algorithm 2. It takes into account that some tasks are aperiodic and some are periodic.

A search for a *new schedule is triggered* if there is i) an arrival of aperiodic task(s), ii) an arrival/withdrawal[4] of periodic task(s), or iii) a fault during task execution. Then, all task copies, which have not yet started their execution, are removed from the former schedule. Afterwards, the algorithm schedules aperiodic tasks and, finally, it schedules periodic tasks.

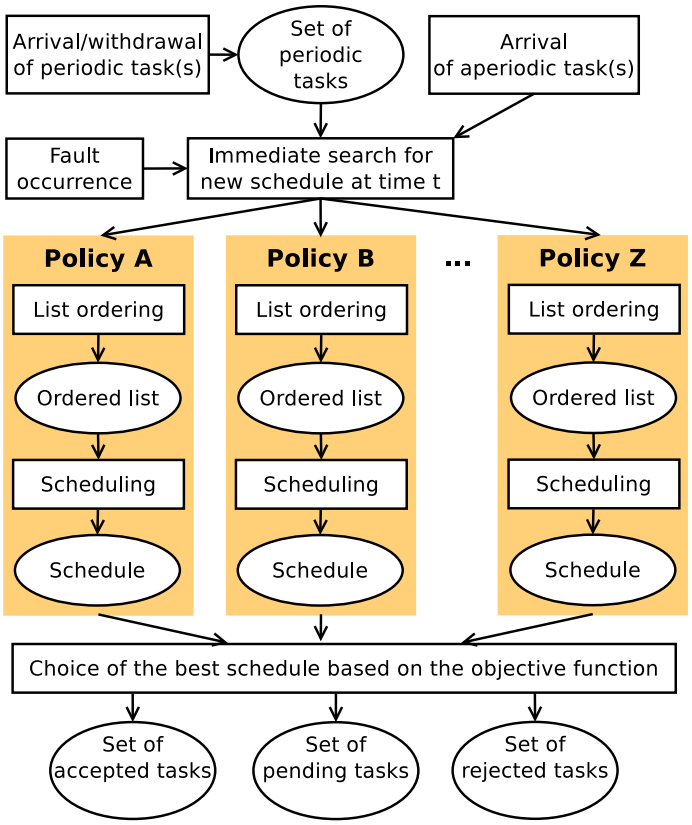



Figure 5: Principle of online algorithm scheduling all tasks as aperiodic or periodic tasks (Algorithm 2)

If there is *no scheduling trigger*, the schedule of one hyperperiod is repeated until a new scheduling trigger.

Similarly to the previous algorithm, several ordering policies are put into practice and the one, which satisfies the best the objective function, is chosen. The ordering policies used by this algorithm 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[5] (RM).

The worst-case complexity for one scheduling attempt is $P+OP.P.$*(# task instances per hyperperiod).(# task copies)*$+OP.$*(# tasks)*.

## V. Experiment Framework

On board of CubeSats, two main phases can be distinguished from the scheduling point of view: phases with and without communication between a CubeSat and a ground station. During the phase without communication, a CubeSat mainly executes periodic tasks related for example to telemetry, reading/storing data or checks. If there is an interrupt due to an unexpected

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

[5]Since the relative deadline is equal to the period, the *rate monotonic* ordering policy is equivalent to the *shortest period first* one.

event, it is considered as an aperiodic task. When a communication with a ground station is possible, periodic tasks related to communication are executed in addition to previously mentioned tasks.

In general, one CubeSat orbit around the Earth takes about 95 minutes. A communication between a CubeSat and a ground station lasts approximately 10 minutes per one orbit but it may happen that there is no communication due to CubeSat trajectory.

The data used in our experiment framework are based on real CubeSat data provided by the Auckland Program for Space Systems (APSS)[6]. These data were gathered by functionality and generalised in order to generate more data for our simulations. They are summarised in Table I, where $U$ accounts for a uniform distribution and one hyperperiod is the least common multiple of task periods, i.e. 60000 $ms$ in the studied case. Consequently, there are 6431 independent tasks per hyperperiod during the phase with communication and 911 independent tasks per hyperperiod during the phase without communication.

To analyse the presented algorithms, 20 simulations of 2 hyperperiods were carried out and the obtained values were averaged. There was no fault injection in the simulation scenario, so only primary copies were executed.

Table I: Set of tasks

**Periodic tasks**

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

**Sporadic tasks related to communication**

| Function | Task type | Phase $\phi_i$ | 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 | C | U(0; $100000ms$) | U($1ms$; $10ms$) | 1 |

As for the metrics, we make use of the *rejection rate*, which is the ratio of rejected tasks to all arriving tasks, and the *processor load*. The algorithm run-time is evaluated by the *number of scheduling searches*, i.e. how many times a search for a new schedule was carried out. This metric is important for CubeSats because it is related to the energy consumption.

## VI. Results

We first compare the two algorithms and then we analyse performances of each ordering policy in order to choose one, which is the most efficient from the point of view of the rejection rate. Once a choice is made, other ordering policies will not be considered any more during scheduling, which will reduce the algorithm run-time.

### A. Comparison of Scheduling Algorithms

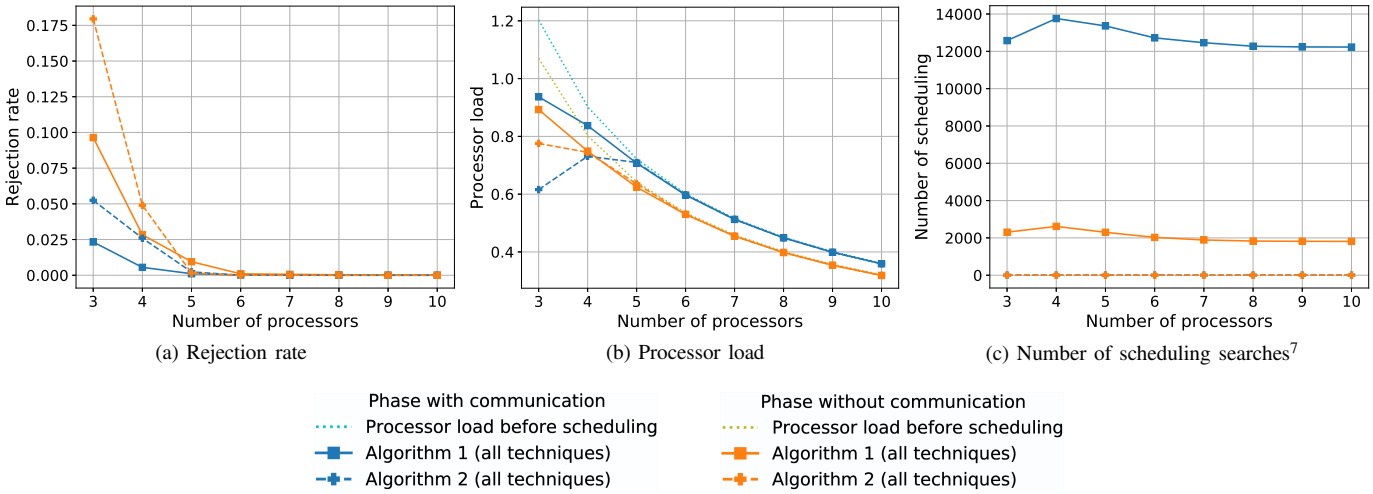
In this section, we compare Algorithms 1 and 2. When searching for a new schedule, we consider that all ordering policies respectively listed in Sections IV-A and IV-B and their combination are tested and the schedule minimising the rejection rate is chosen at each scheduling trigger.

Figures 6a, 6b and 6c respectively show the rejection rate, the processor load and the number of scheduling searches as a function of the number of processors for phases with and without communication. Since the number of tasks to be executed on board of a CubeSat is always the same for a given phase, the higher the number of processors, the lower the rejection rate and the lower the processor load. Furthermore, the rejection rate of Algorithm 1 is generally significantly lower than the one of Algorithm 2, especially when the number of processors is low. Actually, Algorithm 1 is more efficient due to its frequent scheduling searches.

It can also be seen that tasks are more rejected during the phase without communication. This phenomenon, which may seem contradictory since there are less tasks during the no communication phase, is explained by the fact that there are 29.4% of critical tasks during the phase without communication against 4.2% of critical tasks during the phase with communication. We remind the reader that critical tasks must have two primary copies.

In Figure 6b, two additional curves represent the processor load computed from data before scheduling. The values therefore account for the maximal processor load, i.e. in case of no task rejection. For both scheduling algorithms, we conclude that a CubeSat dealing with our data requires at least 6 processors to be able to satisfy all demands due to real time and spatial constraints.

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




(a) Rejection rate (b) Processor load (c) Number of scheduling searches[7]

Figure 6: Comparison of the online algorithm scheduling all tasks as aperiodic ones (Algorithm 1) and the online algorithm treating all tasks as aperiodic or periodic tasks (Algorithm 2)

Figure 6c points out that the number of scheduling searches of Algorithm 1 is significantly higher compared to Algorithm 2. The former algorithm has the number of scheduling searches at most equal to and in general lower than the sum of the number of tasks to be scheduled and the number of task copies. The latter algorithm has only two searches (no matter whether there is a communication or not) since there are only two scheduling triggers in our simulations: the first one takes place at the beginning when periodic tasks arrive and the second one is when an aperiodic task arrives.

In order to reduce the number of scheduling searches for Algorithm 1, we modified the algorithm, which was not triggered by each task arrival but by the multiple of several task arrivals, e.g. 5 or 10. The results (not presented in this paper) showed that when this number decreases, the rejection rate significantly increases.

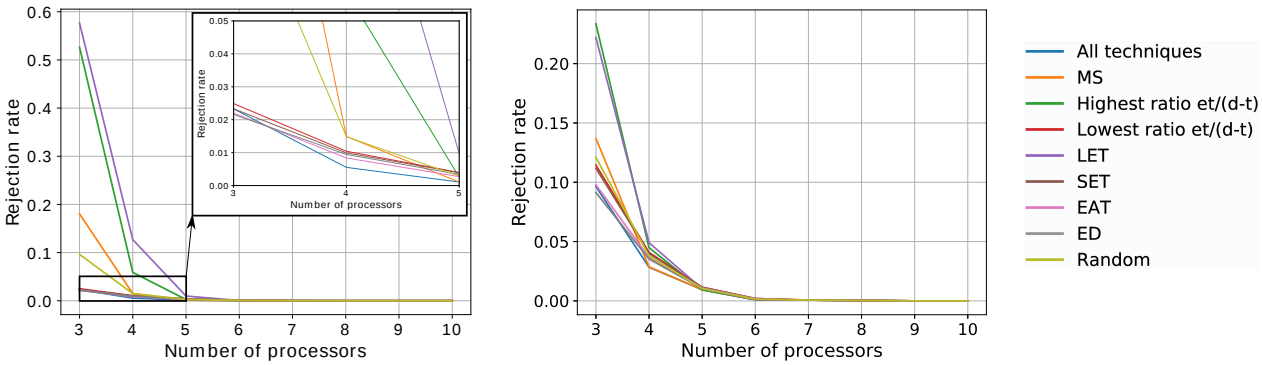
To sum up, the choice of algorithm depends on a trade-off between the rejection rate and the energy consumption. On the one hand, Algorithm 1 has lower rejection rate but at the cost of higher energy consumption due to frequent scheduling. On the other hand, Algorithm 2 has higher rejection rate but its energy consumption is lower since an already determined schedule of periodic tasks can be repeated until a new scheduling trigger.

### *B. Scheduling Algorithm for All Tasks Scheduled as Aperiodic Tasks*

In this section, we compare different ordering policies when using Algorithm 1. The aim is to analyse their performances in order to choose one ordering policy which rejects the least number of tasks. This one will be then chosen and considered, which will reduce the number of computations during scheduling.

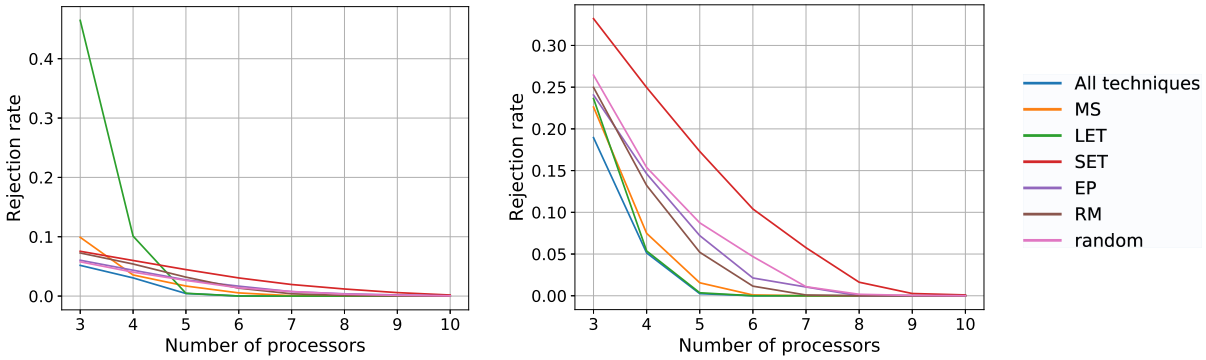
Figures 7a and 7b respectively show the rejection rate for both communication and no communication phases as a function of the number of processors. When an ordering policy is mentioned in the legend it means that it is exclusively used by the algorithm and no other ordering policy is considered, whereas ”All techniques” signifies that all ordering policies listed in Section IV-A were tested to find a schedule at every scheduling trigger and the one, which satisfies the best the objective function, is chosen.

We can again conclude that the higher the number of processors, the lower the rejection rate, and that the rejection rate is higher during the communication phase than during the phase without communication because the theoretical processor load due to task copies is higher. We also state that the ”Longest Execution Time” and ”Highest ratio of $et_i$ to ($d_i$-$t$)” ordering policies do not perform well because the number of rejected tasks is the highest. Although there are several ordering policies having the similar values during the communication phase (zoom in Figure 7a), Figure 7b representing results during the no communication phase shows that ”All techniques”, ”Earliest Deadline” and ”Earliest Arrival Time” ordering policies reject the least tasks. It can be seen that none stand-alone ordering policy achieves the best results (from the objective function point of view) for all processors during both communication and no communication phases. Nevertheless, the ”Earliest Deadline” and ”Earliest Arrival Time” techniques provide satisfactory results, i.e. very close to the ”All techniques”, which at each scheduling trigger compares schedules of all ordering policies before choosing the best one.




(a) Communication phase (b) No communication phase

Figure 7: Rejection rate when using the algorithm scheduling arriving tasks as aperiodic tasks




(a) Communication phase (b) No communication phase

Figure 8: Rejection rate when using the algorithm scheduling arriving tasks as aperiodic or periodic tasks

### *C. Scheduling Algorithm for All Tasks Scheduled as Aperiodic or Periodic Tasks*

Analogously to the previous section, we study different ordering policies when using Algorithm 2. Figures 8a and 8b respectively depict the rejection rate for both communication and no communication phases as a function of the number of processors.

First of all, we notice that results are more heterogeneous than previously and that for example the ”Longest Execution Time” policy does not perform well during the communication phase but it excels during the no communication phase. If we want to choose one ordering policy having reasonable performances all the time, we may opt for the ”Minimum Slack” technique, even if its results are not so close to the ones obtained when using ”All techniques”.

## VII. Conclusion

CubeSats operate in harsh space environment and are vulnerable to faults. To make CubeSats fault tolerant, we propose to take advantage of their multicore architecture and implement a solution in software. This paper presented two online algorithms, which schedule all tasks on board of a CubeSat, detects faults and takes appropriate measures in order to deliver correct results.

First, the paper evaluated two online scheduling algorithms: the algorithm considering all tasks as aperiodic tasks (Algorithm 1) and the one treating them as aperiodic or periodic tasks (Algorithm 2). It was shown that their performances vary, particularly when the number of processors is low, and that a choice is subject to a trade-off between the rejection rate and the energy consumption.

Then, we separately analysed them to determine which ordering policy is the most suitable for each algorithm in order not to test all policies and waste computations. It was found that the algorithm scheduling tasks as aperiodic works very well using the ”Earliest Deadline” or ”Earliest Arrival Time” technique. The analysis of the algorithm scheduling arriving tasks as aperiodic or periodic tasks indicated that the ”Minimum Slack” ordering policy provides reasonable results.

Although the studied algorithms was presented in the context of CubeSats, they can be also implemented in other applications, such as embedded systems having real-time and energy constraints.

[7]Algorithm 2 achieves the same results for phases with and without communication. Their curves are consequently overlapping.

As our future work, we intend to study further energy constraints, inject faults during simulations and therefore assess performances in environment, which will be very close to practical one.

## ACKNOWLEDGEMENT

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

## REFERENCES

[1] NASA CubeSat Launch Initiative, "CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers," 2017. [Online]. Available: https://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf
[2] Erik Kulu, "Nanosats Database." [Online]. Available: https://www.nanosats.eu/
[3] T. Bleier *et al.*, "QuakeSat Lessons Learned: Notes from the Development of a Triple CubeSat," Tech. Rep., 2014. [Online]. Available: https://www.quakefinder.com/pdf/Lessons_Learned_Final.pdf
[4] Arizona State University, "Phoenix PDR," Presentation on March 24, 2017 at AMSAT-UK Colloquium 2014, 2017. [Online]. Available: http://phxcubesat.asu.edu/sites/default/files/general/phoenix_pdr_part_2_1.pdf
[5] K. A. LaBel, "Radiation Effects on Electronics 101: Simple Concepts and New Challenges," Presentation at NASA Electronic Parts and Packaging (NEPP) Webex Presentation, 2004, https://nepp.nasa.gov/docuploads/392333B0-7A48-4A04-A3A72B0B1DD73343/Rad_Effects_101_WebEx.pdf.
[6] 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.
[7] 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, https://www.dlr.de/iaa.symp/Portaldata/49/Resources/dokumente/archiv10/pdf /0604_IAA-Li-Wei-Chen.pdf.
[8] I. Wali, "Circuit and System, Fault Tolerance Techniques," Ph.D. dissertation, Universit de Montpellier, 2016.
[9] 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.
[10] D. Burlyaev, "System-level Fault-Tolerance Analysis of Small Satellite On-Board Computers," Master's thesis, Faculty of Electrical Engineering, Mathematics and Computer Science, Delf University of Technology, 2012.
[11] 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.
[12] 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.
[13] A. Erlank and C. Bridges, "Reliability Analysis of Multicellular System Architectures for Low-Cost Satellites," in *Acta Astronautica*, vol. 147, 2018, pp. 183–194.
[14] "PW-SAT 2 Preliminary Requirements Review: On-Board Computer," 2014, warsaw University of Technology, https://pw-sat.pl/wp-content/uploads/2014/07/PW-Sat2-A-04.00-OBC-PRR-EN-v1.1.pdf.
[15] T. B. Clausen *et al.*, "Designing On Board Computer and Payload for the AAU CubeSat." [Online]. Available: http://www.space.aau.dk/cubesat/dokumenter/article.pdf
[16] M. Singh, "Performance Analysis of Checkpoint Based Efficient Failure-Aware Scheduling Algorithm," in *International Conference on Computing, Communication and Automation (ICCCA)*, 2017, pp. 859–863.
[17] 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.
[18] 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.
[19] A. Naithani, S. Eyerman, and L. Eeckhout, "Optimizing Soft Error Reliability Through Scheduling on Heterogeneous Multicore Processors," in *IEEE Transactions on Computers*, vol. 67, no. 6, 2018, pp. 830–846.
[20] 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.
[21] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu, "Fault-Tolerant Scheduling for Real-Time Scientific Workflows with Elastic Resource Provisioning in Virtualized Clouds," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, 2016, pp. 3501–3517.
[22] 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.
[23] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2011.