POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Hardware-in-the-Loop Simulation of Nano-satellite Constellations

Laurea Magistrale in Computer Science and Engineering - Ingegneria informatica

Author: Mattia Siriani

Advisor: Prof. Luca Mottola

Academic year: 2022-2023

1. Overview

With the rising costs of regular satellite operations, together with the issues intrinsic in planning, scheduling, and operating large space missions, *nano-satellites* increasingly represent a viable alternative for a number of space applications due to their reduced cost and easier scheduling. De-facto standards for nano-satellite designs, such as CubeSats [3], are simultaneously emerging as a mobile computing platform, enabling both education and research on space systems with reasonable costs and timing.

Nano-satellites may be deployed as stand-alone devices or in groups, forming *constellations* that operate as a single system. The latter deployment configuration enables, for example, higher coverage, better resilience to individual hardware or software faults, and shorter revisit times. These features are key for space applications such as Earth imaging, weather monitoring, and telecommunications [5].

Developing nano-satellite constellations is a multi-disciplinary effort that spans mechanical engineering, computer engineering, and control. Many of the related challenges boil down to two issues: *i)* the extremely *limited energy* envelope available to nano-satellites, which mainly rely on solar energy harvesting, and *ii)* the *harsh conditions* found in outer space, which possibly cause all sorts of faults in hardware and software alike. These issues complicate every phase of development, from the early designs all the way to experimentation and testing.

Simulators play a crucial role in the latter phase, due to difficulties in carrying out these activities in the target environment. Existing simulators concentrate on specific issues, for example, orbital mechanics or wireless transmissions in space. A common simulator of wireless transmission in space is NS-3, which is an open-source networks simulator, designed to assist in the analysis of various networking protocols and scenarios [4]. Instead, a standard orbital simulator is STK, which is a physical simulator able to simulate the whole space mission, from simulating the communications between the devices used in the space mission to simulating the trajectories of the deployed devices [1].

We concentrate on constellations of nano-satellites that offer sensing, computing, and communication functionality, effectively providing an orbital *edge computing platform* [2]. Understanding the energy trade-offs between the processing of different application scenarios is relevant to dictating the scope of a space mission, therefore we aim to implement a system that is able to simulate the specific space mission scenario desired by the user for a constellation of nano-satellites.

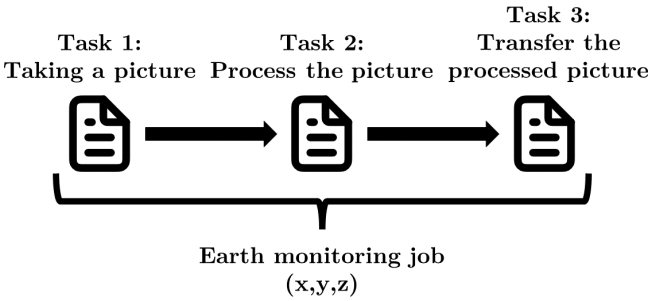

Figure 1: Tasks and jobs.

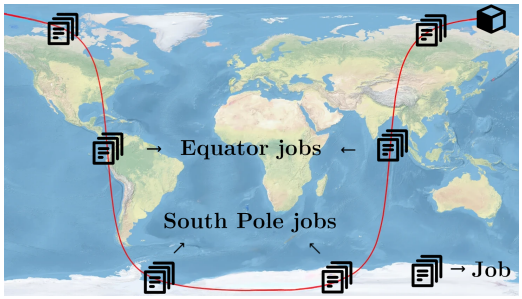

Figure 2: Tasks and jobs in orbit.

However, space simulators have one problem in common: they do not rely on the energy consumption data of real nano-satellites. This leads to the development of simulations that cannot realistically predict the behavior of the devices involved in the simulated space mission.
Our efforts focus on enhancing the realism of our system by implementing a simulation that interacts with a real device in real time. In the literature, this type of simulation is also referred to as *Hardware-In-the-Loop (HIL) simulation.*

2. Design and Implementation

For the development of our system, we take as a basis the *cote* orbital simulator [2]. Our contribution consists of a simulation of a constellation of nano-satellites, which enhances its realism by retrieving data directly from a real nano-satellite and simulating the user's desired space mission scenario. To provide further realism, the system can be configured to simulate radiation-induced errors along the orbit, directly on the real nano-satellite. Our system is scalable and able to interact with multiple physical nano-satellites simultaneously, decreasing the execution time of the simulation.

2.1. Components and Communication

The specification of the mission scenario desired by the user is simulated by two important components, which are tasks and jobs, described in Section 2.1.1. Furthermore, our system is divided into two main components: the simulation and the nano-satellite, analyzed correspondingly in Section 2.1.2 and in Section 2.1.3. These two main components are interconnected by multiple communication channels, illustrated in Section 2.1.4.

2.1.1 Jobs and Tasks

To enhance the realism provided in our system, we let the user define his desired space mission scenario. The constructs that fulfill this goal are the jobs and their sub-components, i.e. tasks, which are shown in Figure 1. A job is the objective that a nano-satellite in a space mission needs to fulfill in a specific position along the orbit. A task is a real program that is executed by the physical nano-satellite, and the execution of one task or the consecutive execution of multiple tasks forms a job.
The ordering of how the tasks are executed is defined by the utilization of two mechanisms: the specification of dependencies between tasks and the application of a priority policy. The output of a task could be used as input for the next task of the job, as we can see in the example of Figure 1. For instance, in Figure 1 we can see a possible subdivision of the tasks of an Earth monitoring job.
An important aspect of every task is that its simulation involves the retrieval of the data recorded on the physical nano-satellite during the execution of the task. A subset of these data is composed of non-functional data, which represents the voltage and current retrieved from the Power Supply Unit (PSU), from the battery, and from the solar panels mounted.
The simulation organizes different jobs on distinct orbital positions of each nano-satellite, as shown in Figure 2. This job structure is already provided by *cote*. In Figure 2, the cube represents the nano-satellite and the other icons represent the jobs. This structure of the jobs enables the system to simulate the same or multiple different objectives in the various orbital positions. For instance, in Figure 2 when the nano-satellite has an orbital position close to the Equator it can simulate a job regarding the monitoring of the temperature and when the orbital

position is close to the South Pole it can simulate a job regarding ice observations.

2.1.2 Simulation

For the simulation, we modify the *cote* simulator to convert it into a HIL simulator, leaving unaltered the orbit calculations and the modeling of the nano-satellite components. We structure the nano-satellites in the constellation, which we refer to as "simulated nano-satellites", as they are the simulated counterparts of the physical nano-satellite. Therefore, each simulated nano-satellite must interact with a physical nano-satellite to be simulated, as shown in Figure 3.
In our system we must deal with two important aspects related to the integration of the physical nano-satellite inside the simulation:

1. **Coordination**: This aspect is related to the coordination between the simulation and the physical nano-satellite on the job to simulate. To achieve this, a priori of the execution of the simulation, the system generates and transfers to the physical nano-satellite a file, which coordinates the two artefacts on the jobs to execute.
2. **Synchronization**: The physical nano-satellite works in real-time, while the simulation is faster; but, they must work in conjunction to ensure seamless operation. Our approach is to temporarily freeze the simulation whenever a simulated nano-satellite requests the processing of a job. This pause enables the physical nano-satellite to execute the different tasks of the job requested and send to the simulation the real data retrieved. Once the simulation receives all the data for the simulated nano-satellite, the simulation is unfrozen and can proceed with its execution. The scalability we provide to our system, described in Section 2.4, improves the synchronization of the two artefacts.

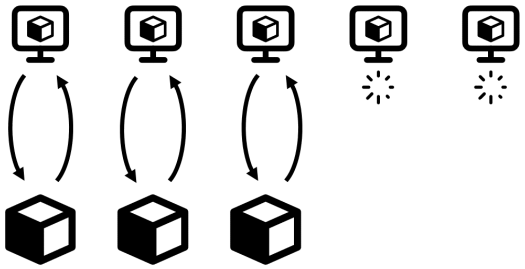

Figure 3: Scalability architecture.

2.1.3 Nano-satellite

For the physical nano-satellite involved in our HIL simulation, we use the CubeSat Simulator (CubeSatSim). The CubeSatSim is an open-source, cost-effective device, which is able to emulate the functionality of a standard CubeSat. The CubeSatSim is equipped with a Raspberry PI, which is interconnected to the other components of the CubeSatSim, from which it retrieves the data recorded from the different sensors mounted. These sensors record data from the environment and from the behaviors of the nano-satellite, providing fundamental data for the development of our HIL simulation.
The data retrieved from the physical nano-satellite is used to simulate a job regarding the objective of a space mission scenario.
Lastly, we reason about the fidelity of the energy consumption data that we retrieve from the physical nano-satellite. We focus on gathering a higher amount of data because, in this way, there is a higher probability of retrieving current peaks unseen by collecting fewer data.

2.1.4 Communication

The two main components of our system are interconnected by two communication channels, which are used at different times of the simulation. We designed the communication channels with platform-independent protocols, avoiding tying ourselves to the hardware in use. Furthermore, the communication channels we used are reliable because we want to guarantee that both artefacts receive what they need. The first communication channel is a unidirectional channel used to coordinate the simulation with the physical nano-satellite before the beginning of the simulation. As a protocol for this communication channel, we use SSH because it is platform-independent and reliable.
The latter communication channel is a bidirectional communication channel used by the simulation to request the execution of the tasks of the different jobs and retrieve the real data in response. As a protocol for this communication channel, we use MQTT because it is platform-independent, reliable, scalable, enables bidirectional communication, and is lightweight, making it ideal for our purposes since we are using embedded devices.

2.2. User Configuration

While developing our HIL simulation, we focus on making the system as user-configurable as possible, allowing the end user to configure various aspects of the simulation.

- **Simulation parameters**: The user can configure parameters regarding the functioning of the simulation, the nano-satellite, and the communication channels. Examples include the number of simulated nano-satellites, the simulation of radiation errors, and the topics used for MQTT communication.
- **Tasks and jobs**: The user must provide the programs that are used as tasks, as it is the user who determines the desired space mission scenario. For instance, if the user needs to simulate an Earth monitoring scenario, he must provide the programs needed to execute Earth monitoring activities, such as those in Figure 1. The user must compose the jobs with the different tasks he provides, controlling their order, and then he must assign the jobs to the simulated nano-satellites. Lastly, the user can control the number of jobs that each simulated nano-satellite simulates.
- **Changing nano-satellite**: The generality, provided by the design choices we made, is broad to such an extent that it is possible to modify the physical nano-satellite in use. The only steps required to do this operation are: *i)* provide hardware-dependent software to record the real data from the new nano-satellite and *ii)* modify the task component to support the new data structure. At this point, the nano-satellite is integrated into the system and can start working without further modification to the rest of the system.
  Therefore, the following aspects of the simulation remain unchanged: *i)* the interaction between the simulation and the nano-satellite, *ii)* the initial configuration of the simulation, *iii)* the functioning mechanism of the simulation, *iv)* the use and configuration of the tasks and jobs, *v)* the simulation of radiation-induced errors, *vi)* the configuration capabilities provided to the user, and *vii)* the scalability provided to the system.

2.3. Radiation-Induced Errors

Simulating radiation-induced error is not trivial because it involves accurately modeling the intricate interactions between radiation and the nano-satellite components. However, this obstacle becomes manageable by using the HIL simulation approach, leveraging the physical nano-satellite to directly simulate radiation-induced errors. Our approach involves the random corruption of the tasks before their execution. Specifically, we randomly bit-flip or corrupt the memory addresses of a task. In this way, we simulate a radiation error occurring during the objective mission. The system provides insights into how the physical nano-satellite involved in the system supports corruption errors.

2.4. Scalability

The use of a single physical nano-satellite can introduce bottlenecks in the simulation caused by our approach to the synchronization issue, described in Section 2.1.2. To enhance the scalability of our system, we can address both the nano-satellite side and the simulation side. In Figure 3, we can see the working mechanism of our scalability approach, where the elements at the top represent the simulated nano-satellites, while the elements at the bottom represent the physical nano-satellites.
Our approach on the physical nano-satellite side, shown at the bottom of Figure 3, involves using multiple physical nano-satellites. Our approach on the simulation side, shown at the top of Figure 3, focuses on parallelizing the execution of the simulated nano-satellites. For instance, if a simulated nano-satellite of the constellation has to process a long job, the other simulated nano-satellites are not forced to wait for the job execution to end, but they can continue with their own simulation. The two approaches combined enable the simulation to communicate with multiple physical nano-satellites simultaneously, as we can see in the center of Figure 3.

2.5. Implementation Highlights

We now focus on some implementation highlights of our system.

2.5.1 HIL Simulation

By using the *cote* simulator as a starting base, our system is written in C++. Before the ex-

ecution of the simulator, the simulation coordinates with the physical nano-satellite, using a file called mapper. The mapper shows the allocation of the tasks into the different jobs and how these jobs are distributed among the various simulated nano-satellites, following the configuration of the user. Furthermore, once the simulation starts, whenever it asks and then retrieves the real data from the physical nano-satellite, it dynamically integrates this data into the corresponding task component for the simulated nano-satellite that requests the processing of the job. In conclusion, the simulation outputs the results of all the events occurring for the simulated nano-satellites, which are used to generate different plots useful for the user to visualize the simulation results.

2.5.2 Nano-satellite

The nano-satellite has two main objectives: interacting with the simulation and recording the real data during the execution of the tasks requested. The first objective is accomplished by a single program, which receives the request to process the jobs from the simulation, starts the executions of the tasks, and for each task sends back to the simulation the data recorded. The latter objective is achieved through two programs. One program records the voltage and the current of the nano-satellite components, while the other records the data related to the environment and the behavior of the nano-satellite. The two programs store the data retrieved on separate files, accessed by the program interacting with the simulation to retrieve the data it needs to send.

3. Results

We now showcase the results obtained from the experimental evaluation we conducted on two different metrics: realism and scalability. The metrics are evaluated using 10 simulated nano-satellites, 1 physical nano-satellite, with 10 jobs for realism and 20 jobs for scalability.

3.1. Realism

For this metric, we want to evaluate if our system is able to retrieve and integrate real data from a physical nano-satellite. This metric is crucial to understand if our system provides a comprehensive HIL simulation. To evaluate this metric, we compare the energy consumption retrieved from the physical nano-satellite and integrated into the various simulated nano-satellites, to the energy consumption retrieved from the datasheets of the components of the CubeSatSim. We opt to use the power consumption of datasheets found online because these data represent a real-world scenario.

To convert the power consumption of the datasheets to energy consumption, which is the measure we use in Figure 4, we multiply for every simulated nano-satellite the power consumption of the datasheets by the interval in which the physical nano-satellite processes all the jobs of the simulated nano-satellite.

As we can see in Figure 4, for every simulated nano-satellite, the energy consumption has nearly the same magnitude as the energy consumption retrieved from the datasheets. The variations between the datasheets and the simulated nano-satellites could be related to multiple factors: *i)* environmental factors such as temperature or humidity, *ii)* different CPUs usage, *iii)* different programs used to stress the component, and *iv)* variation in the voltage in a real-case scenario, compared to the use of a standard voltage value in the datasheets.

Since the energy consumption we compared has nearly the same magnitude, we can conclude that our simulation is able to integrate the real data provided by the physical nano-satellite in the various simulated nano-satellites.

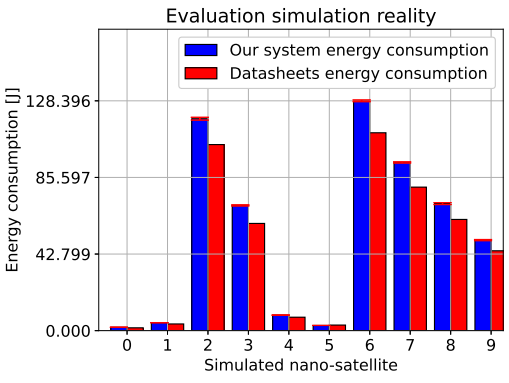

Figure 4: Realism evaluation.

3.2. Scalability

We aim to test the enhanced scalability we provide to our system. The simulation runs on a server with 48 CPUs, 256 GB of memory, and 1 TB of disk. In Figure 5, we evaluate if our system is capable of scale, to understand if it

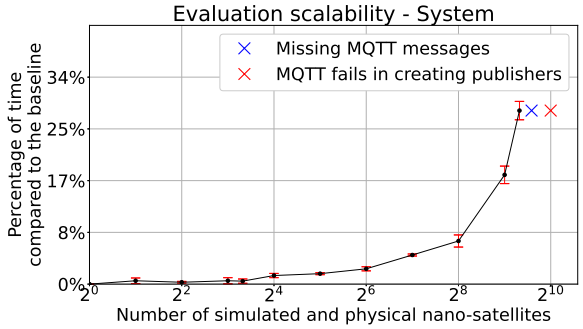

Figure 5: System scalability.

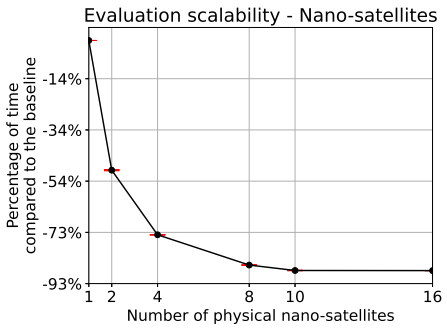

Figure 6: Physical nano-satellites scalability.

can parallelize the execution, maintaining the performance when the number of simulated and physical nano-satellites increases. Initially, the curve is flat because there is always a physical nano-satellite ready to process the job requested by a simulated nano-satellite.

At a certain point, we can observe an exponential increase. This situation is attributed to the saturation of the processing cores on the machine that runs the simulation, which is forced to limit its concurrent processes, reducing the parallelization.

Once we reach the X-marks, we encounter errors related to the communication channel, i.e. MQTT, which results in an erroneous simulation. This test enables us to understand that our system is capable of managing ∼700 physical nano-satellites simultaneously.

In Figure 6, we evaluate how the number of physical nano-satellites involved in the simulation impacts the simulation performance.

Increasing the number of physical nano-satellites reduces the execution time of the simulation due to the parallel execution of the simulated nano-satellites and the use of multiple physical nano-satellites, following our scalability approach of Figure 3. So, if a physical nano-satellite is working for a simulated counterpart, the other simulated nano-satellites can request the data from the free physical nano-satellites.

However, when the number of physical nano-satellites equals the simulated nano-satellites the curve flattens out because every simulated nano-satellite always has a physical nano-satellite to interact with, therefore, when we have more physical than simulated nano-satellites, the extra physical nano-satellites never work.

In conclusion, from the results we obtained, we can say the scalability provided to our system is able to decrease the overall execution time of the simulation by a factor of 88% when using multiple nano-satellites, compared to using a single physical nano-satellite.

4. Conclusions

In this thesis, we provide a HIL simulation, which is able to use real data, retrieved from a real nano-satellite. The system can also simulate radiation-induced errors, in real time, on the real nano-satellite. Lastly, the scalability provided by our system, enables the simulation to be interconnected with multiple nano-satellites simultaneously. Our results prove that our simulation can use data retrieved from a physical nano-satellite. Furthermore, we empirically verify that our scalability approach shortens the overall execution time of the simulation by a factor of 88% when using multiple nano-satellites, compared to using a single physical nano-satellite, also enabling the HIL simulation to manage ∼700 physical nano-satellites simultaneously.

References

[1] Ansys. Stk space simulator, ansys.com, 2022.

[2] Bradley Denby and Brandon Lucia. Orbital edge computing: Nanosatellite constellations as a new class of computer system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[3] A Johnstone. Cubesat design specification (rev. 14.1). *Cal Poly SLO*, 2022.

[4] Nsnam. Ns-3 simulator, nsnam.org, 2023.

[5] Roger Walker. Esa, cubesats specifications, esa.int.