UNIVERSITAT POLITÈCNICA DE CATALUNYA

A scalable distributed autonomy system for fractionated satellite missions

A Degree Thesis Submitted to the Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona by

Santiago RODRIGO MUÑOZ

in partial fulfilment of the requirements for the

Degree in Science and Telecommunication Technologies Engineering

Advisors:
Elisenda BOU BALUST
Carles ARAGUZ LÓPEZ

Reporting advisor (ponent):
Dr. Eduard ALARCÓN COT

January 25, 2016

Dedicated to my family and friends, who are always there when I need them.

Acknowledgements

First of all, I would like to thank Prof. Eduard Alarcón, Elisenda Bou and Carles Araguz, the advisors of this project, for the opportunity of carrying out this work and for guiding and helping me during all this time.

To all my family and friends goes my most sincere gratitude for their invaluable affection and support.

I am also very grateful to all the people in the NanoSat Lab who helped me, specially to Marc Marí and Arnau Prat, who introduced me to the software development for space applications, in the context of the ABS project.

Finally, I would like to express my gratitude to those professors of the UPC who knew how to transmit the enthusiasm over what they do at university.

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Abstract

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Degree in Science and Telecommunication Technologies Engineering

A scalable distributed autonomy system for fractionated satellite missions

by Santiago RODRIGO MUÑOZ

Current trends in space systems are towards the design of autonomous spacecraft that are capable to generate their own plans of action based on system constraints, on-board data analysis and mission goals set by ground operators. At the same time, the requests in Earth Observation applications has forced the exploration of innovative satellite systems such as satellite swarms, constellations, fully-fractionated spacecraft and federated satellite systems. Enabled by miniaturization techniques and small-spacecraft technologies (e.g. nano-satellites), these distributed mission architectures present improved system qualities (i.e. modularity, scalability, resiliency, incremental deployment...) and performance (e.g. spatial resolution, shorter revisit times...)

Although Mission Planning Systems (MPS) for satellites are traditionally ground-based and centralized, providing autonomous capabilities to distributed spacecraft could be the only feasible solution in these new mission architectures where requests are served in a cooperative manner. Regardless of the benefits of autonomous distributed spacecraft, these approaches pose significant challenges in the core component of their planning systems, namely, distributed task schedulers. In this context, this thesis will contribute to the exploration of task schedulers for distributed spacecraft.

The purpose of this thesis is twofold: on the one hand to fully design and implement a distributed task scheduling policy. This approach is targeted for the broad range of distributed satellite missions, presenting an adaptive management policy based on the collaboration between two levels of control for dynamic environments with limited computational capabilities.

On the other hand the performance of this policy is tested in a simulated environment, against another state-of-the-art alternative, which has been also completely implemented. A benchmark framework has been specifically designed for this purpose. The obtained results have allowed to detect the weaknesses and strengths of this policy and the next steps to be taken in order to have a distributed task scheduler prepared for the future distributed spacecraft architectures.

UNIVERSIDAD POLITÉCNICA DE CATALUÑA

Resumen

Escuela Técnica Superior de Ingeniería de Telecomunicación de Barcelona

Grado en Ciencias y Tecnologías de las Telecomunicaciones

Un sistema distribuido escalable de autonomía para misiones de satélites fraccionados

por Santiago RODRIGO MUÑOZ

Las tendencias actuales en los sistemas espaciales se dirigen hacia el diseño de naves espaciales autónomas que sean capaces de generar sus propios planes de acción teniendo en cuenta las limitaciones del sistema, los objetivos de la misión establecidos por los operadores de tierra y los datos obtenidos por la instrumentación de a bordo. Al mismo tiempo, los requerimientos de las aplicaciones de observación de la Tierra han llevado a la exploración de innovadores sistemas de satélites, como los enjambres (*swarms*)de satélites, las constelaciones, los satélites fraccionados y los sistemas de satélites federados. Gracias a las técnicas existentes de miniaturización y las tecnologías orientadas a los satélites de tamaño reducido (como los nanosatélites), estas arquitecturas de misión distribuidas presentan mejoras en cuanto a sistema (como la modularidad, escalabilidad, flexibilidad, implementación incremental,...) y en cuanto al rendimiento (una mejor resolución espacial, periodos de revisión más cortos,...).

Aunque tradicionalmente los sistemas de planificación de misión (MPS) para satélites han sido centralizados y controlados desde tierra, proporcionar una mayor autonomía a las naves espaciales distribuidas podría ser la única solución factible en estas nuevas arquitecturas de misión donde se atienden las peticiones de una manera cooperativa. Independientemente de los beneficios de los sistemas espaciales autónomos, estos enfoques plantean desafíos significativos en el componente principal de sus sistemas de planificación: a saber, los planificadores de tareas distribuidos. En este contexto, esta tesis contribuye a la exploración de los planificadores de tareas para naves espaciales distribuidas.

El propósito de esta tesis es doble: por un lado, diseñar completamente e implementar una política de planificación de tareas distribuida. Esta propuesta está dirigida a toda la amplia gama de sistemas espaciales distribuidos, puesto que presenta una política de gestión adaptativa basada en la colaboración entre dos niveles de control para sistemas con continuos cambios y con capacidades computacionales limitadas.

Por otra parte, la realización de esta política se ha puesto a prueba en un entorno simulado, comparándola con otra alternativa del *estado del arte* en este campo, y que ha sido también completamente implementada. Para este propósito, se ha diseñado un entorno de simulación específico. Los resultados obtenidos han permitido detectar las debilidades y fortalezas de esta política y los próximos pasos a seguir con el fin de tener un planificador de tareas distribuido preparado para las futuras arquitecturas de satélites distribuidas.

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Resum

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Grau en Ciències i Tecnologies de Telecomunicació

Un sistema distribuït escalable de autonomia per missions de satèl.lits fraccionats

per Santiago RODRIGO MUÑOZ

Les tendències actuals en els sistemes espacials es dirigeixen cap al disseny de satèl.lits autònoms que siguin capaços de generar els seus propis plans d'acció tenint en compte les limitacions del sistema, els objectius de la missió establerts pels operadors de terra i les dades obtingudes per la instrumentació de bord. Alhora, els requeriments de les aplicacions d'observació de la Terra han portat a l'exploració d'innovadors sistemes de satèl.lits, com els eixams (*swarms*) de satèl.lits, les constel.lacions, els satèl.lits fraccionats i els sistemes de satèl.lits federats. Gràcies a les tècniques existents de miniaturització i les tecnologies orientades als satèl.lits de mida reduïda (com els nanosatèl.lits), aquestes arquitectures de missió distribuïdes presenten millores pel que fa al sistema (com ara la modularitat, escalabilitat, flexibilitat, implementació incremental,...) i pel que fa al rendiment (una millor resolució espacial, períodes de revisió més curts,...).

Encara que tradicionalment els sistemes de planificació de missió (MPS) per a satèl.lits han estat centralitzats i controlats des de terra, proporcionar una major autonomia a les naus espacials distribuïdes podria ser l'única solució factible en aquestes noves arquitectures de missió on s'atenen les peticions d'una manera cooperativa. Independentment dels beneficis dels sistemes espacials autònoms, aquests enfocaments plantegen desafiaments significatius en el component principal dels seus sistemes de planificació: a saber, els planificadors de tasques distribuïts. En aquest context, aquesta tesi contribueix a l'exploració dels planificadors de tasques per a naus espacials distribuïdes.

El propòsit d'aquesta tesi és doble: d'una banda, dissenyar completament i implementar una política de planificació de tasques distribuïda. Aquesta proposta està dirigida a tota l'àmplia gamma de sistemes espacials distribuïts, ja que presenta una política de gestió adaptativa basada en la col.laboració entre dos nivells de control per a sistemes amb continus canvis i amb capacitats computacionals limitades.

D'altra banda, la realització d'aquesta política s'ha posat a prova en un entorn simulat, comparant-la amb una altra alternativa de l'*estat de l'art* d'aquest camp, i que ha estat també completament implementada. Per a aquest propòsit, s'ha dissenyat un entorn de simulació específic. Els resultats obtinguts han permès detectar les debilitats i fortaleses d'aquesta política i els propers passos a seguir amb la finalitat de tenir un planificador de tasques distribuït preparat per a les futures arquitectures de satèl·lits distribuïdes.

Contents

List of Figures

List of Abbreviations

GPS	**G**lobal **P**ositioning **S**ystem
MPS	**M**ission **P**lanning **S**ystem
ABS	**A**ndroid **B**eyond the **S**tratosphere
UPC	***U**niversitat **P**olitècnica de **C**atalunya*
EDD	**E**arliest **D**ue **D**ate scheduling
EDF	**E**arliest **D**eadline **F**irst scheduling
RM	**R**ate **M**onotonic scheduling
IoT	**I**nternet **o**f **T**hings
WSN	**W**ireless **S**ensor **N**etworks
UAV	**U**nmanned **A**erial **V**ehicle
ISL	**I**nter-**S**atellite **L**ink
EST	**E**arliest **S**tart **T**ime
LST	**L**atest **S**tart **T**ime
LEO	**L**ow **E**arth **O**rbit

Chapter 1

Introduction

1.1 Autonomy and decentralization in satellite systems

In the context of satellite missions, new mission architectures and original approaches are currently being explored. Traditional monolithic satellites are being replaced by decentralized satellite architectures, such as fractionated spacecraft or satellite swarms.

Miniaturization of technology, reduced launch costs, and standardization are increasing the interest in small satellites for these **distributed missions**. These small satellites are satellites of reduced size and low mass (under 500 kg). They can be classified according to its mass in: *small-satellites* from 500 kg to 100 kg, *micro-satellites* from 100 kg to 10 kg, *nano-satellites* from 10 kg to 1 kg, *pico-satellites* from 1 kg to 100 g and *femto-satellites* for less than 100 g. With clusters of these smaller modules, distributed missions are aimed at serving Earth Observation requests with similar or better characteristics (e.g. spatial resolution, revisit time) than those of large monolithic satellite systems, while reducing their cost and development times.

They are also seen as a feasible alternative to achieve better spatial resolutions and shorter revisit times. In fact, the requirements of future Earth Observation applications (e.g. real-time video, stereoscopic radar, multi-spectral imagery) might only be cost- and time-effective through the implementation of these new mission approaches, where several small units work cooperatively to acquire data.

Moreover, the fractionation of satellites and their decentralization allows the system to continue operating even when some of its parts is malfunctioning, something that increases the robustness and reliability of the system.

Despite the fact that distributed mission are still being explored, this context requires the study of management systems capable to govern these complex systems. Traditional management systems are mainly based on centralized complex planners that run on the ground station, but this approach does not benefit from all the advantages of the distributed satellite system. For this reason, the management systems for distributed satellite missions under research tend to be decentralized.

Simultaneously, the research on satellite missions is also becoming more and more focused on a new paradigm in the satellite's conception: to provide increasing **autonomy** to the spacecraft. Some benefits of having autonomous spacecraft are:

- To provide a better resilience to the system (e.g., it does not depend on other entity and the state of the communication channel in between), which will increase the robustness of the whole mission.
- To improve the mission's performance through on-board data analysis. This allows to adjust on-board instruments and focus on higher interest locations or events without transmitting the data to ground and waiting for instructions.
- To enable huge distributed systems with many interconnected nodes to decentralize the management, as in these cases a centralized operation would be unfeasible.

The autonomy and decentralization are current trends on the design of satellite missions, motivated by the requirements of new Earth Observation applications (e.g., better image resolution, shorter revisit times, etc.). Several novel mission architectures based on these concepts have appeared [1, 2]:

- **Fully-fractionated spacecraft**: a satellite decomposed into several physically-detached modules (i.e., energy system, communications, storage module, data processing module, etc.) orbiting closer and exchanging system resources (e.g., power and data).
- **Satellite swarms**: distributed missions in which the infrastructure nodes are independent satellites that perform their own tasks without resource exchange or collaboration. This type of distributed satellite tends to be composed of homogeneous modules with similar or identical capabilities.
- **Trains or constellations**: group of sparsely distributed satellites with coordinated maximum ground coverage, for applications such as Earth-Observation, communication or geo-location (GPS).
- **Federated Satellite Systems**: purely collaborative missions in which the spacecraft opportunistically share their commodities with others in order to satisfy the needs of a global request (at the constellation level) [3].
- **Hybrid missions**: heterogeneous missions which combines characteristics of more than one of the presented archetypes, as they represent theoretical extremes.

These complex architectures have led to the exploration of adapted Mission Planning Systems (MPS), requiring them to be **distributed** and **executed on-board**. These MPSs aim at providing the needed autonomy to the constellation and solving the spacecraft management problem.

In this context, different Mission Planning Systems and policies have been proposed. One of these policies is the one that has been presented for the Android Beyond the Stratosphere (ABS) project,[1] the Local-Global policy [1].

A peculiarity of this approach is that it focuses on the nano-satellite based fractionated spacecrafts, taking into account the importance they have as the next-generation satellite missions architectures. A big challenge present in these class of space systems is that the computational capacities are reduced due to the existing power restrictions, something considered in this policy's design.

The required computational efficiency can be achieved by a partition of the scheduling problem in several simpler sub-problems such that each satellite is able to solve one of them (this could be compared to what is done for meteorology predictions, in which a number of local models are combined within a global macro-model).

The main aim of this thesis is to explore and assess a satellite autonomy system which applies the Local-Global policy, by designing, implementing and comparing it with a state-of-the-art distributed scheduling algorithm to determine whether it performs good at solving the problems it has been designed for.

First, the Local-Global policy and other state-of-the-art planning algorithms will be presented, and the details of the implementation of the two compared algorithms will be explored. Then, a benchmark for the comparison will be designed, and metrics for the experimental measures will be defined.

The most important desired result is to conclude whether the ABS policy behaves well in a simulated environment compared to other algorithms. By *behaving well* it is meant that, for the defined metrics, it achieves to schedule the tasks among the workers within a minimum time and using an optimal allocation of resources.

[1] *Universitat Politècnica de Catalunya*'s (UPC) ABS project, in which context this thesis has been developed, aims at developing a nano-satellite platform standing on modular, low-cost open-source components and standards with an Android-based satellite-on-a-phone architecture.

In order to accomplish this goal the algorithms have been tested with a benchmark of multiple simulated scheduling problems. The experimental results analysis are shown at the end of the present document, before concluding and stating the future work to be done.

1.2 The Android Beyond the Stratosphere Project

This thesis has been carried out in the context of the Android Beyond the Stratosphere project. This project, performed at the Nano-Satellite and Payload Laboratory of the Technical University of Catalonia – UPC BarcelonaTech, aims at designing a standardized modular open-source nano-satellite platform based on commercial components and open standards. The satellite-on-a-phone architecture is being explored to implement a low-cost nano-satellite based on a well-known system such as Android.

Moreover, it is intended to develop a fully distributed system in which a number of these low-capacity Android-based satellites interact and collaborate to achieve global targets. The needed synchronization for this collaborative satellite constellation, as it has been stated in the previous section, requires an advanced scheduler algorithm that allows the system to distribute the tasks among all the satellites forming the system in a balanced and optimal way (in terms of time and resource consumption). This task scheduler is the core of this thesis, which is intended to design, implement and compare it with other state-of-the-art distributed task allocator, and experimentally test its efficiency and optimality.

A centralized scheduler running in a *leader* satellite could be a possible solution, but the complexity of the problem and the low computational capabilities of these low-cost nano-satellites make this approach unfeasible. Although much more challenging, this type of distributed approaches provide autonomous decision capabilities to the satellites in a constellation and greatly improve the qualities of the system. Adaptiveness and low resource consumption: these are the key concepts in a system like this. With a distributed algorithm running on every satellite the computational load is shared among all and the autonomy of each satellite is preserved.

1.3 Scheduling tasks: a hard optimization problem

The core of this thesis is a typical optimization problem: to schedule some input tasks with varying or fixed processing time taking into account the resources available in the system and the time requirements that the tasks may have, such as different arrival times, deadlines or dependence between tasks.

Scheduling becomes a hard problem that has to be solved with advanced programming techniques such as constraint-based programming or heuristics. In fact, most problems, such as multiprocessor scheduling or Job Shop Scheduling, are proven to be NP-hard optimization problems.

This problem has been studied deeply (in fact it is a very mature field of research), and there are many well-known scheduler algorithms for one or many workers (e.g., EDD, EDF, RM... [4]). However, they are mainly mono-processor algorithms, i.e. the algorithm runs in a single machine and if the solution applies to other machines, it is sent to them. The recent growth of distributed systems with increasing autonomy and processing capacity, very present in the IoT, WSNs (Wireless Sensor Networks) and robotics, has led to accelerate the research on distributed schedulers.

This last case applies to this work: The nodes have to cooperatively solve an NP-hard problem, i.e. the satellites executing the algorithm must achieve a solution which is good for all the system by only knowing its own state and communicating with the others to agree the final solution. Moreover, the nodes are working with heavily constrained inter-satellite communication channels. It can be easily concluded that *distributing* a task scheduler adds much complexity to an already computationally hard problem. But this is not necessarily true: one of

the most widely used computer science paradigms is "divide and conquer". The advantages of sharing the computational burden clearly outweigh the communication costs.

1.4 Distributed systems

Distributed systems are formally defined in [5] as *"software systems in which components located on networked computers communicate and coordinate their actions by passing messages"*. Therefore, the basis of these type of software systems is the communication. By *talking* with the other workers in the network a component of the system can take into account the shared knowledge about the state of the rest of nodes and work to obtain the whole system's goal.

There are multiple architecture designs available for distributed systems, having both **centralized** and **fully-independent** structures [6, p. 36]. Sometimes it is needed to have a single node doing some special functions to control the state of the entire system[2]. However, it means always a single point-of-failure, a weakness that affects the robustness of the entire system, apart from decreasing the system throughput whenever it is needed to contact the leader, which can be a bottleneck. This does not mean that in some situations the presence of a *master* node is a strength for the system, because it itself is a high-capacity robust node (e.g., consider the ground station of a satellite constellation).

1.4.1 Distributed mission planners

Distributed algorithms are pieces of code designed to run on several nodes, having as a key function the intercommunication between nodes.

The autonomy of the nodes running the distributed planners modifies some typical programming patterns. Also synchronism problems or node failures appear. These new challenges are solved with different approaches: problem's solution achieved by *consensus* among nodes, clock synchronization algorithms, data consistency and reliable communication, redundancy and failure recovery... [6]

The present thesis will not focus on the synchronism and failures problems, but on the ability of distributing computational efforts to solve a complex scheduling problem. This will be key to improve the computational efficiency of the system: how it takes profit of the intercommunication to precisely distribute the work without missing crucial information for obtaining the optimal scheduler.

1.5 Previous work

ABS project has been developed for two years and a half, and a prototype of an individual Android-based satellite is being finished, with some work in hardware integration still to be done. The distributed software architecture is also a key part of this project: it is intended to be as generic as possible, in order to enable the deployment of any distributed satellite system topology.

The task scheduler is a key part of this distributed architecture, as it enables the whole system to achieve the global goal while distributing smaller tasks among all the nodes composing the constellation.

1.5.1 The Local-Global policy

In [1] the main characteristics of the distributed task scheduler proposed for the ABS project were described, and its procedure completely explained. However, an implementation was to

[2] In fact, leader election and recovery is one of the most critical functionalities in distributed programming.

be done and there were still no experimental results to be shown. In this thesis this policy has been fully designed and implemented, and experimental measures have been taken through a benchmark specifically designed. Below a brief description of the presented algorithm is shown.

This scheduling policy aims at providing an adaptive technique for a distributed spacecraft to find an optimal schedule for an arbitrary number of satellites composing it. To be capable of modelling the possible heterogeneity present among the different nodes, this algorithm takes into account the processing capabilities and available resources of every node.

The policy basically divides the multi-ple-tasks multiple-workers problem into several multiple-tasks single-worker sub-problems. Every *Local* entity (i.e. every system node) will evaluate each sub-problem, as if it was a fully-local scheduling problem, and the solutions found locally are sent to the *Global* layer, represented by a master node, which will be in charge of combining them and finding out the optimal schedule. Once the final solution is generated, the *Global* node sends it to the rest of the constellation.

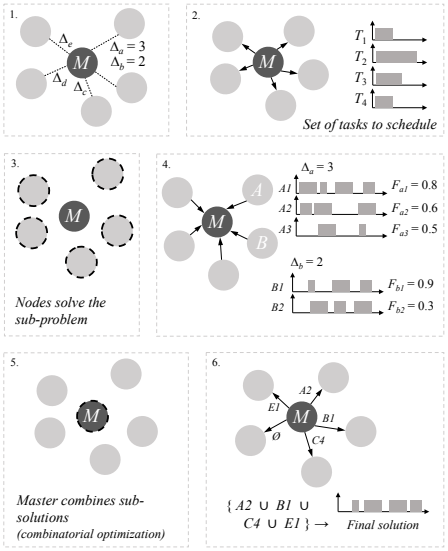

FIGURE 1.1: Local-Global policy steps (as extracted from [1])

To make possible the re-composition of the initial problem and to enable the master to find out the optimal combination taking into account as much information of the *quality* of each local solution as possible, two parameters are defined: the figure of merit F and the golden index Δ. The first of these parameters measures the goodness of each local solution, as a set of parameters describing it. The second one is the number of possible solutions that each *Local* entity can report to the global layer, and is meant to balance the potential heterogeneity in the system: nodes possessing higher processing capabilities will obtain a higher Δ than those with lower computational resources.

Having described the basics of the Local-Global algorithm the procedure it follows will be now divided in six steps (see also Fig. 3.1):

1. **Characterization.** The Δ value for each satellite is assigned, after considering its computing capabilities.
2. **Task delivery.** Master node selects the time window (duration of the schedule to be produced by the scheduling algorithm) and discards the tasks that cannot be scheduled *a priori*. After this, it sends to every *Local* entity the set of tasks, so that they can begin to compute sub-solutions.
3. **Local evaluation.** Local entities' task planners produce as many solutions as its Δ_i value, attaching to each sub-solution an F value.

4. **Submission of solutions.** Each satellite provides the set of at most Δ_i solutions, sending for each one the list of tasks included in that schedule solution and its figure of merit F.
5. **Global selection and combination.** The master triggers a combinatorial optimization process that selects at most one sub-solution per satellite in order to maximize the aggregated F value, which is not necessarily the sum of the single F values.
6. **Distribution of solution.** The master notifies each *Local* entity with the identity of its selected sub-solution, if any.

In section 3.1 the focus will be on the implementation that has been developed in this thesis, specifying the *Local* entity task planner characteristics and the optimization algorithm used in the global layer.

Chapter 2

State of the art

2.1 A taxonomy of the existing task scheduler paradigms

The existing distributed system architectures –and also the existing distributed satellite systems– can be classified according to the degree of centralization, from completely hierarchical systems to fully independent nodes. The research in the distributed algorithms field can also be divided into centralized and non-centralized paradigms, although it is not limited to that. However, completely new problems appear in the distributed systems, such as synchronization, leader election or data consistency, and every different problem opens new approaches and new paradigms.

In particular, into the field of task scheduling, many approaches have been taken. There are plenty of scheduling policies and algorithms that could be compared with the Local-Global, from high-complexity fully-centralized algorithms that may be solved using dynamic or constraint programming (mainly for grid-computing or High Performance Computing, as in [7–9]), to low-computational decentralized schedulers [10, 11].

In the last years, distributed task scheduler research has mainly worked over three different paradigms, although not all of them have been applied to the distributed satellite systems context:

- **Negotiation or consensus** policies, in which nodes in the system *agree* the final schedule, such as the distributable scheduling proposed in [12] for a distributed real-time system with hard end-to-end time constrains in the activities executed in parallel threads or the consensus protocol for balancing the burden of distributed applications execution in IoT contexts presented in [13]

  Other examples of this type of distributed consensus-based schedulers are the algorithm presented in [14] –which is based on an iterative and asynchronous local optimization of the tasks allocations between neighbour nodes– and the price-based[1] approach for balancing energy consumption among nodes in WSNs of [15].

  Two very interesting approaches can also be found in [16] and in [17]. The contexts are very similar to the ABS satellite constellation. In the first policy, a combination of consensus and auction-based decision-making is used to coordinate a fleet of autonomous vehicles (UAVs) with two decentralized algorithms. The second paper describes a communication-constrained task allocator in the same context of that of ABS project: a satellite constellation.

- **Centralized** optimized schedulers –where the degree of centralization can vary–, such as the solution presented in [18], in which a *Coordinator* that possesses all the intelligence of the network is in charge of planning the job distribution in a WSN or the multiserver-oriented algorithm of [19] where a central job scheduler measures the server idle-time measurements to balance the servers load.

[1] In a price-based consensus algorithm each node in the system calculates a bid depending on their knowledge and state, and the lowest value is chosen as the agreed value.

- The more recent **bio-inspired** approaches –which take profit from observing and imitating the behaviour of the distributed systems present in the nature–, such as the adaptation of the stigmergy used by ants to stablish an optimal path of [20] or the resources balancing technique extracted from the one observed in ant colonies [21]. These techniques are the least explored and applied for distributed satellite systems.

On the other hand, the Local-Global policy could be described as a hybrid distributed algorithm[2]. Among all the algorithms presented a candidate for the comparison with the Local-Global must be chosen. To do so, some of the most interesting approaches that have been found, and what are the reasons for having chosen a particular one, will be briefly described in the next section.

2.2 Comparison: algorithms chosen to benchmark the Local-Global

Having explored the paradigms that are being investigated in the distributed task scheduler field, a state-of-the-art algorithm must be found out for the comparison with the Local-Global policy: it should be designed for (or applicable in) similar contexts and resource requirements.

As it has been already stated, the research in the field of distributed satellite systems task schedulers is still in its first steps. Moreover, the computational capabilities of nano-satellites have been traditionally very low to consider a fully-autonomous system-wide distributed task scheduler running on every node. Furthermore, distributed satellite missions are still unfeasible due to the lack of accurate Inter-Satellite Links (ISL)[3] for huge constellations of satellites. For this reason, the research on the distributed satellite system schedulers cannot be applied to real environments yet.

Relevant papers presenting distributed task schedulers suitable for similar contexts to ABS are only a few, as many times some characteristics of the presented system are too different to compare this policy with. There are policies for nearly infinite bandwidth communication among nodes, multi-processor scheduling algorithms, distributed task allocators for nodes with high computational capabilities...

The context in which the Local-Global policy is meant to work is highly-constrained environment in terms of resources and communication, a moderate processing capacity (as it is aimed to work on a satellite-on-a-phone nano-satellite, which has higher computing capabilities than standard nano-satellites) and possibly fully independent nodes (depending on the final distributed software architecture).

This reduces the search field: algorithms that are designed to run on practically infinite communication bandwidth grid computing platforms (as the one presented in [19]) cannot be compared with the Local-Global.

Policies that are, in fact, designed for distributed Operating Systems or deploying heavy distributed applications, where the timing requirements are highly-constrained but the communication is practically unlimited [12, 14] are useless for the comparison. Other policies with innovative concepts can be found, such as the auction decision model presented in [22]. However, it is invalid as long as it does consider resource requirements in a simple way and the problem resolution is fully centralized, with no participation from the system nodes.

There are some highly interesting bio-inspired approaches. In [20], stigmergy[4] is introduced as a novel method for coordinating a satellite cluster. Stigmergy also provides a coordination system for environments with highly constrained communications, as in distributed satellite

[2]Local-Global policy combines the hierarchy of centralized approaches in the *Global* entity while sharing the computational load among all the nodes for a fully-distributed resolution of the scheduling problem.

[3]ISLs are stand-alone communication links among two or more satellites, not depending on the ground station.

[4]Stigmergy is a mechanism of indirect coordination between agents or actions. The trace left in the environment by an agent while performing a previous action stimulates other (or the same) agents the performance of the next action. It is usually found in nature, as in ant collonies coordination [23].

missions. In [21] a mathematical model for analysing the job division in ant colonies is presented. This allows to explore simple task allocation mechanisms to be used to achieve an optimal division of work. In this case, the strategy followed is based on a binary feedback function that senses the current labour allocation (i.e., wether the activity being performed needs more workers or not).

Nonetheless, both approaches, as well as other papers related to bio-inspired scheduling techniques for distributed systems, are mainly theoretical and have been tested in unrealistic simulated scenarios.

[16] and [17] could be also suitable for the comparison, as both of them are designed for a fully-distributed system. In the first one, auction-based coordination guarantees conflict-free task assignment, but the problem description is more complex than the one that is needed for the ABS case. In the second paper a collaboration method based on an incremental coalition is presented. The satellites forming the constellation are supposed to be able to communicate with other neighbour nodes occasionaly: whenever they meet at the points where their orbits cross. In this small amount of time, the satellites communicate its intentions about the tasks to be scheduled to each other. In this way, knowledge is transmitted in an *epidemic* manner through the constellation and the satellites finally agree on a scheduling decision. This description could apply to the ABS case. However, the local scheduler of each satellite –which is in charge of processing the intentions received and providing new ones– is not described in the paper, hence it is not possible to fairly implement it.

Finally, a distributed-architecture-agnostic price-based approach was found in [15]. The description of the tasks is adequate to what is needed and its simplicity yet completeness shows a fully-decentralized distributed scheduler good for the comparison.

Additionally, a ring-architecture task scheduler algorithm has been designed from the leader election algorithm described in [6, p. 266]. The reason of designing this algorithm was to use a typical distributed structure such as a ring for combining it with the Local-Global policy (as an optimization) in future research.

2.2.1 A price-based approach

The work of [15] presents an adaptive task allocation scheme, suitable for Wireless Sensor Networks, a similar context to that of the ABS distributed system: a resource-constrained environment. The designed algorithm aims at distributing a set of tasks among several nodes in an energy-balanced manner, to maximize the overall network lifetime. Each node in the WSN is fully autonomous and resource-constrained, and every node has the capabilities to perform any task, as long as it has the required processing time and energy to carry out the task within its deadline.

A price-based architecture is proposed, in which each node is modelled as a *seller* who calculates the cost of executing a specific task and communicates it to the *consumer* (the task sender), adapting the price to the changing resources availability for a better energy balance amongst the nodes composing the system.

In this section a brief description of the presented algorithm, as described in [15], is presented, and in section 3.2 the details of the implementation carried out in this Thesis will be detailed.

The basic operation of this price-based task allocator is the one that follows: whenever a task arrives to the system, all the nodes in the system are broadcast the task information. Each node calculates the price it will offer based on its resources and time constraints. High prices mean less energy remaining in the node and/or later processing of the task. Two methods for determining the winner in each round are introduced, having a centralized scheme and a fully distributed one. Results presented in [15] state that the distributed scheme requires less overhead and is more efficient, simply by delaying the price transmission a period of time proportional to the calculated price. This mechanism is further explained later.

The algorithm can be divided in three phases:

1. **Listing Phase.** A decomposition of an initial set of tasks into a set of smaller sub-tasks generates a task sequence that respects the time constraints and concurrency requirements of the initial group of tasks. Each subtask is represented by an Earliest Start Time (EST, the first time instant in which it can be executed without interfering with its predecessor tasks) and a Latest Start Time (LST, the last time instant in which the task can begin to allow the successor tasks to be executed). The tasks are queued on a list ordered by their ESTs and LSTs. This queue keeps the order in which the tasks will be broadcast to the nodes, ready for the task-assignment phase.

2. **Price-Based Task Assignment Phase.** This phase is executed *online*: the tasks are scheduled as they arrive to the system. The core of this phase is the price formulation which in addition allows to increase the privacy of the nodes, as they do not transmit the real value of their remaining resources, but only a derived cost.

   The parameters used to calculate task prices are: task size, energy price, base task price, communication cost, task deadline and processor release time.

   Task Size (S): the energy needed to process that particular task.

   Energy Price (EP): this value's objective is twofold: to show higher prices for devices with lower energy available (in fact, it is in some way inversely proportional to the instantaneous energy level) and to reflect the cost of recharging the energy. Its value for the node i is defined as:

   $$EP_i = \frac{a}{1 - e^{E_i/b}} \tag{2.1}$$

   Base Price (BP): the computational cost of processing a task in a particular node. It is defined as (node i, task j):

   $$BP_{ij} = S_j \times EP_i \tag{2.2}$$

   Communication Cost ($CommCost$): the cost of transmitting the output of a task to the node that will process its successor task.

   Task Deadline (TD): the LST already defined in the Listing Phase.

   Processor Release Time (RT): the time at which the task execution would finish if the node finally schedules it.

   With all these variables, the proposed price calculation is the one in (2.3), with DL being the arriving time of the task. This price calculation's goal is to balance the energy consumption among the nodes and to achieve a quicker processing time to more urgent tasks.

   $$P_{ij} = (CommCost + BP_{ij}) \left[1 + exp^{\left[\frac{\lambda(t, DL_j)}{\gamma(t, RT_i)}\right]}\right] \tag{2.3}$$

   $$\lambda(t, DL_j) = \begin{cases} k(t - DL_j) & \text{for } t \geq DL_j \\ \epsilon & \text{for } t \leq DL_j \end{cases} \tag{2.4}$$

   $$\gamma(t, RT_i) = \begin{cases} k(t - RT_i) & \text{for } t \geq RT_i \\ \epsilon & \text{for } t \leq RT_i \end{cases} \tag{2.5}$$

3. **Recovery Phase.** This phase is intended for recovering from node failures during the task assignment phase, taking into account the existing dependence between tasks.

4. **Price offering.** Two methods are presented to select the winner bid. This thesis' implementation focuses on the more efficient decentralized method. Instead of transmitting to the *consumer* the price calculated as soon as it has been obtained, each node waits for a proportional waiting time before broadcasting it to all nodes and goes to a LISTEN

mode. If a lower price is received, the node leaves the competition and does not broadcast its bid. Therefore, only the node with lower price effectively sends its bid, reducing the amount of transmitted information.

2.2.2 A ring-based approach

Despite the simplicity of the leader election algorithm found in [6] it represents a typical distributed architecture: the ring structure. The design that has been performed in this work to adapt this algorithm for having a distributed task scheduler will be now described. Instead of agreeing on who will be the next leader after a leader failure, the agreement is focused on a schedule for the whole system.

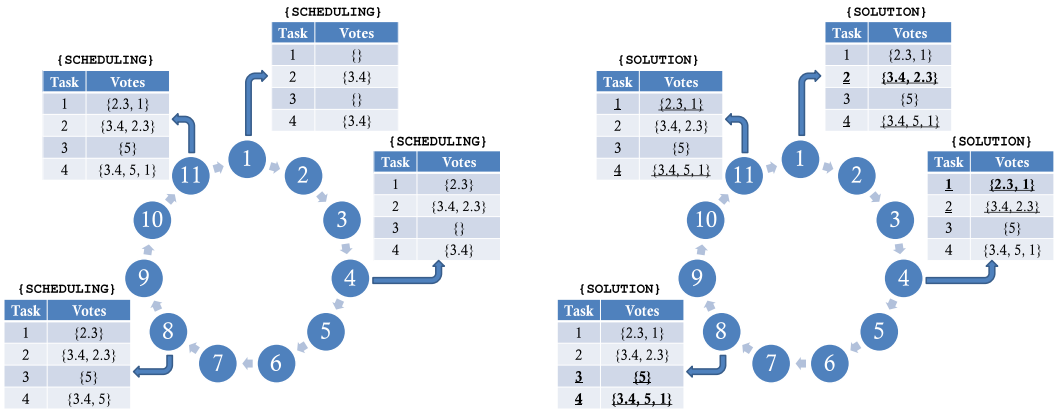

FIGURE 2.1: Ring-based scheduler approach

It is assumed that the satellites in the system are physically[5] or logically ordered, so that each satellite knows who its successor is and is able to communicate with it. When a task scheduling process is triggered, the global layer sends to a *master* satellite the set of tasks to be performed. Then the master satellite runs its local scheduler to provide a number of local scheduling sub-solutions for them. Then, it builds up a message with the task set and votes the best sub-solution of those that he has found. The vote consists on *marking* with the sub-solution's *figure of merit* every single task scheduled in that particular sub-solution. This SCHEDULING message is sent to its successor in the ring. At each step, each satellite calculates a number of scheduling sub-solutions and votes the corresponding tasks.

Eventually, the message arrives again to the *master* satellite –he will recognize this message as it will contain its own votes–. Now, the master changes the message type to SOLUTION and looks for the tasks that he had voted. He will effectively assign himself those tasks in which his vote is the highest one. The message travels around the ring again and when it turns to the *master* leader, the scheduling process finishes. In Fig. 2.1 a satellite ring formed by 11 nodes is scheduling a set of four tasks. Observe that not all the nodes vote for tasks, as they may not have resources for performing the tasks. In the SOLUTION phase, the previously voted tasks are underlined and bold text is used for tasks finally assigned to the satellite.

Another variation would be that instead of having only one *voting* round, there could be many of them. At each new round, if the satellite has not won the previous *voting* round, it would vote the tasks of other calculated sub-solution. The algorithm converges to a consensual solution when no more sub-solutions can be voted or every satellite has won the previous *voting* round for the tasks it voted.

[5] In fact, the ring formation is a practical architecture for satellite constellations.

Chapter 3

Design and implementation of a distributed task scheduler

3.1 Local-Global implementation

Carrying out a real executable implementation from a more or less theoretical description of an algorithm involves taking some design decisions and choosing programming techniques to achieve an efficient and fair result. A quicker and simpler approach may cause extremely bad results in performance terms. The comparison carried out in this work requires to carefully design the code of both algorithms in order to have an impartial and trustful result.

As a result of this design and implementation process, the first real implementation of this distributed task scheduler has been achieved, being sufficiently complete to derive conclusions on its performance.

In the case of the Local-Global implementation, two clearly different modules have to distinguished: the *Local* entity and the *Global* one. Each one can be described as a unique problem and therefore, each one has to be studied individually.

3.1.1 Local algorithm: Adapting a Prolog satellite local scheduler

The complete independence of this problem from the rest of the system makes it a traditional local task scheduler problem, with the particularity of having to obtain more than one schedule solution for the same set of tasks. This is very important, since it means that most previously designed and implemented local task schedulers applicable to the satellite context can be adapted to satisfy the needs of the *Local* entity in the Local-Global policy. To prove that, different alternatives have been explored. First, an ILOG CPLEX based task scheduler was considered, taking advantage of its performance at solving constrained programming problems. However, a simpler alternative was found: to implement the necessary pieces of code for adjusting an existing local task scheduler used in a previous project of the UPC, the ^{3}Cat-1. This local scheduler perfectly fit with the context of the Local-Global policy, as it was specifically designed for a real nano-satellite mission.

This local scheduler of the ^{3}Cat-1 project was written in Prolog[1], and is able to solve scheduling problems from a given set of tasks with their time and resource constraints within a scheduling window time T_w and a given set of resources. The main characteristics of this scheduler are:

1. **Multi-resource**: the satellite environment defined in this local scheduler has several types of resources that can be required by a task during its execution. In this way, each task requires a certain amount of resource capacity from a certain set of resources. As not all the resources have the same characteristics, the algorithm has to treat them differently. Some examples of the resources that can be defined are: instantaneous power

[1] Prolog comes from the french words *PROgrammation en LOGique*, and is a declarative logic programming language

available, accumulated energy, on-board storage, sub-system availability... A classification between *instantaneous* resources and *cumulative* resources is also defined: the completion of a task returns the capacity of an *instantaneous* resource to its initial value, while this is not necessarily true for *cumulative* resources. Examples of *instantaneous* resources are power and sub-system availability. On the other hand, energy and storage are *cumulative* resources.

2. **Fully elasticity**: the algorithm takes account of the possible variation of the resource capacity, apart from considering the resource consumption required by the tasks. This imposes the use of more constraints, which increases the complexity if the problem to be solved.

3. **Task priority**: The algorithm handle task priorities to allow the solver to remove low-priority tasks when high-priority ones can be allocated and both do not fit in the same schedule. To do this, an iterative approach is implemented: it generates a valid combination of tasks by gradually expanding an initial sub-problem with more tasks. The addition of new tasks is performed in priority order and starts with the simplest sub-problem: allocate resources to the highest-priority task. After solving this problem, the next task in priority order is added to the list and this new sub-problem is solved. The process is repeated until the iterator reaches the problem containing all the tasks or one of the sub-problems is unfeasible. An example with four tasks, **a**, **b**, **c** and **d**, sorted in priority order, is shown in Fig.

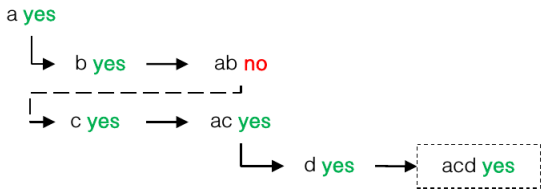

FIGURE 3.1: Example of [3]Cat-1 local scheduler prioritization heuristics

4. **Task definition**: Task objects contain all the relevant parameters and information for the scheduler. Each task is described with a task identifier, the task priority, and the restrictions for temperature, radiation or position domains. Finally, apart from providing the periodicity and the initial delay, the duration of a task is also defined. Periodic tasks that should only be scheduled a given number of times, can provide the number of iterations.

Some of these characteristics have to be adapted for the Local-Global context and extended functionalities needed to actually have a practical *Local* entity have to be added. These changes are described below:

1. **Calculate Δ_i scheduling solutions.** Simply repeating the execution of the task scheduler Δ_i times is not sufficient and neither it is time nor memory efficient. Moreover, it will give Δ_i identical solutions. The solver must be forced to iterate Δ_i times for exploring the solutions space and collect a number of them.

   The main trouble here is to avoid identical solutions, but the solution is in the core of Prolog's logic: as it is a declarative programming language, its execution procedure is based on checking that the input query can be proven as true according to the input facts and rules that constrain and define the problem. Whenever during the execution of the query an inconsistency is found, backtracking is used to a previous state and explores other possible conditions that are able to satisfy that inconsistency. In this case, this built-in backtracking capabilities could be used to find more than one scheduling solution that satisfies all the constrain.

2. **Enabling solutions with "unscheduled" tasks.** At the moment, task priorities are not considered for the Local-Global policy. Hence the [3]Cat-1 *iterator* previously described is not needed. However, without this *iterator* the core solver of this Prolog-based local scheduler is not able to provide a scheduling solution whenever a task set is unfeasible in the given time T_w.

The solution to this issue was to expand the scheduling time in such a way that an artificial extra period of time is added to the initial scheduling window. This period of time is special because it has *infinite* resources, so any task can be scheduled within this time. Therefore, whenever a task or a number of them that cannot be scheduled because of resource or time constraints is found, the solver will simply provide a solution with these tasks placed in this extra time period.

3. **Figure of merit (F) definition and calculation.** A main part of the Local-Global policy is the figure of merit, which describes the goodness of each sub-solution reported to the *Global* layer. A critical contribution of this thesis has been a proposal for the definition of F. This does not mean only to enumerate a set of variables that can describe the solution, but to study these variables and stablish a valid bounded adaptive combination of all of them depending on each parameter's contribution to the schedule.

   The figure of merit is the sole parameter of goodness information about each sub-solution that the *Global* layer will receive. So, it is the unique information it has to obtain the optimal combination of sub-solutions. Instead of having an extensive knowledge of the resources available in each satellite, it only possesses the figure of merit's value. Because of that, the definition must be as complete as possible, containing all and only the variables that really characterize the sub-solution against any other one. F is completely defined in section 3.1.1.

4. **Communication with the *Global*.** The local scheduler has been designed to output the first found solution, but the *Local* entity hast to send a set of sub-solutions to the *Global* layer. For the version implemented in this thesis this communication with the *Global* has been modelled as outputting to a file the Δ_i solutions found. This files generated by all the *Local* schedulers will be later processed by the *Global* process.

Apart from these added abilities, the [3]Cat-1 local scheduler was entirely revised and individually tested for ensuring the best performance.

Formal Local-Global problem description

The definition of the figure of merit F is essential in the Local-Global policy, as it provides the *goodness* of the global solution.

Let all the terms in the multiple-satellite multiple-task scheduling problem be defined:

S Number of satellites in the constellation.

Δ_i Golden number: number of sub-solutions requested to/delivered by satellite i. This value is either set dynamically by the global algorithm or generated statically to equalize the computational load in each local scheduler.

P_{ij} The set of sub-solutions generated by satellite i to a given scheduling problem. This term is defined with the pair $\langle A_{ij}, F_{ij} \rangle$, where

A_{ij} is the task subset[2] included in sub-solution j of satellite i and

F_{ij} is the figure of merit for sub-solution j from satellite i.

T_{begin} Absolute time at which the scheduling window begins.

T_{end} Absolute time at which the scheduling window ends.

T_w Scheduling time window shared across all satellites, defined as:

$$T_w = T_{\text{end}} - T_{\text{begin}} \tag{3.1}$$

Five variables finally form the definition of the figure of merit F value: deadline-based priority, resource utilization, eagerness, satellite processing utilization and responsiveness. This parameters are described below.

[2]Letter A is chosen to prevent confusing the term with time-related variables, denoted with T.

Since the Local-Global policy is aimed at planning tasks within the scheduling window T_w, the algorithm may yield a final combination of sub-solutions which excludes some tasks. This may be caused either due to their execution domains not being within the current T_w (e.g., a point in the orbit which is never reached by any of the satellites in the constellation) or because the tasks are only present in sub-solutions that are not part of the final global one. In order to deal with this behaviour and to include a prioritization method for tasks with shorter deadlines, the global scheduler will consider a fixed number of future scheduling windows and will promote those solutions where there is a task with sorter deadline[3] than that. In order to formulate this feature, let the following terms be defined:

L_{ij} The minimum distance (in time) between a task deadline and T_{begin}, for sub-solution j in satellite i.

N_s Number of periods in deadline prioritization. N_s is a static parameter (see (3.2)).

Therefore, the **prioritization term $\mathbf{D_{ij}}$** can be defined as follows:

$$D_{ij} = \begin{cases} 2 - \dfrac{L_{ij}}{N_s \cdot T_w} & \text{if} \quad L_{ij} \leq N_s \cdot T_w \\ 1 & \text{otherwise} \end{cases} \tag{3.2}$$

Despite the *Global* section of the policy not requiring details about the resources and their capacity allocation to tasks (this is actually what *Local* entities solve), part of a solution's figure of merit (F_{ij}), which represents the goodness of a solution, is computed from each satellite's resource usage that derives from each local plan. In order to complete F definition, the capacities and consumptions of each (local) resource are defined:

R_i Set of resources present in satellite i. Therefore, $\bigcup_i R_i$ represents the total set of resources of the infrastructure.

$c_{ijk}(t)$ Aggregated[4] consumption of resource k for satellite i and sub-solution j at time t.

$m_{ik}(t)$ Capacity of the resource k for satellite i at time t.

$\mathbf{C_{ij}}$ is then defined to provide a metric to evaluate sub-solutions in terms of **resource utilization** as:

$$C_{ij} = \max_t \left\{ \sum_{k \in R_i} \left(1 - \frac{c_{ijk}(t)}{m_{ik}(t)}\right) \frac{1}{|R_i|} \right\} \qquad C_{ij} \in [0, 1] \tag{3.3}$$

Another variable to evaluate the goodness of a given sub-solution is $\mathbf{G_{ij}}$, which somehow represents the **eagerness** of the local satellite i with respect to the execution of tasks in sub-solution j. A sub-solution will be better if it includes more tasks. However, not all satellites are able to perform every task. Some tasks might have constraints that are impossible to meet for satellites (e.g., a position in the orbit that they never reach) or require the use of specialized instruments which are not common for all satellites. Therefore, the figure of merit needs to evaluate the goodness of a sub-solution with respect to the tasks which each satellite has the capability to execute. In order to do so, G_{ij} is defined as follows:

A'_i Subset of tasks that satellite i has the capability to perform. If a given satellite is equipped with a resource k, but this resource does not have enough capacity to perform a given task, this task will still be present in this subset ($a \in A'_i$).

$$G_{ij} = \frac{|A_{ij}|}{|A'_i|} \tag{3.4}$$

Having G and C to evaluate the the number of tasks in a sub-solution and the utilization of resources and D to modify the figure of merit of priority tasks, the following parameters

[3] Deadlines are time values set by ground operators corresponding to the task's $t^{a_j}_{\text{max}}$ value

[4] The sum of resource consumptions by each scheduled task.

will assess the goodness of a sub-solution with respect to the satellite **utilization** ($\mathbf{U_{ij}}$) and **responsiveness** ($\mathbf{E_{ij}}$).

$$U_{ij} = \frac{t_{1(ij)}}{T_{\text{end}}} \tag{3.5}$$

$$E_{ij} = \frac{t_{1(ij)} - t_{0(ij)}}{T_w} \tag{3.6}$$

Where

t_0 Minimum start time among all tasks in sub-solution j, corresponding to $\min_{\forall a \in A'_i} start(a)$.

t_1 Maximum end time among all tasks in sub-solution j, corresponding to $\max_{\forall a \in A'_i} end(a)$.

These five parameters describe in very different contexts the quality of the sub-solution, providing knowledge of each *Local* entity to the *Global* with small overhead. It should be observed that all the parameters are bounded within the interval $[0, 1]$, with the exception of D_{ij}, which is bounded within the interval $[1, 2]$. This could be unnecessary as long as every parameter was proportional to the goodness aspect it represents, but is required by the *Global* entity to allow for an efficient optimization. Moreover, this bounding happens to give each parameter a range of values that can vary at most exactly a value equal to 1, so the relative importance given to each parameter is normalized.

Putting everything together, the **figure of merit F** is finally defined as:

$$F_{ij} = w_c \cdot C_{ij} + w_g \cdot G_{ij} + w_u \cdot U_{ij} + w_e \cdot E_{ij} + w_d \cdot D_{ij} \tag{3.7}$$

The combination of the five parameters is a weighted sum of their values, Where w are the static weights for each parameter, which can modify the by-default balanced relative importance of each parameter.

3.1.2 The Global algorithm

The global scheduling algorithm is basically a combinatorial optimization problem, that can be described as in (3.8). The *Global* layer receives sub-solution sets of size Δ_i. Each solution is described by the pair formed by the scheduled tasks set and its figure of merit $\langle A_{ij}, F_{ij} \rangle$. A potential solution is a combination of at most S sub-solutions, selecting one or none for each satellite. However, it is very important to determine how the combination of sub-solutions can be qualified in terms of the figures of merit of each one of them.

Tthe values could simply be aggregated by summing them, but there is something that must be considered when combining scheduling sub-solutions: a task can appear in more than one sub-solution of the set forming the combination: how does this affect to the quality of this combination as a possible final global solution?

To reflect this degradation in the quality of the global solution, the sum of the F values will be weighted with a multiplying factor that depends on the sum of the number of occurrences of all tasks (N_b) normalized to the maximum number of occurrences, which is equal to $S \cdot |A|$, where S is the number of satellites in the constellation and $|A|$ is the number of input tasks. Below the complete definition of the global problem can be found, where the binary decision variables $x_{ij} = 1$ if sub-solution P_{ij} is part of the final combination and 0 otherwise:

$$\text{Maximize} \qquad r(P) = \left(\sum_{i=0}^{S-1} \sum_{j=0}^{\Delta_i - 1} x_{ij} \prod_{f \in F_{ij}} f \right) \cdot \left(1 - \frac{N_b}{S \cdot |A|} \right) \tag{3.8a}$$

$$\text{where:} \qquad N_b = \sum_{i=0}^{S-1} \sum_{j=0}^{\Delta_i - 1} \sum_{k \in A} B_{ijk} \cdot x_{ij} \tag{3.8b}$$

$$B_{ijk} = \begin{cases} 1 & \text{if } k \in A_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{3.8c}$$

The description of the *Global* entity implementation is divided in three different subsections: the first one is devoted to briefly analyse the computational complexity of the optimization combinatorial problem to be solved, the second is basically an enumeration of the efficiency-oriented design and recursive improvements made on the algorithm, and finally in the third part the coordination with the *Local* entity is explored.

Complexity analysis: an exploding combinatorial problem

The optimization problem described in (3.8) is basically an optimal search problem characterised by mainly three variables: number of satellites S, number of sub-solutions provided by each satellite Δ and the number of scheduling input tasks $|A|$. The entire space of possible combinations is formed by combinations of S elements in which each element is the sub-solution identity selected on each satellite. For instance, for a satellite constellation formed by 5 satellites and having that the first satellite provides 4 sub-solutions, the second provides 3, the third only 2, and both the fourth and the fifth provide 5 sub-solutions, a possible *Global* combination can be represented as the following (see also Fig. 3.2:

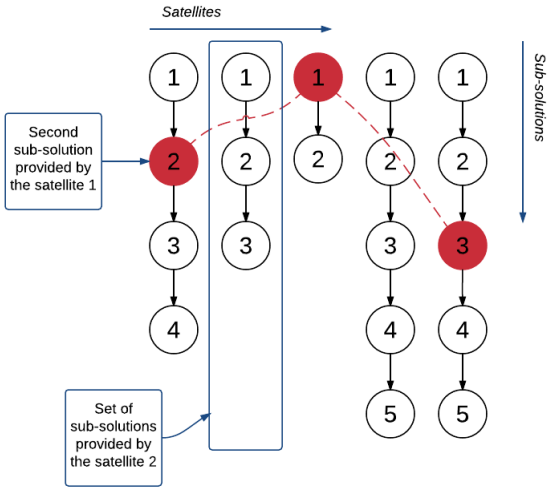

FIGURE 3.2: A graphical representation of the combination of several sub-solutions

$$(20103)$$

This would mean that this particular combination has selected the second sub-solution provided by the first satellite, the first sub-solution provided by the third and the third one from the fifth satellite. Note that it has not selected any sub-solution for the satellites 2 and 4.

Summarizing, it can be easily observed that the entire space of combinations would contain $\prod_{i=1}^{S} (\Delta_i + 1)$ combinations, where, for an homogeneous case where $\Delta_i = \Delta_{\text{system}}$, could be approximated to $\Delta_{\text{system}}^{S}$.

However, this expression does not completely include the complexity of the search. To find out the optimal combination the value of the $r(P)$ function should be calculated for each sub-solution combination, and this calculation depends basically on both the number of input tasks and the number of satellites of the system. This dependence is linear, as one can easily observe that the calculation is fully dominated by the sum of F values, which varies linearly with S (at most exactly S sums must be done) and by the calculation of N_b, which depends linearly with the number of tasks $|A|$ (see (3.8)).

In a basic computational complexity analysis notation, this would lead to a problem dependence on these three variables as shown below:

$$Global(\Delta_{\text{system}}, S, |A|) \in \Theta\big((\Delta_{\text{system}})^S \cdot |A|\big) \tag{3.9}$$

To conclude, it can be observed the evolution of the computational complexity when varying only one input with the others kept as a constant value:

– If both S and $|A|$ are kept constant a **potential variation** is observed.

$$Global(\Delta, S = S_0, |A| = |A|_0) \in \Theta\big(|A|_0(\Delta_{\text{system}})^{S_0}\big) = \Theta\big((\Delta_{\text{system}})^{S_0}\big) \tag{3.10}$$

– If $\Delta_{\text{system}} = \Delta_0$ and $|A|$ is kept constant an **exponential variation** is observed.

$$Global(\Delta_{\text{system}} = \Delta_0, S, |A| = |A|_0) \in \Theta\big(|A|_0(\Delta_0)^S\big) = \Theta\big((\Delta_0)^S\big) \tag{3.11}$$

– If $\Delta_{\text{system}} = \Delta_0$ and S is kept constant a **linear variation** is observed.

$$Global(\Delta_{\text{system}} = \Delta_0, S = S_0, |A|) \in \Theta\big(|A|(\Delta_{\text{system}})^{S_0}\big) = \Theta(|A|) \tag{3.12}$$

Therefore, an exponentially exploding problem is found when solving the *Global* entity combinatorial optimization.

An efficiency-oriented design of the combinatorial search

The most basic and simple solution to this problem would be a brute force approach: exploring the entire space of possible combinations and simply comparing the values of the $r(P)$ function for each of them to finally select the one which maximizes it. However, when facing a problem that depends exponentially as the number of satellites and the Δ value for each one are increased, the brute force approach can be completely unfeasible.

That is why some efficient improvements have been applied and a careful study on the algorithmics and data structures that could be used has been performed.

To begin with, it is very important to observe two characteristics of the function to be optimized, $r(P)$ (see (3.8)):

- It is a bounded function, concretely in the interval $[0, W]$, where W is the sum of the F weights (see (3.7)): $w_c + w_g + w_u + w_e + w_d \cdot 2$ (remember that D_{ij} is bounded in $[1, 2]$ instead of $[0, 1]$).
- It is formed by the multiplication of two different terms: the sum of F and a weighting value depending on the tasks occurrences, which is also bounded in $[0, 1]$

This means that $r(P)$ itself is bounded in the interval $[0, W]$ and, what is even more specific, in the interval $[1, \mathbb{F}]$, where $\mathbb{F}$ is the sum of F of a particular combination.

Therefore, **finding out the way of exploring the space of combinations in a way that $\mathbb{F}$ value is decreasing, will allow to cut the exploration** as soon as a combination such that its $r(P)$ value is equal to its $\mathbb{F}$ value[5], or a combination which $\mathbb{F}$ value is equal or below the maximum $r(P)$ found till now, is found.

[5]This combination will be proved later to be the optimal combination.

A brief justification of these two affirmations is the one that follows: the fact that $r(P)$ value is bounded in the interval $[1, \mathbb{F}]$, and that the combinations space is being explored in decreasing $\mathbb{F}$ makes that no combination with lower $\mathbb{F}$ than any other with $r(P) = \mathbb{F}$ will have an $r(P)$ higher than this one because $r(P)$ is never greater than $\mathbb{F}$ (first cutting context) and that no combination with lower $\mathbb{F}$ than the maximum $r(P)$ value found till now will have a higher $r(P)$ than this maximum, for the same reason (second cutting context). In Fig. 3.3 an example of this early searching cut can be seen.

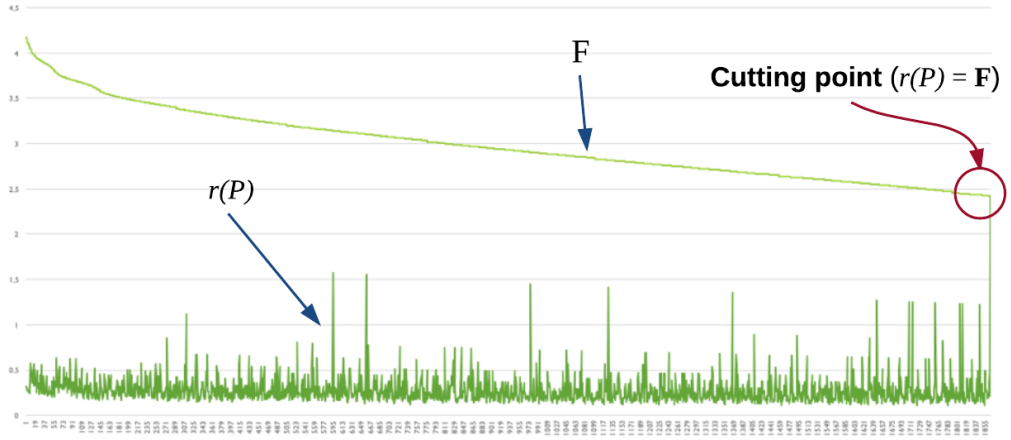

FIGURE 3.3: Efficient optimal combination search

The search heuristics designed and implemented to enable a way of exploring the combinations space in decreasing $\mathbb{F}$ is completely explained in Appendix B. In this project, this optimized design has been programmed using C language, and an efficient ordered list library has been also developed in order to have everything in the code controlled for best performance. In chapter 4, Fig. 4.13, particular results for this implementation compared with a brute force search are shown.

Coordination with the *Local* entities

Concluding this section, this part will be devoted to explain the implementation of the *Global*'s process which is in charge of collecting all the solutions submitted by all the *Local* entities and process them for preparing the optimization search described before.

In the simulations performed in this work, the *Local* entities write out in a text file the Δ_i sub-solutions, detailing for each one the subset of tasks included and the F value.

Therefore, the *Global* has to read that output text files and process them to initialize the internal variables that will represent the satellites' sub-solutions for combining them. If any of the satellites has not been able to send its sub-solutions (e.g., it is down or is not able to communicate with the *Global* entity), the process simply considers it as it had a $\Delta_i = 0$.

At last, this input processing instance orders by descending F value each set of sub-solutions, as the optimized combinatorial search requires.

3.2 Price-based algorithm implementation

In order to implement the algorithm described in [15], some implementation details which are not included in the paper (e.g., the system architecture, the communication cost, the scheduling time window...) have been designed. Also some other aspects of the algorithm, such as

the price calculation or some synchronization problems, have been improved for a best performance in the tests again the Local-Global policy. In this section the main features of this implementation are described, while in Appendix A all the modifications over the original proposal are detailed.

3.2.1 A state graph model

A good approach for implementing a distributed algorithm like this is to model each node in the system as a state graph. In this way, programming the nodes is limited to programming all the possible states in the node, the state changes whenever it arrives to the system a particular message (or any other particular event occurs) with the functionalities performed in any of these states and the variables maintained by the node.

In this case, all the *sellers* can be in two different states: WAITING and LISTENING. In the first one, the satellite is ready to begin a scheduling process as soon as a TASK message arrives to the system. In the second state, the system is performing a scheduling process and the node is waiting to transmit its task bid or surrender if a lower-price bid is received from other satellite. The flowchart of the implemented algorithm can be seen in Fig. 3.4.

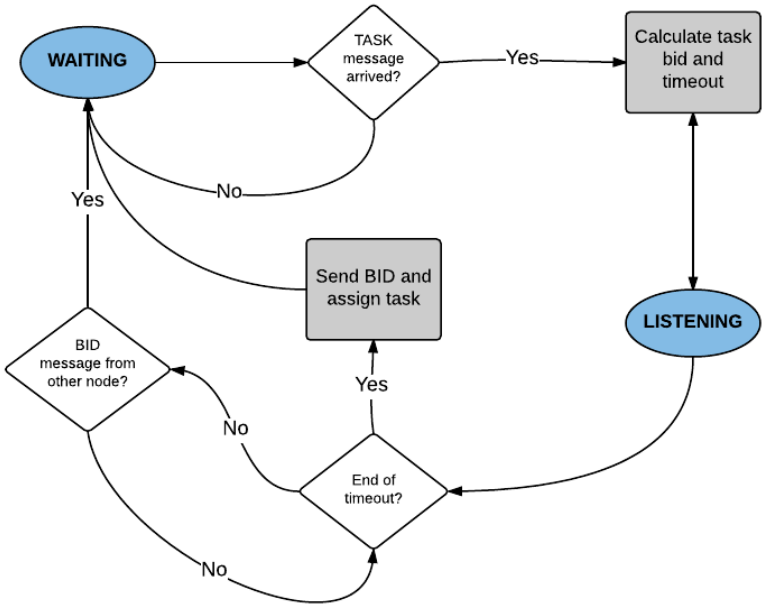

FIGURE 3.4: Price-based policy flowchart

The auxiliary functionalities that have been implemented are mainly two: the price calculation and the task addition to the local scheduler. The first one is a calculation based on the current state of the node and the second is executed whenever the node has won a scheduling process and assigns itself the task, reserving the corresponding resources and refreshing its own state knowledge.

The node's state knowledge will be now described. In this work the designed set of local variables maintained by each satellite are:

1. **Name.** The satellite's ID, used for identifying each satellite's message.
2. **Energy.** The amount of energy that has not been reserving for executing already assigned tasks.
3. **Processor release time.** The time instant in which the processor will finish the current scheduled tasks.

4. **Beginning time.** The initial time of the current scheduling window, used for determining timestamps relative to it, for optimizing the overhead caused by the use of a long absolute time stamp.
5. **Peers list.** The other satellites' IDs present in the constellation. It also includes the communication cost for each one.
6. **Schedule.** The local schedule assigned until that particular moment, generated from the different scheduling processes that have been completed since the beginning of the scheduling window.
7. **Task assignment.** For each already assigned tasks, it contains the ID of the satellite that has been assigned to execute it.

The current implementation has been developed in Erlang code, a functional programming and distributed systems-oriented language. Its distributed nature has allowed to directly simulate different nodes and a real message passing system between the nodes. This truly distributed environment has required a specific architecture design for a better system management.

Because of that, a *leader-slaves* centralized architecture has been implemented. The *leader-slave* relationship is only valid for system registering or de-registering[6], although it is meaningless for the scheduling algorithm.

Another feature of this implementation has been the simplification of the *Listing Phase* (see section 2.2.1). Instead of dividing complex tasks into simpler ones, an already dependence-processed input task sets has been considered. The dependence among two tasks has been modelled as an added task attribute which contains the ID of its *predecessor* task i.e. the task that must be completed to begin this one. In the next section the task dependence's importance on calculating the task bid on a scheduling process is explained along with other implemented modifications in order to optimize the algorithm.

[6]Whenever a satellite wants to enter in the system, it has to communicate it to the current leader, and the same for whenever a satellite leaves the system.

Chapter 4

Results

4.1 Parametrizing a distributed task scheduler

In the last section all the details of both Local-Global and price-based distributed task schedulers implementations carried out in this thesis have been detailed. The present section shows the experimental results of the benchmark of simulations that has been performed.

In order to better understand the numeric data, input variables and how these do affect to the time and memory spent in resolving this particular problem must be analysed.

First of all, we will highlight the parameters that affect the general scheduling problem, independent of the particular algorithm chosen to solve it (whether it is the Local-Global or the price-based or any other one):

- **Number of tasks.** As the number of tasks to be scheduled increase, the complexity of the problem increases, as with the same available resources (energy, time...) more tasks are to be done. In this sense, the problem is harder to solve.
- **Satellite resources.** For the same number of tasks, if the resources available at the system are decreased, the same problem when increasing the number of tasks is found.
- **Number of satellites.** Even though this implies having more resources available and hence more possible solutions, an increasing number of satellites translates to a larger distributed system, potentially leading to an unmanageable system full of resources but unable to schedule any task.

Secondly, the variables that can particularly influence on the performance of the Local-Global policy will be enumerated:

- **Golden index.** This variable is particular to this algorithm. For very high values of the golden index, more sub-solution combinations are possible and, therefore, the complexity of the problem is greater (see (3.9)). This potential dependence is mitigated in part for low values of golden index thanks to the optimizations of the global combinatorial search.
- **Number of satellites.** Although this parameter affects the abstract scheduling problem, it's influence on the Local-Global should be highlighted as the number of sub-solutions combinations to be analysed by the *Global* entity depends exponentially of this variable. This influence is also palliated by the global search.
- **Scheduling window.** If the scheduling window time is increased, the task allocation complexity could also increase, as there are more timing combinations available to *test* in the schedule problem. However, if it is set to a very low value, it could create an unsolvable problem, as the same tasks are trying to be scheduled in an insufficient time period.

Finally, the price-based algorithm is mainly affected by the following variables:

- **Satellites resources.** The results of the algorithm can be quite poor in terms of optimality of the final schedule when the satellite have not much resources, because this lead to high bid calculations and therefore to large scheduling process times and less scheduled tasks.

- **Number of tasks.** The main drawback of price-based is the sequential nature of the scheduling process: each task is processed and scheduled in non-overlapping rounds, so a high number of tasks could lead to an overloaded system not able to schedule tasks at the same velocity that they arrive in the system.

Sweeping these parameters will show the behaviour of each algorithm and its input limits (i.e. the limits on the input's complexity for the algorithm to be able to solve it in a reasonable period of time and spending a reasonable amount of memory). This previous analysis shall be confirmed by the experimental simulations presented in this section.

4.2 Performance Tests

Having theoretically enumerated the critical variables which contribute to the complexity of each algorithm, experimental simulations have been carried out.

A simple tester application has been implemented, in order to have a suitable testing platform to perform extensive simulations and sweeps of the critical variables to capture statistical measures. This application receives the set of input variables (number of satellites, number of tasks, golden index, scheduling window...) and the required sweep. It then simulates an Erlang execution with random generated tasks satisfying the input parameters and, after that, it simulates a Local-Global environment, using the same task set for both. Finally, it measures the time and memory expense and reports it to an output file, in such a format that can be easily processed by a Matlab script. Each simulation is performed several times in order to extract the statistical information: the mean and standard deviation for each of the measured variables.

Six main test sets have been performed: three tests in which one out of the main critical variables (i.e. number of satellites, number of tasks and golden index) have been fixed; and three tests sweeping only one of the three variables in order to find the limits on the problem size for each algorithm to be able to solve it[1]:

1. In the first benchmark, the number of satellites is swept from 1 to 5 and the golden index[2] is swept from 1 to 7, having fixed the number of tasks to 15.
2. In the second benchmark, the number of satellites is swept from 1 to 5 and the number of tasks is swept from 1 to 30, having fixed the golden index to 4.
3. In the third benchmark, the number of satellites is fixed to 5 while the golden index is swept from 1 to 7 and the number of tasks is swept from 1 to 30.
4. The fourth benchmark fixes both number of satellites and golden index to 5 and 4, respectively. The number of tasks is swept from 10 to 100 in steps of 10 tasks.
5. The fifth benchmark sweeps the golden index from 5 to 20 in steps of 5 while fixing the number of satellites to 5 and the number of tasks to 15.
6. The sixth benchmark explores a number of satellites sweep from 1 to 14[3] while fixing the number of tasks to 15 and the golden index to 2.

All the tests were performed with a scheduling time window of 90 minutes and the duration of the tasks was randomly chosen as a value from 1 to 2 minutes. The execution time shown in the plots is referred to the time spent in scheduling tasks during the whole scheduling window. All the satellites in the system have the same amount of resources: the needed power to be permanently processing tasks. The 2D plots reflect the error bars and envelope error functions[4] for the standard deviation calculated from each one of the simulations.

[1] The sweeping ranges and the fixed values are chosen from the most representative cases.

[2] Note that the golden index for the price-based policy is irrelevant.

[3] A distributed satellite environment with more than 14 satellites could not be simulated with the computer available for the tests because of memory constraints.

[4] These envelope functions are plotted with discontinuous lines.

For a better comprehension and analysis of each one of the simulations performed, the results will be shown in three different sections: first the Local-Global policy and the price-based results are separately analysed, and finally both are compared.

4.2.1 Local-Global

The time spent in solving each problem of the benchmark is shown in figures 4.1, 4.2 and 4.3. It can be observed that for this values of number of satellites and golden index the time is almost constant, thanks to the optimizations that have been implemented in the *Global* entity, explained in section 3.1.2. However, the number of tasks causes a linear growth in the execution time.

A similar analysis can be done in terms of memory usage: the global combinatorial search optimizations reduce the effects of increasing the number of satellites (the memory used when sweeping this variable is nearly constant) and the golden index (in this case, the memory increases linearly with a small slope). The memory consumption for each test can be seen in figures 4.4, 4.5 and 4.6 respectively.

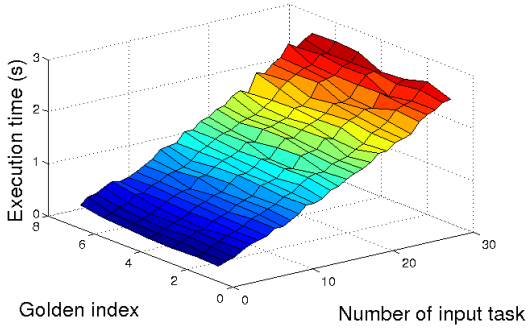

FIGURE 4.1: Local-Global: execution time (satellites fixed)

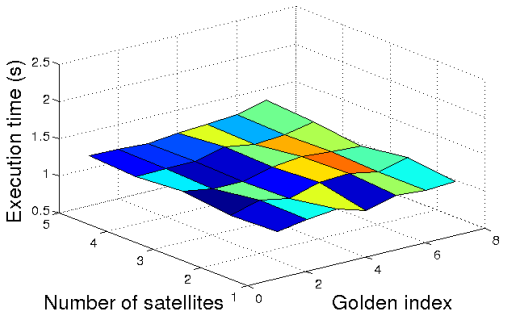

FIGURE 4.2: Local-Global: execution time (input tasks fixed)

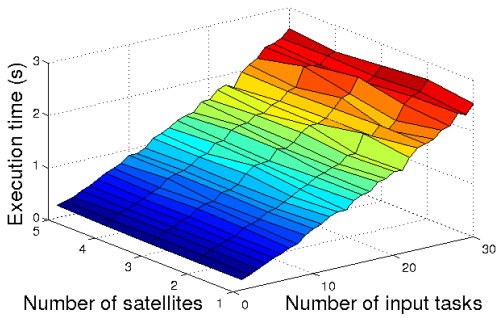

FIGURE 4.3: Local-Global: execution time (golden index fixed)

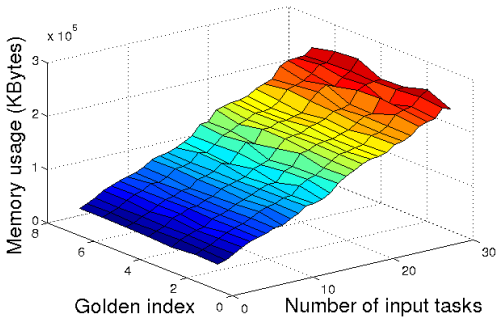

FIGURE 4.4: Local-Global: memory usage (satellites fixed)

However, the improvement achieved for low values of satellites and golden index for time and memory measures is not preserved for high values, as it can be observed in figures 4.7, 4.8, and 4.9. Nevertheless, the linear dependence from input tasks is kept.

Finally, the number of total scheduled tasks (i.e. the number of tasks that are present in the final schedule solution) is represented in figures 4.10 (number of satellites fixed) and 4.11 (golden index fixed). Both plots are very similar and highlight an important characteristic of the Local-Global policy with the implementation carried out in this Bachelor Thesis: for less than 10 input tasks (in this particular benchmark) the system achieves a final schedule which schedules all the tasks available. Then, the system begins to saturate and it is not able to schedule

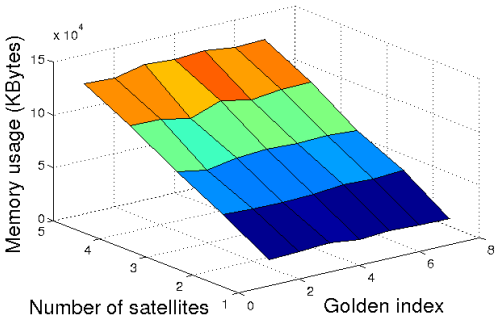

FIGURE 4.5: Local-Global: memory usage (input tasks fixed)

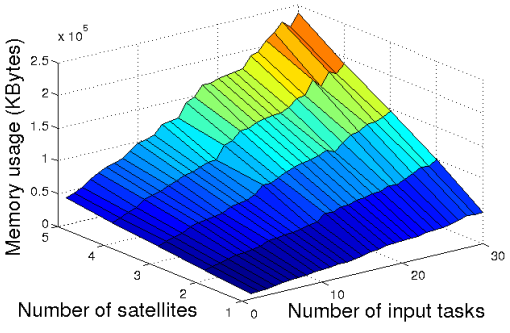

FIGURE 4.6: Local-Global: memory usage (golden index fixed)

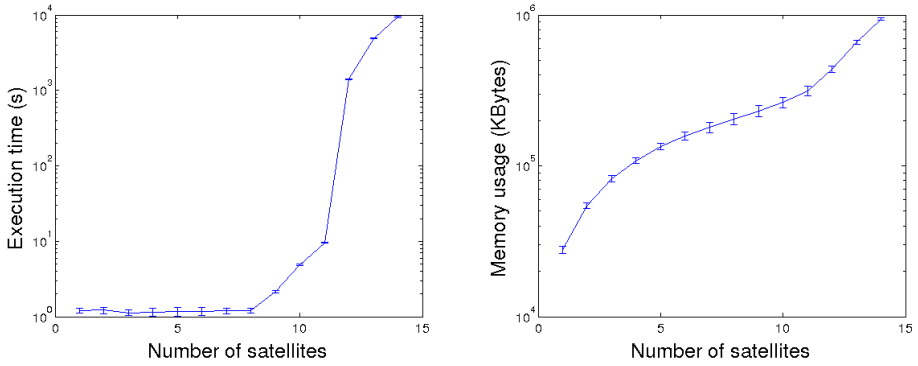

FIGURE 4.7: Local-Global: execution time and memory usage (varying number of satellites)

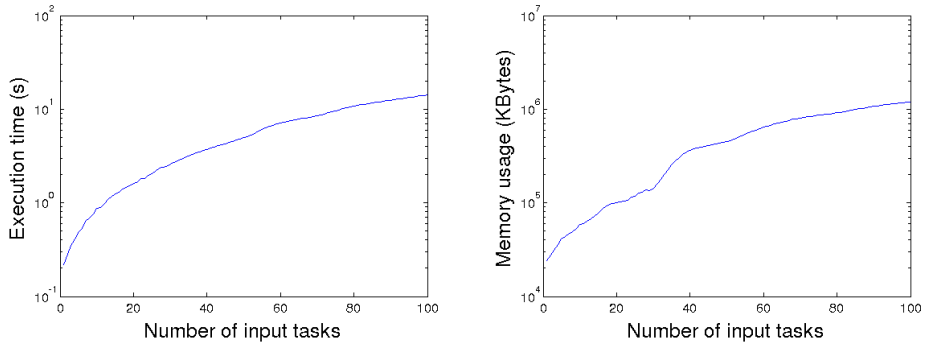

FIGURE 4.8: Local-Global: execution time and memory usage (varying number of input tasks)

all the input tasks. However, the ability of scheduling more tasks at high number of input tasks increases, but not at the same rate than the one of the input tasks. In Fig. 4.12 a final constant saturation point at 80 input tasks can be observed. This behaviour can be optimized by enhancing the *Local* entity so it sends better sub-solutions, as the one implemented for this Thesis still produces some suboptimal sub-solutions that affect the combining capabilities of the *Global*.

In this policy, special attention to the global combinatorial search optimizations has to be paid. In order to quickly analyse the improvement that represents to the entire policy, a performance comparison with a brute-force search has been carried out. Results are shown in Fig. 4.13, where the mitigation (in the optimized version) of exponential increase with the problem size is clearly observed.

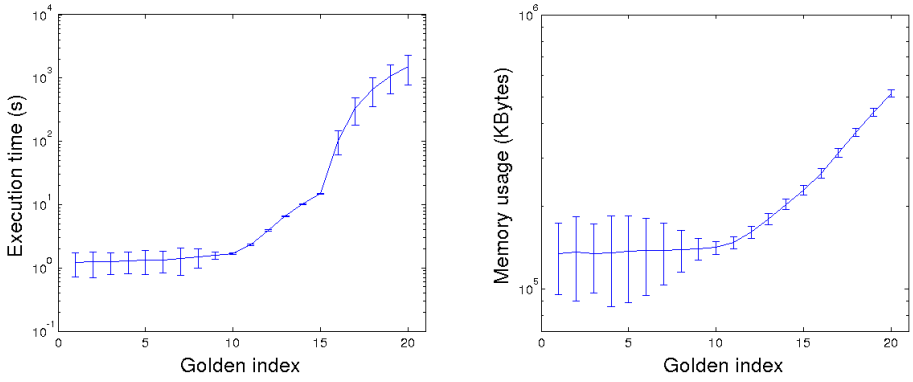

FIGURE 4.9: Local-Global: execution time and memory usage (varying golden index)

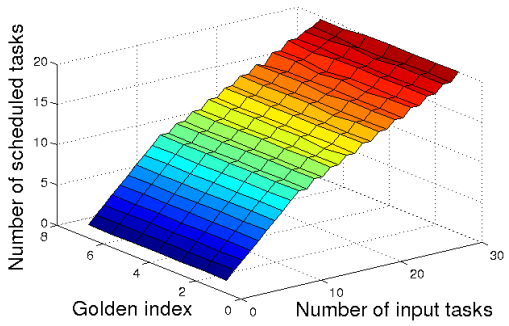

FIGURE 4.10: Local-Global: scheduled tasks (satellites fixed)

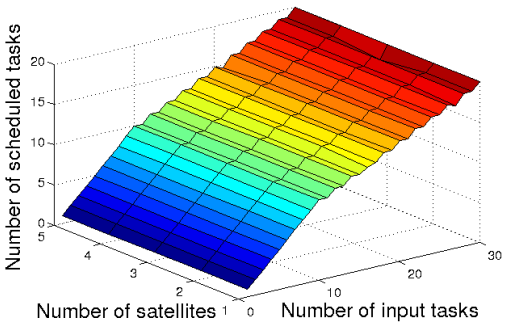

FIGURE 4.11: Local-Global: scheduled tasks (golden index fixed)

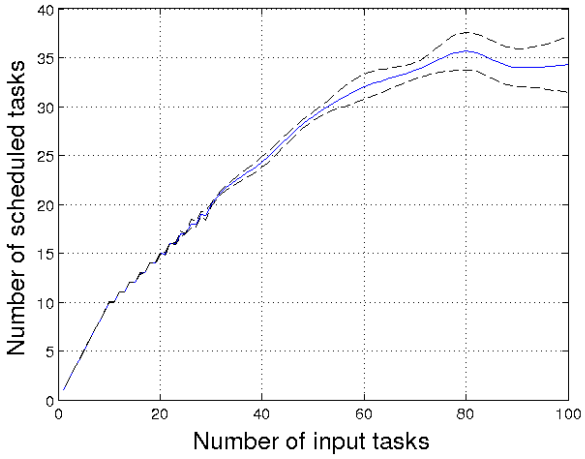

FIGURE 4.12: Local-Global: scheduled tasks for large input tasks range

4.2.2 Price-based

The price-based results are simpler, as there are only two main variables to sweep: number of satellites and number of tasks. The test results are shown in Fig. 4.14 for time measures, in Fig. 4.15 for memory measures and in Fig. 4.16 for scheduled tasks number.

Regarding time and memory, a clear influence of the increase in the number of tasks is observed, while the number of satellites do not affect the final result. The results in the finally

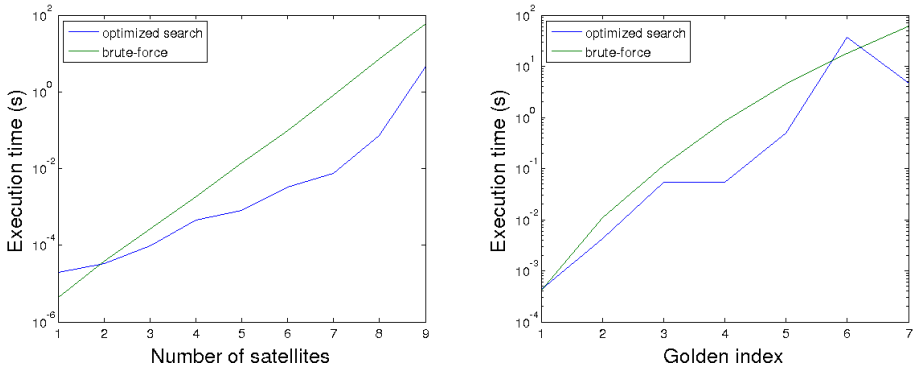

FIGURE 4.13: Global optimization: execution time comparison (combinatorial search vs. brute-force)

scheduled number of tasks must be highlighted: the price-based algorithm achieves to schedule almost all the input tasks for this values. However, when it is swept over the number of tasks at high values (see Fig. 4.17), it can be very clearly observed that the performance decreases completely for more than 30 input tasks.

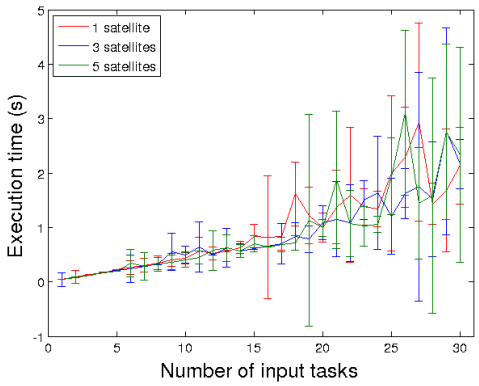

FIGURE 4.14: Price-based: execution time

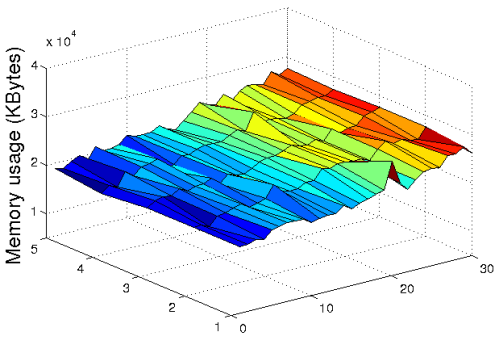

FIGURE 4.15: Price-based: memory usage

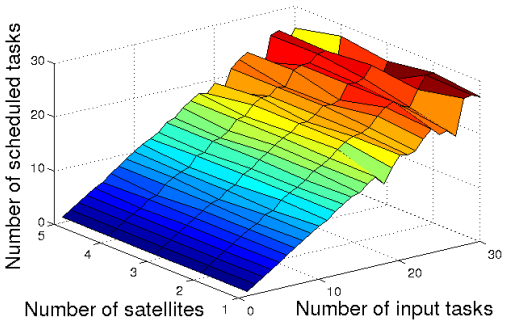

FIGURE 4.16: Price-based: scheduled tasks

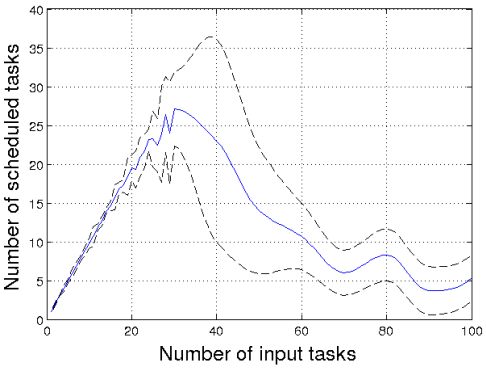

FIGURE 4.17: Price-based: scheduled tasks for large input tasks range

4.2.3 The comparison

As a first concluding result, some comparison representations are shown below, highlighting now the difference in the behaviour of both schedulers that analysing separately each one can remain out of the sight.

Regarding the time spent in solving the input tests, it can be observed in Fig. 4.18 that both increasing the number of satellites or the number of tasks in the system, the price-based scheduler has lower execution times for almost all the cases. Moreover, the Local-Global policy tends to increase exponentially the resources it spends. The memory utilization behaviour is very similar to the time measures (see Fig. 4.19)

However, it can be observed that the Local-Global policy demonstrates its strength by obtaining a constant throughput in terms of scheduled tasks (besides other parameters of the solution quality such as satellite utilization or responsiveness) when increasing the number of input tasks (see Fig. 4.20): it achieves scheduling more tasks than the price-based scheduler for almost all the tests. The range in which the price-based performs better than the Local-Global is from 10 to 40 tasks (i.e. between the first saturation point of the Local-Global and the saturation point of the price-based).

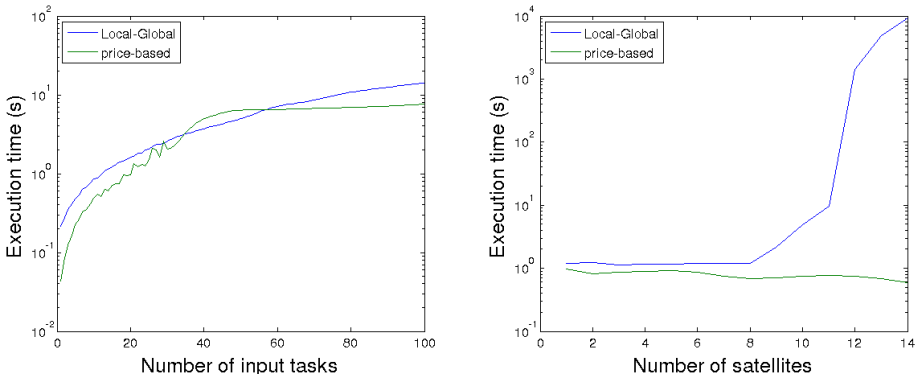

FIGURE 4.18: Local-Global vs price-based: execution time comparison

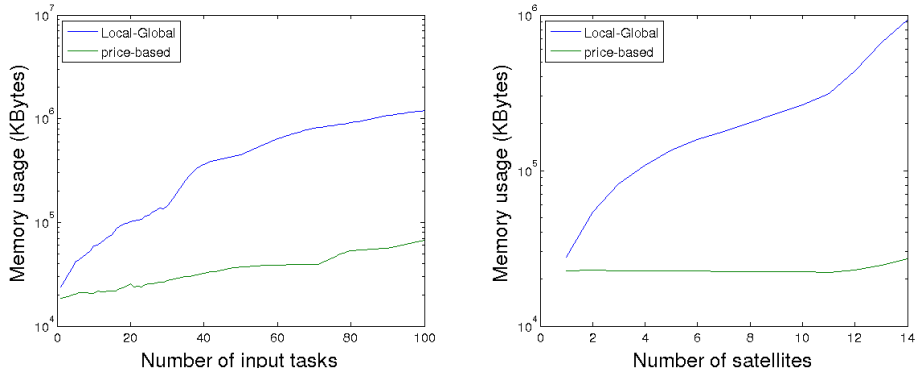

FIGURE 4.19: Local-Global vs price-based: memory usage comparison

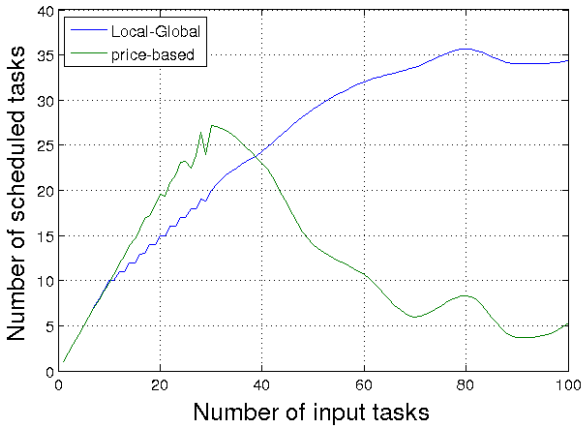

FIGURE 4.20: Local-Global vs price-based: scheduled tasks comparison

Chapter 5

Conclusions

In the context of autonomous distributed satellite missions, new architectures and innovative control systems are still under exploration. The challenges presented by these distributed systems have to be solved to enable the migration from monolithic platforms to more complex architectures. Distributed on-board Mission Planning Systems are a key element of such satellite missions. As one of the core elements of these autonomy systems, distributed task scheduling policies are studied and have been the main focus of this thesis.

In this work the Local-Global policy has been completely designed and implemented. An appropriate state-of-the-art system has also been implemented for comparing and testing the performance of the Local-Global policy. The parameters for the comparison have been defined and large testing benchmarks have been performed for both algorithms, quantitatively measuring the behaviour of each one of them. All these results have to be analysed to obtain relevant conclusions about the development of the Local-Global policy and the future work to be done.

5.1 Analysis of the results and the quality of the solutions

The most significant results have been shown in chapter 4. However, some particularities of each algorithm that cannot be observed in those figures have to be explained and need to be commented in order to properly understand the advantages and disadvantages of each algorithm.

To begin with, it is important to note that the Local-Global policy considers several quality parameters in its optimization process: not only does it consider the amount of scheduled tasks, but also the associated resource consumption, the eagerness of the system to serve the requests, its responsiveness and utility (see definition of F in section 3.1.1). Moreover, the tasks can be scheduled concurrently and any resource definition can be introduced in the problem definition. Finally, the golden index can be optimized for each satellite, either statically or dynamically by the *Global* instance. This is not the case for the price-based algorithm which does not present such characteristics. This can model much better the heterogeneity among the satellites. In fact, homogeneity in the system deteriorates the results of the Local-Global, as very similar sub-solutions are delivered to the *Global* from several satellites, restricting the number of chances to find a good solution.

The price-based scheduler, on the other hand, also has some characteristics that are not found on Local-Global: dependence between tasks and its resulting communication cost (passing the results of a task from one satellite to another) is considered during the schedule generation. Furthermore, the market model allows to include new features in the bid calculation which could take into account other system characteristics. Such redefinition of the bid could be done statically or dynamically.

When both algorithms are compared, it can be observed that the price-based execution times and memory usage are lower than those of the Local-Global, but there are some characteristics to be noted: while the price-based sequential nature of the scheduling process is a bottleneck and causes a saturation point that decreases the performance for high number of input tasks

(Fig. 4.17), the Local-Global's saturation is softer and it achieves to maintain the number of scheduled tasks for this situation (Fig. 4.12). The price-based scheduler also requires good communication among the satellites forming the cluster, a condition which could be hard to satisfy in a highly-constrained bandwidth context such as the open space. The Local-Global's poor performance for high values of number of satellites and golden index can be improved by optimizing the *Local* entity and smartly adapting the values of the golden index.

5.2 Future work

As a core part of a Mission Planning System for distributed satellite missions, this thesis has developed a distributed task scheduler based on the of the Local-Global policy presented in [1]. The results obtained allow to identify future strands of work for the Local-Global policy:

- Although the current performance can be deemed reasonably good as a first approach, an improvement can be obtained by optimizing the *Local* scheduler. This importance of the *Local* scheduler comes from the fact that it is the algorithm that generates the sub-solutions to be combined by the *Global*: to find the optimal combination of sub-solutions, the set of available sub-solutions has to be of high quality and as heterogeneous as possible.

  The optimization of this scheduler should be done on the sub-solutions generation: although each sub-solution, if considered on its own, is a good result, obtained in very low time, the whole set of Δ_i sub-solutions are not optimal, as they may not be the best sub-solutions to be found, and they may contain repetitions among them.

- On the other hand, some optimizations could be implemented for the *Global* combinatorial search. Despite having very good performance when the constellation is composed of heterogeneous satellites, it decreases dramatically for homogeneous cases. These cases need to be studied in detal in order to further improve the algorithm.

- In order to adapt better to heterogeneous systems with satellites which greatly differ in computational capabilities, a dynamic control on the golden index value for each satellite should be designed and implemented.

- Implementing the Local-Global policy in a hierarchical system would allow to explore the scalability in cornerstone scenarios (e.g. constellations with hundreds of satellites). A modified version of the system that could group the satellites in small clusters (with different cluster generation approaches) would allow to explore new system features and situations where local algorithms may perform even better (e.g. clusters with specialized satellites each member would only schedule tasks for what it has been designed for, thus reducing the complexity of the local problem).

- Task dependence defined in the price-based scheduler could be included in the policy, as it is a practical functionality to be implemented for small tasks forming an entire mission or activity.

- The presented algorithm requires a single point of communication during the scheduling phase,a situation that can be slightly unrealistic in some scenarios. A more constrained communication model should be implemented to see how this affects the Local-Global performance (with a communication model similar to that of [17]).

Bibliography

[1] C. Araguz et al. "On autonomous software architectures for distributed spacecraft: A Local-Global Policy". In: *Aerospace Conference, 2015 IEEE*. 2015, pp. 1–9. DOI: 10.1109/AERO.2015.7119182.

[2] Marco D'Errico. *Distributed Space Missions for Earth System Monitoring*. Vol. 31. Springer Science & Business Media, 2012.

[3] Alessandro Golkar. "Federated Satellite Systems: a vision towards an innovation in space systems design". In: *9th IAA Symposium on Small Satellites for Earth Observation, International Academy of Astronautics, Berlin, Germany*. 2013.

[4] Joseph YT Leung. *Handbook of scheduling: algorithms, models, and performance analysis*. CRC Press, 2004.

[5] George Coulouris et al. *Distributed Systems: Concepts and Design*. 5th. USA: Addison-Wesley Publishing Company, 2011. ISBN: 0132143011, 9780132143011.

[6] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN: 0132392275.

[7] David P Anderson, Eric Korpela, and Rom Walton. "High-performance task distribution for volunteer computing". In: *e-Science and Grid Computing, 2005. First International Conference on*. IEEE. 2005, 8–pp.

[8] Krithi Ramamritham and John A Stankovic. "Dynamic task scheduling in hard real-time distributed systems". In: *IEEE software* 3 (1984), pp. 65–75.

[9] Jia Yu and Rajkumar Buyya. "A taxonomy of scientific workflow systems for grid computing". In: *ACM Sigmod Record* 34.3 (2005), pp. 44–49.

[10] S. Giannecchini, M. Caccamo, and Chi-Sheng Shih. "Collaborative resource allocation in wireless sensor networks". In: *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*. 2004, pp. 35–44. DOI: 10.1109/EMRTS.2004.1310996.

[11] Jinghua Zhu, Jianzhong Li, and Hong Gao. "Tasks allocation for real-time applications in heterogeneous sensor networks for energy minimization". In: *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on*. Vol. 2. IEEE. 2007, pp. 20–25.

[12] Jonathan S Anderson, Binoy Ravindran, and E Douglas Jensen. "Consensus-driven distributable thread scheduling in networked embedded systems". In: *Embedded and Ubiquitous Computing*. Springer, 2007, pp. 247–260.

[13] Giuseppe Colistra, Virginia Pilloni, and Luigi Atzori. "The problem of task allocation in the internet of things and the consensus-based approach". In: *Computer Networks* 73 (2014), pp. 98–111.

[14] Virginia Pilloni et al. "A decentralized lifetime maximization algorithm for distributed applications in wireless sensor networks". In: *Communications (ICC), 2012 IEEE International Conference on*. IEEE. 2012, pp. 1372–1377.

[15] N. Edalat et al. "A price-based adaptive task allocation for Wireless Sensor Network". In: *Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on*. 2009, pp. 888–893. DOI: 10.1109/MOBHOC.2009.5337039.

[16] Han-Lim Choi, Luc Brunet, and Jonathan P How. "Consensus-based decentralized auctions for robust task allocation". In: *Robotics, IEEE Transactions on* 25.4 (2009), pp. 912–926.

[17] Grégory Bonnet and Catherine Tessier. "Coordination despite constrained communications: a satellite constellation case". In: *3rd National Conference on Control Architectures of Robots*. 2008, pp. 89–100.

[18] Virginia Pilloni and Luigi Atzori. "Deployment of distributed applications in wireless sensor networks". In: *Sensors* 11.8 (2011), pp. 7395–7419.

[19] F. Bonomi and A. Kumar. "Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler". In: *Computers, IEEE Transactions on* 39.10 (1990), pp. 1232–1250. ISSN: 0018-9340. DOI: 10.1109/12.59854.

[20] Howard Tripp and Phil Palmer. "Stigmergy based behavioural coordination for satellite clusters". In: *Acta Astronautica* 66.7–8 (2010), pp. 1052 –1071. ISSN: 0094-5765. DOI: http://dx.doi.org/10.1016/j.actaastro.2009.09.017. URL: http://www.sciencedirect.com/science/article/pii/S0094576509004561.

[21] Alejandro Cornejo et al. "Task Allocation in Ant Colonies". English. In: *Distributed Computing*. Ed. by Fabian Kuhn. Vol. 8784. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 46–60. ISBN: 978-3-662-45173-1. DOI: 10.1007/978-3-662-45174-8_4. URL: http://dx.doi.org/10.1007/978-3-662-45174-8_4.

[22] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. "Distributed algorithm design for multi-robot task assignment with deadlines for tasks". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 3007–3013.

[23] Leslie Marsh and Christian Onof. "Stigmergic epistemology, stigmergic cognition". In: *Cognitive Systems Research* 9.1 (2008), pp. 136–149.

Appendix A

Modifications over the original price-based algorithm

As it has been stated in section 3.2, for the sake of performance of the implementation of the price-based algorithm chosen for the comparison with the Local-Global policy, some improvements to the original algorithm have been implemented. In this appendix these modifications are discussed in detail.

First of all, to better compare both Local-Global and price-based algorithms, a time constraint has been added to this algorithm: the **scheduling window**. This means that a scheduling process is bounded by this time window, in the sense that all the tasks pertaining to a scheduling execution arrive to the system and are executed during this period of time.

Secondly, the **communication cost** for transmitting any task result from one satellite to another, which is present in the bid calculation, has been defined as a value proportional to the distance in kilometres among both satellites. In addition, the calculation of the communication cost from the task's dependence with its predecessor has been implemented in the following way: a task bid's communication cost is calculated as a value proportional to the distance from the satellite performing the calculation and the other one that has been assigned to execute the predecessor task, if any.

Finally, it has been proposed and implementation a different but similar **price calculation**. The main problem of the reference paper's implementation is that the given definition results in a non-bounded bid. This causes that in some situations in which all the satellites have very constrained resources all of them would calculate a very high bid value, leading to a very long waiting time, which would mean that the system would be idle in the scheduling process too much time. Moreover, when two satellites calculate exactly the same bid, no tie-breaker policy is defined.

To solve these problems, an optimized bid definition has been designed and implemented, modifying the following bid's components:

Base Price (BP): To allow a bounded value, a gaussian function has been used. Parameters a and b modify the maximum value and the velocity of price decay (in fact b is the gaussian's variance) as the available energy is incremented, respectively. Its value for the node i and the task a_j (with task size l^{a_j}) is defined as:

$$BP_{ij} = a \cdot \exp^{-\left(\frac{1-\left(l^{a_j}/E_i\right)^2}{b}\right)} \quad \text{if } E_i \geq l^{a_j} \tag{A.1}$$

Communication Cost (CC): this value has been also redefined to be bounded supposing that all satellites in the system are in the same LEO[1] orbit and because of that are not further than CC_{max}, which is the diameter of the surface containing the orbit, approximated by an sphere.

[1] LEO stands for Low Earth Orbit

$$CC = \begin{cases} 0 & \text{if no satellite assigned to } pred(a_j) \\ \dfrac{d(i,k)}{CC_{\max}} & \text{if satellite } k \text{ has been assigned to } pred(a_j) \end{cases} \tag{A.2}$$

Task Deadline (TD): since the *Listing Phase* is not necessary in this implementation, the Task Deadline has been redefined as a value attached to each task message that represents the instant of time in which the task execution should be finished.

With these modifications, the proposed price calculation is the one of (refMBPrice2), where RT_i is the processor i release time.

$$P_{ij} = \begin{cases} CC + BP_{ij} + \dfrac{1}{DL - RT_i + 1} & \text{if } BP_{ij} \text{ defined and } DL \geq RT_i \\ 2 + a & \text{otherwise} \end{cases} \tag{A.3}$$

A small random time is added to the waiting time calculated from this value of P_{ij}, as a simple tie-breaking policy.

Having in mind all these modifications, each satellite's execution procedure can be reduced to the following description: a satellite can be in two possible scheduling states: WAITING and LISTENING. In the first state, it is ready to receive task messages to trigger a new scheduling event, during the scheduling window time. When a task message is receive, the node goes to LISTENING state. In this state it first calculates the bid following the definition shown in (A.3) and waits for a waiting time T_{wait} proportional to the P_{ij} calculated plus a small random time. If it receives within this time a bid for this task, it surrenders, saves the ID of the winner satellite for future references and returns to WAITING state. Else, it considers itself the winner of that round and schedules the task, reserving the corresponding energy. Finally, it returns to the WAITING state.

Appendix B

Search heuristics for the Global algorithm

In section 3.1.2, the efficient-oriented implementation of the combinatorial optimization problem that has to be solved by the Global algorithm is described. A way of exploring the space of combinations in decreasing $\mathbb{F}$ is needed to efficiently search for the optimal combination. In this appendix, the designed search heuristics to solve this problem are explained.

First of all, it must be assumed that the S lists of Δ_i sub-solutions delivered by each satellite are ordered by decreasing F. This means that combination $(11...1))$ is the one which has the highest $\mathbb{F}$ value. However, this does not imply that this combination is the one that has a maximum $r(P)$ value.

Therefore, to explore the combinations space in decreasing $\mathbb{F}$, the combination $(11...1))$ has to be be the first one. The way to follow the exploration is explained below.

Let two key concepts for this explanation be defined (see also Fig. B.1):

- A combination's **successor** is any other combination that is the selection of exactly the same sub-solutions as the first one except by one and only one satellite, for which it has selected the sub-solution immediately following the one that has selected the first combination in order of decreasing $\mathbb{F}$. For instance, the combination $(\mathbf{3}3240))$ is the successor of $(23\mathbf{2}40))$. Observe, however, that it is also successor of $(3\mathbf{2}240))$.
- In the same way, a combination's **predecessor** is any other combination that is the selection of exactly the same sub-solutions as the first one except by one and only one satellite, for which it has selected the sub-solution immediately preceding the one that has selected the first combination in order of decreasing F. $(23\mathbf{2}40))$ and $(\mathbf{2}2250))$ are predecessors of $(2\mathbf{3}2\mathbf{5}0))$.

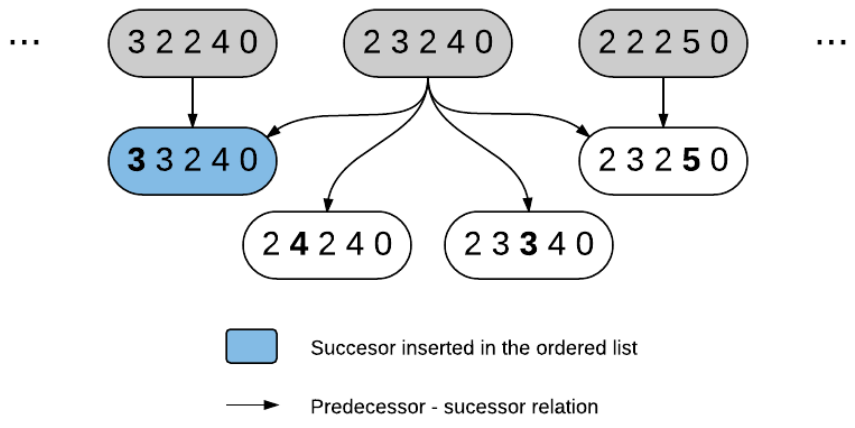

FIGURE B.1: Predecessor and sucesors in the combinations space

Therefore, it can be proven that the exploration of the combinations space will be performed in decreasing $\mathbb{F}$ with the following procedure: for each combination being processed its successors are inserted in an ordered list and the first element in the list is selected as the following combination to be processed, starting from the combination $(11...1)$).

Noteworthy, a combination has more than one predecessor, so it would be inserted more than once in the ordered list, and hence it would be unnecessarily processed more than once. In order to insert the elements once and only once in the ordered list, each combination selects the successors to be inserted in the following way (see also the example of Fig. B.1): from all the set of successors of a combination c of the form $(11...c_k...c_n))$ where $c_k \in [2, \Delta_i]$ and $k \in [1, n]$, the inserted combinations subset is formed by all the successors of c which differ from c in any of the first k components / satellite sub-solutions selections (e.g., the combination (11426) would insert (21426), (12426) and (11526)).