UNIVERSITAT POLITÈCNICA DE CATALUNYA

A scalable distributed autonomy system for fractionated satellite missions

A Degree Thesis Submitted to the Faculty of the Escola Tècnica 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:
Carles ARAGUZ LÓPEZ
Elisenda BOU BALUST

Reporting advisor (ponent):
Dr. Eduard ALARCÓN COT

January 20, 2016

For/Dedicated to/To my…

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Abstract

Escola Tècnica 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

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too…

Contents

List of Figures

List of Abbreviations

ABS	**A**ndroid **B**eyond the **S**tratosphere
UPC	**U**niversitat **P**olitècnica de **C**atalunya
IoT	**I**nternet of **T**hings
EST	**E**arliest **S**tart **T**ime
LST	**L**atest **S**tart **T**ime
LEO	**L**ow **E**arth **O**rbit

Chapter 1

Introduction

The main aim of this thesis is to implement and compare a distributed task scheduler proposed in [1] with some distributed scheduling algorithms to determine whether it performs good at solving the problems it has been designed for.

A scheduling algorithm is a type of optimization problem consisting on distributing tasks over a determined time window (and maybe also among a number of workers, although there could be only one) while satisfying some resource or time requirements, such as task deadlines or processing resources in the worker. In our case, we are studying a particular task scheduler which has the particularity of being *distributed*: this means that in this case the scheduling problem is not solved by a sole machine but by a group of them.

This thesis has been developed in the context of the Android Beyond the Stratosphere (ABS) UPC project, which will be explained later in the document. But before that, a brief description of what an task scheduler algorithm is and how complex it can be will be done. The ABS proposal and other state-of-the-art task allocating algorithms will be presented, and the details of the implementation of the compared algorithms will be explored.

The most important desired result is to conclude whether the ABS proposal behaves well in a simulated environment –similar to that which is designed for– compared to other algorithms or not.By *behaving well* we mean that it achieves to schedule the tasks among the workers in a reasonable time and using a reasonable quantity of the limited resources of each node. For accomplishing this goal we have tested the algorithms with a benchmark of multiple simulated scheduling problems and studied the results, which are shown at the end of the present document, before concluding and stating the future work to be done.

1.1 Scheduling tasks: a hard optimization problem

The core of this thesis is a typical optimization problem: to schedule some input tasks with varying or fix 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 turns out into a hard problem that has to be solved with advanced programming techniques such as constraint-based programming or heuristics. In fact, multiprocessor scheduling and some other scheduling problems are NP-hard optimization problems.

The problem that our algorithm must solve can be reduced to a multiprocessor scheduling problem (so it can be demonstrated it is a NP-hard problem): given a set A of jobs where job a_i has a processing time equal to l^{a_i}, arrives to the system at $t_0^{a_i}$ and has to be finished before the deadline $t_{\max}^{a_i}$, and a set of workers W, which is the optimal schedule of the maximum of jobs such that all resource and time requirements are accomplished? The optimality here can be described as a set of attributes of the solution that are chosen for a particular problem. In the algorithm description section we will explain the parameters that we have chosen as *optimality testers*.

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.: EDF, RM. . .). However, they are meanly mono-processor algorithms, i.e. the algorithm runs in a unique machine and if the solution applies to other machines, it is sent to them. The possibility of having a distributed scheduler (as we are looking for) has not been a research field of interest till the recent growth of distributed systems with increasing processing capacity, very present in the IoT (wireless sensor networks) and robotics.

This last case applies to our issue: we want to solve an NP-hard problem collaboratively, i.e. the nodes 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 in a highly-constrained environment, which will make very difficult the communication among them. It can be easily concluded that *distributing* a task scheduler adds much complexity to an already computationally hard problem. But it is not necessarily true: one of the most widely used computer science paradigm is "divide and conquer". Distributing a problem among several nodes, if done in an efficient manner, should at last be also very helpful.

1.2 Distributed systems

The objective of this section is twofold: to have a basic knowledge of what is and what is not a distributed system and to describe the characteristics of distributed programming that make it quite different from typical mono-processor programming. This is important as long as this Thesis' core is a *distributed* algorithm for a *distributed* system.

In [2] a distributed system is described as *"a software system in which components located on networked computers communicate and coordinate their actions by passing messages"*. This tell us about the basis of these type of software systems: the communication. By *talking* with the other nodes in the network a component of the system can take into account the shared knowledge about the others' state 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. However, although sometimes it is needed to have a single node doing some special functions to control the state of the entire system (and, in fact, leader election and recovery is one of the most critical functionalities in distributed programming), 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).

1.2.1 Distributed algorithms

Distributed algorithms are not just a sequential single-processor code that has been decomposed into several pieces and assigned to a number of processors. Distributed algorithms are pieces of code designed for being running on several hardware or software nodes, having as a key function the intercommunication between nodes.

The autonomy of the nodes running the distributed code modifies some typical programming patterns. Also synchronism problems or node failures appear. That means new challenges and the apparition of different approaches: problem's solution achieved by *consensus* among nodes, clock synchronization algorithms, data consistency and reliable communication, redundancy and failure recovery... [3]

In our problem, we 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 the key of the efficiency of the resulting algorithm: how it take profit of the intercommunication to precisely distribute the work without loosing crucial information for obtaining the optimal scheduler.

1.3 The Android Beyond the Stratosphere Project

Before deepening on the distributed task scheduler theory, let us contextualize a little more the work that is being presented within this report. This Bachelor Thesis has been carried out in the context of a bigger project: Android Beyond the Stratosphere. This project, performed at the Laboratory of Small Satellites and Payloads of the Technical University of Catalonia UPC BarcelonaTech, aims to design 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 make possible 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 requires some advanced scheduler algorithm that allows the system to distribute the tasks among all the satellites forming the system in a fair and optimal way (in terms of time and resource consumption). This task scheduler is the main objective of this Bachelor Thesis and the previous work that has already been carried out.

We could think of a centralized scheduler running in a *leader* satellite, but the complexity of the problem and the constrains on processing capacities and resources that these low-cost nano-satellites have make this conception practically impossible. Furthermore, it is more interesting in terms of the autonomy of the distributed system that the system itself can be the responsible of planning how and when it will do the tasks that must be executed to complete its mission. Adaptiveness and low resource consumption: these are the key concepts in a system like this. A distributed algorithm running on every satellite composing the constellation will have the advantage of sharing the computational effort of solving these problem while providing pretty high autonomy to each satellite.

1.4 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. However, some design work has also been carried out in the distributed software architecture, which is intended to be as generic as possible, so it can support any required architecture –from fully-fractionated satellite structure to a highly unconstrained satellite swarm–.

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.

1.4.1 The Local-Global proposal

Last year a distributed task scheduler was proposed for the ABS project [1]. The main characteristics of this algorithm were described, and its logic completely explained. In this Bachelor Thesis it has been fully implemented, and below we will make a brief description of it.

This scheduling policy aims to provide 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 multiple-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 combination. At last, the final scheduling solution is sent back to every *Local* entity.

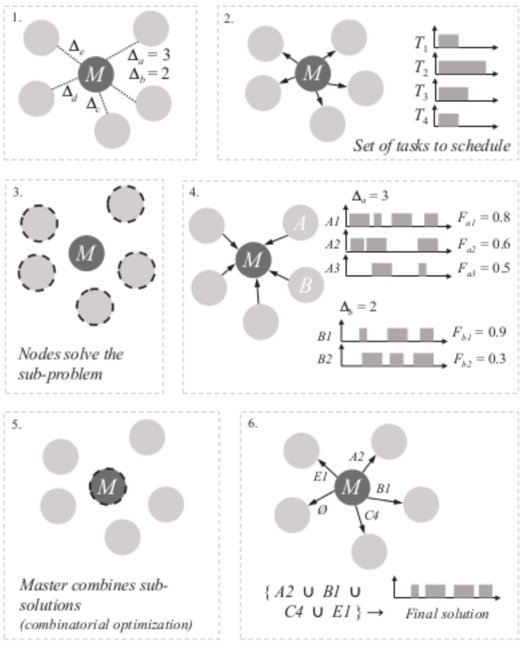

FIGURE 1.1: Local-Global policy steps (as extracted from [1])

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, by combining a number of variables 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 solve the possibly high 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 we will now specify the procedure it follows in six steps (see also Fig. 1.1):

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 scheduloing algorithm) and discard tasks not fitting within it. 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 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 answer each *Local* entity with the identity of its selected sub-solution, if any.

   In section 3.1 we will focus on the implementation that has been developed in this Bachelor 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

As it was mentioned before, the existing distributed system architectures can be classified according to the grade 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, but it cannot be 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. While searching for state-of-the-art algorithms that could be compared with the Local-Global proposal, we found from high-complexity fully-centralized algorithms that may be solved using dynamic or constraint-based programming (meanly for grid-computing or High Performance Computing, as in [4–6]), to low-computational decentralized schedulers [7, 8].

In the last years, distributed task scheduler research has mainly worked over three different paradigms: **negotiation or consensus** algorithms, in which nodes in the system *agree* the final schedule –it can be done *offline* as well as *online*–, **centralized** optimized schedulers –where the degree of centralization can vary– and more recently **bio-inspired** approaches –which take profit from observing and imitating the behaviour of the distributed systems present in the nature–.

Of these three categories, the one which almost unexplored is the third one: the research on it is almost only theoretical and only simple and descriptive approaches have been published. Usually they propose a procedure that imitates an observed behaviour on ant colonies or other such distributed *nature* systems. For instance, an adaptation of the stigmergy used by ants to stablish an optimal path is presented in [9] and a resources balancing technique extracted from the one observed in ant colonies is described in [10]. Although these are very interesting approaches, they are still not quite practical for a comparison like the one we want to do.

Completing the classification, our Local-Global policy could be described as a centralized distributed algorithm, while consensus-based distributed task schedulers would be a research-mature third option, fitting our needs for the comparison. As these are actually candidates for the comparison, we will briefly describe in the next section some of the most interesting approaches that have been found, and what are the reasons for having chosen a particular one.

2.2 Comparison: algorithms chosen to test the Local-Global

Having explored the paradigms that are being investigated in the distributed task scheduler field, we must find out a state-of-the-art algorithm that can be compared with the

Local-Global proposal: it should be designed for a similar context and resources requirements.

As it has been already stated, the research in the field of distributed task schedulers is still in its first steps, having that the computational capabilities of nodes in typical distributed systems has been traditionally very low to consider a fully-autonomous system-wide distributed task scheduler running on every node.

The criteria used for choosing the state-of-the-art algorithm to compare the Local-Global with was that it should be as representative of distributed algorithms as possible. Papers presenting such kind of distributed task scheduler are only a few, and sometimes the context is too different to compare our algorithm with (nearly infinite bandwidth, multiprocessor scheduling...).

In fact, the context in which the Local-Global policy is meant to work is a very special one: it is a highly-constrained environment in terms of resources and communication, a medium 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 our search field: algorithms thought to run on practically infinite communication bandwidth grid computing platforms (as the one presented in [11]) cannot be compared with the Local-Global.

We cannot also use for the comparison algorithms that are in fact designed to be used for distributed Operating Systems or deploying distributed heavy applications, where the timing requirements are highly-constrained but the communication is practically unlimited [12, 13]. There are also other invalid approaches such as that presented in [14] as it does consider resources requirements in a simple way and the problem resolution is fully centralized, with no participation of the system nodes. However, it presents an auction decision model, something which is also present in the final chosen algorithm.

Two very interesting approaches can be found in [15] and in [16]. The contexts are very similar to the ABS satellite constellation. In the first proposal, a combination of consensus and auction-based decision making is used to coordinate a fleet of autonomous vehicles with two decentralized algorithms. Nevertheless, the task description is more complex than the one we need for the ABS case. The second paper describes a communication-constrained based task allocator in the same context of that on ABS project: a satellite constellation. However, the local scheduler of each satellite is not described in the paper, which makes very difficult to be able to use it for comparing with the Local-Global.

Finally, an distributed-architecture-agnostic market-based approach was found in [17]. The description of the tasks is adequate to what we need and its simplicity yet completeness shows a fully decentralized distributed proposal good for our comparison.

Additionally, a ring-architecture task scheduler algorithm has been designed from the leader election algorithm described in [3, 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 market-based approach

The proposal of [17] is an adaptive task allocation scheme, which is presented to be used on wireless sensor networks, which is a similar context to that of our initial problem: a resource-constrained environment. A market-based architecture is proposed, in which each node is modelled as a seller who calculates the price for deploying a specific task and offers 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. It is important to observe that it schedules the tasks *on the go*, i.e. it is an *online* task scheduler.

In this section a brief description of the presented algorithm, as described in the article mentioned before, is presented, and in 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 broadcasted the task information. Each node calculates the price it will offer based on his 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 the paper 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.** It is proposed a decomposition of an initial set of tasks into a set of smaller sub-tasks forming 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 be begun for letting 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 broadcasted to the nodes, ready for the task-assignment phase.

2. **Price-Based Task Assignment Phase.** As it have been stated before, this phase is executed *online*, so the tasks are scheduled as they arrive in the system. The core of this phase is the price formulation which moreover 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 that the proposal includes for the price formulation 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 of (2.3), with DL 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.

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

$$\lambda(t, DL_j) = \begin{cases} k(t - DL_j), & \text{for} \quad t \geq DL_j \\ \epsilon & \text{for} \quad t \leq DL_j \end{cases} \qquad \gamma(t, RT_i) = \begin{cases} k(t - RT_i), & \text{for} \quad t \geq RT_i \\ \epsilon & \text{for} \quad t \leq RT_i \end{cases}$$

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. Here we will focus on the more efficient decentralized method, as it is the one which we will implement. The core of this procedure is that instead of transmitting to the client 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 Another ring algorithm

The leader election algorithm found on [3] is a quite simple one, but it is very representative of a typical distributed architecture: the ring structure. We will now describe the design that has been performed in this work to adapt this algorithm for having a distributed task scheduler. Instead of agreeing on who will be the next leader after a leader failure, we want to agree on a schedule for the whole system.

We assume that the satellites in the system are physically (in fact, the ring formation is a practical architecture for satellite constellations) 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 goes round the ring again and when it turns to the *master* leader, the scheduling process has finished.

Another variation would be that instead of having only one *voting* round, there would be many of them. At each new round, the satellite would vote the tasks of other calculated sub-solution, if the first task set he voted has been won by other satellite. The algorithm converges to a consensual solution when no more sub-solutions can be voted or every satellite has won the task set it has voted.

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. Of course, the comparison we are carrying out requires to carefully design the code of both algorithms for having an impartial and trustful result.

As a result of this rigorously design and implementation process, we have achieved the first real implementation of this distributed task scheduler, being sufficiently complete to conclude with strong arguments about the behaviour of it.

In the case of the Local-Global implementation, two clearly different modules can be told apart: 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 Adapting a Prolog satellite local scheduler

If we go back to the description of the Local-Global policy of the section 1.4, we can observe that the problem to be solved by each *Local* entity in the Local-Global policy is fully system-agnostic and can be expressed as it follows: given a set A of tasks where task a_j has a processing time equal to l^{a_j}, arrives to the system at $t_0^{a_j}$ and has to be finished before the deadline $t_{\text{max}}^{a_j}$, and a set of resources R_i which will constrain the set of possible schedule solutions, obtain a sub-solution set P_i formed by at most Δ_i schedules that locally meet all the constraints.

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 any previously designed and implemented valid local task scheduler can be adapted to satisfy the needs of the *Local* entity in the Local-Global policy. To prove that, we decided to do exactly that: 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 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.

Hence, we needed to add extended functionalities to actually have a practical *Local* entity, which are the ones listed below:

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

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. We had to add a logic to force the solver 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 our case, we could use this built-in backtracking capabilities to find more than one scheduling solution that satisfies all the constrain.

2. **Enable and upgrade the task selection capabilities for the schedule solution.** The ^{3}Cat-1 task scheduler's procedure to select the tasks that could fit in the schedule was an iterative process that uses task priorities, something that at the moment we do not need to consider for the Local-Global policy. This iteration process generates a valid combination of tasks by gradually expanding an initial sub-problem with more tasks. It starts with the simplest sub-problem (e.g. "allocate resources to task a_1") and continuous adding subsequent tasks, backtracking in the combinations tree whenever an unfeasible problem is found.

   In our case, we do not iterate in this way but we always try to solve the problem with *all* the input tasks. However, we need some way to be 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 we find a task or a number of them that cannot be scheduled because of resource or time constraints, 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. We further define F it below.

4. **Communication with the *Global*.** The local scheduler designed for ^{3}Cat-1 simply output the found solution, but the *Local* entity must also send the set of sub-solutions to the *Global* layer. As it has been already said, the information that the *Local* must transmit is limited to the subset of tasks included in each sub-solution, and its figure of merit. 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.

Mathematically describing a scheduling solution

As it has been previously said, defining the F is a crucial step for implementing the Local-Global task scheduler, as it will provide the final *goodness* of the global solution fruit of the combination of the *Locals* provided sub-solutions.

Let all the terms present in the multiple-satellites multiple-tasks scheduler problem that is to be solved by the Local-Global policy 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.

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

$T_{\textbf{end}}$ 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.

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 account for 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:

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

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

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 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 for making a more 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:

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

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

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 Optimizing the optimization: the Global algorithm

As it has already been explained, the global scheduling algorithm is basically a combinatorial optimization problem, that can be described as in (3.8). The *Global* layer receives S sets of Δ_i sub-solutions each, described by a pair formed by the scheduled tasks set and its figure of merit. 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.

We could simply aggregate the values 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 of the global solution's quality, we have decided to weight the sum of the F values 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 $S \cdot |A|$. Below the complete definition of the global problem can be found. Note that the binary decision variables $x_{ij} = 1$ if sub-solution P_{ij} is part of the final combination and 0 otherwise:

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

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

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

We will divide the description of the *Global* entity implementation 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-based design and recursive improvements made on the algorithm, and finally in the third part the coordination with the *Local* entity is explored.

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:

$$(20103)$$

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.

In brief, it can be easily observed that the entire space of combinations would be a set of $\prod_{i=1}^{S} (\Delta_i + 1)$ combinations, which in an homogeneous case where $\Delta_i = \Delta_{\text{system}}$ for every satellite would be approximately $\Delta_{\text{system}}^{S}$.

However, this expression does not completely include the complexity of the search. To find out the optimal combination the value of $r(P)$ function must be calculated for each one, 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:

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

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:

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

– If $\Delta_{\text{system}} = \Delta_0$ and $|A|$ is kept constant:

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

– If $\Delta_{\text{system}} = \Delta_0$ and S is kept constant:

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

Hence, we are facing an exponentially exploding problem when solving the *Global* entity combinatorial optimization

An efficiency-based design of the combinatorial search

The most basic and simple resolution procedure of this problem would be a brute force approach: explore the entire space of possible combinations and simply compare the values of $r(P)$ function of each of them and select the one who maximizes it. However, when facing a problem that depends exponentially as we increase the number of satellites and the Δ value of each one, the brute force approach can be completely useless, because of the long time it takes to end the exploration.

That is why some efficiency implementation decisions have been taken and a careful study on the algorithmics and data structures that could be used has been performed. The design explained below –and its subsequent implementation– has been completely carried out during the development of this Bachelor Thesis.

First of all, it is very important that we 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 $\left[1, \dot{F}\right]$, where $\dot{F}$ is the sum of F of a particular combination.

Therefore, **if we find out the way of exploring the space of combinations in a way that $\dot{F}$ value is decreasing, we will be able to cut the exploration** as soon as we find a combination such that its $r(P)$ value is equal to its $\dot{F}$ value (and this is the optimal combination, as we will prove later) or we find a combination which $\dot{F}$ value is equal or below the maximum $r(P)$ found till now.

A brief justification of these two affirmations is the one that follows: the fact that $r(P)$ value is bounded in the interval $\left[1, \dot{F}\right]$, and that we are exploring the combinations space in decreasing $\dot{F}$ makes that no combination with lower $\dot{F}$ than any other with $r(P) = \dot{F}$ will have an $r(P)$ higher than this one because $r(P$ is never bigger than $\dot{F}$ value (first cutting context) and that no combination with lower $\dot{F}$ 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. **??** this two situations can be seen.

Having this, we must find the way of exploring the combinations space in decreasing $\dot{F}$. Let us assume that we have the S lists of Δ_i sub-solutions delivered by each satellite ordered by decreasing F. This means that combination $(11...1))$ is the one who has the highest $\dot{F}$ value. However, this does not mean that this combination is the one that has a maximum $r(P)$ value.

Hence, if we want to explore the combinations space in decreasing $\dot{F}$, we already know where to begin: by the combination $(11...1))$. The way to follow the exploration is explained below.

Let us define two key concepts for this explanation:

- 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 F. For instance, the combination $(23\mathbf{3}40))$ is the successor of $(23\mathbf{2}40))$. Observe, however, that it is also successor of $(233\mathbf{3}0))$.
- 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 $(233\mathbf{3}0))$ are predecessors of $(23\mathbf{34}0))$.

Therefore, it can be proved that we will explore in decreasing $\dot{F}$ the combinations space if we follow the following procedure: for each combination we are processing we insert its successors in an ordered list and select the first element in the list as the following combination to be processed, starting from the combination $(11...1))$.

There is only an observation that must be made: as it has been already said, 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: 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)).

For the implementation carried out in this project, this optimized design has been implemented using C language, and an efficient ordered list library has been also implemented in order to have everything in the code controlled for best performance. In chapter 4, particular results for this implementation compared with a brute force search are shown.

Coordination with the *Local* entities

To end up this section, we will devote this part 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.

As it has been already said, for our simulations 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 The price-based adaptive Task Allocator

To implemented the proposal described in [17], some implementation details which are not comprehensively specified (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.

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 state graph implemented for this algorithm can be seen in Fig. **??**.

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.

Let us talk now about describing the node's state knowledge. In this Bachelor Thesis' implementation it has been designed as a set of local variables maintained by each satellite:

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 till that particular moment, generated from the different scheduling processes that have been completed since the beginning of the scheduling window.
7. **Assignation of tasks.** 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, which is a functional programming and distributed systems oriented language. Its distributed nature has allowed us 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 –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–, but is meaningless for the scheduling algorithm.

Another specificity of our implementation has been the simplification of the *Listing Phase* (see 2.2.1). Instead of dividing complex tasks into more simple ones, we have considered and already dependence-processed input task set. 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.

3.2.2 Modifications over the original proposal

As it has been said before, for the sake of performance of our implementation of this price-based proposal, some modifications to the original algorithm have been implemented.

First of all, to better compare both Local-Global and market-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:

$$BP_{ij} = a \cdot e^{-\left(\frac{1-(l^{a_j}/E_i)^2}{b}\right)} \text{ if } E_i \geq l^{a_j} \tag{3.10}$$

Communication Cost (CC): this value has been also redefined to be bounded supposing that all satellites in the system are in the same LEO[5] 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.

$$CC = \begin{cases} 0 & \text{if no satellite assigned to } pred(a_j) \\ \dfrac{d(i,k)}{CC_{\text{max}}} & \text{if satellite } k \text{ has been assigned to } pred(a_j) \end{cases} \tag{3.11}$$

Task Deadline (TD): since the *Listing Phase* is not necessary in this implementation, we have redefined the Task Deadline 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.

$$P_{ij} = \begin{cases} CC + BP_{ij} + \dfrac{1}{DL - RT_i + 1} & \text{if } BP_{ij} \text{ defined and } DL \geq RT_i \\ 2 + a & \text{otherwise} \end{cases} \tag{3.12}$$

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 logic 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 (3.12) 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.

[5]LEO stands for Low Earth Orbit

Chapter 4

Results

4.1 Parametrizing a distributed task scheduler

In the last section all the details of both Local-Global and market-based distributed task schedulers implementations carried out in this Bachelor Thesis have been detailed. However, to be able to detect the differences in the performance of both algorithms, a benchmark of simulations had to be done.

Nevertheless, whenever is wanted to test an algorithm, its input variables and how these do affect to the time and memory spent in resolving a particular problem must be analysed.

First of all, let us 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 market-based or any other one):

- **Number of tasks.** As the number of tasks to be scheduled increase, the problems complexity increases, as we have the same resources (energy, time...) for more work. In this sense, the problem is more difficult to solve.
- **Satellites resources.** For the same number of tasks, if we decrease the resources available at the system, we will have the same problem as in the case of increasing number of tasks.
- **Number of satellites.** Even though this supposes having more resources available and hence more possibilities of finding out a solution, the increasing number of satellites causes a bigger distributed system to coordinate, which could lead to an unmanageable system full of resources but unable to schedule any task.
- **Scheduling window.** If the scheduling window time is increased, the task allocation problem's complexity could increase, as there are more timing combinations available to *test*. However, if it is set to a very low value, it could create an unsolvable problem, as the same tasks are trying to be nested in an insufficient time period.

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

- **Golden index.** This variable is of course completely particular to this algorithm as it is a value defined for it. For very high values of the golden index, more sub-solutions 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.

Finally, the market-based algorithm variable analysis lead us to highlight the following:

- **Satellites resources.** The algorithm results can be quite poor in terms of the optimality if 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 market-based has a big handicap in the "sequentialization" that it introduces in the scheduling process: each task is processed and scheduled in non-overlapping rounds, so a high number of tasks could lead to a overloaded system not able to schedule tasks at the same velocity that they arrive in the system.

Sweeping these parameters will show us 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). Moreover, this previous analysis should be confirmed by the experimental simulations shown below.

4.2 Performance Tests

After having theoretically analysed the critical variables for the generic scheduling problem and for each one of the algorithms, experimental simulations have been carried out.

In order to have a good testing platform for being able to perform extensive simulations sweeping the critical variables and executing several tests for a statistically measure, a simple tester application tester has been implemented. This application receives the set of input variables (number of satellites, number of tasks, golden index, scheduling window...) and the sweep required for them. Then it first simulates an Erlang execution with random generated tasks satisfying the input parameters and after that 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 specific Matlab script, also developed for this Bachelor Thesis. Each simulation is performed several times for being able to statistically approximate the real behaviour.

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 while the other two were swept 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]. 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 market-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 see in figures 4.4, 4.5 and 4.6 respectively.

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

Finally, the total scheduled task number (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

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

(golden index fixed). Both plots are very similar and highlight an important characteristic of the behaviour 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 them all. Then, the system begins to saturate and it is not able to schedule 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 must be paid. To quickly analyse the improvement it 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.

4.2.2 Price-based

The price-based results analysis is 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 scheduled number of tasks must be highlighted: the price-based algorithm achieves to schedule almost all the input tasks for this values. However, when we sweep 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.

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 our sight.

Regarding the time spent in solving the input tests, we can observe 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, if we look to the number of scheduled tasks when increasing the number of input tasks (see Fig. 4.20), the Local-Global policy demonstrates its strength on obtaining a near-optimal solution in terms of scheduled tasks (besides other parameters of the solution quality such as satellite utilization or responsiveness): it achieves scheduling more tasks than the price-based scheduler for almost all the tests. The sole range in which the Local-Global is advanced by the price-based is the 10 to 40 range in this case (i.e. between the first saturation point of the Local-Global and the saturation point of the price-based).

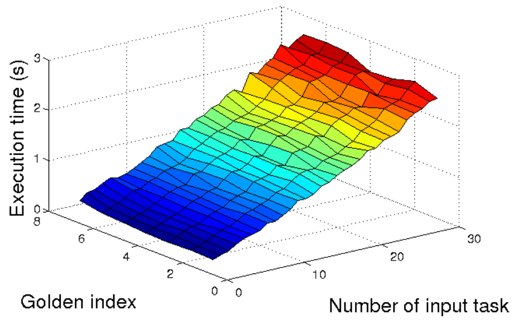

FIGURE 4.1: Execution time (satellites fixed)

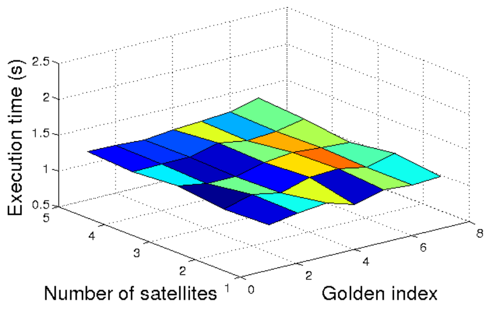

FIGURE 4.2: Execution time (input tasks fixed)

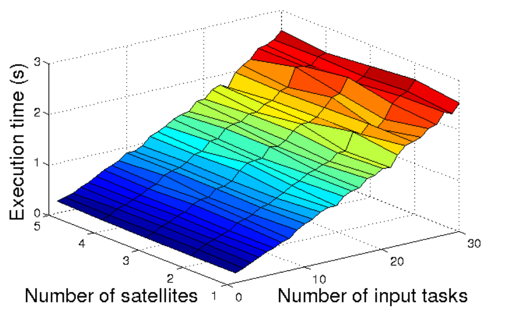

FIGURE 4.3: Execution time (golden index fixed)

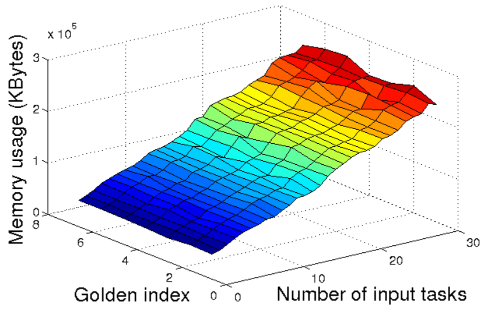

FIGURE 4.4: Memory usage (satellites fixed)

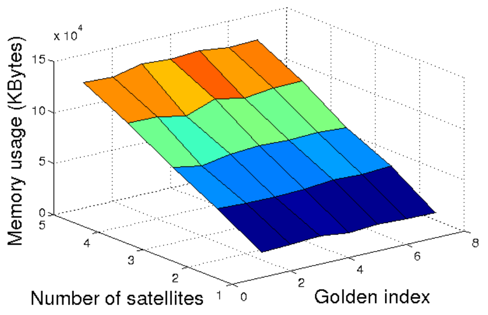

FIGURE 4.5: Memory usage (input tasks fixed)

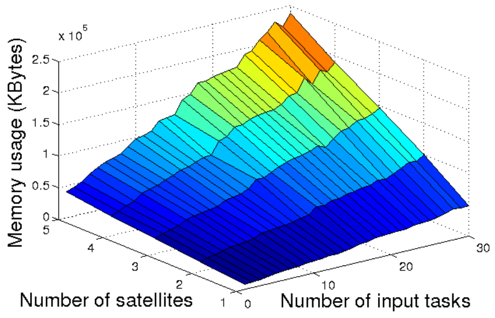

FIGURE 4.6: Memory usage (golden index fixed)

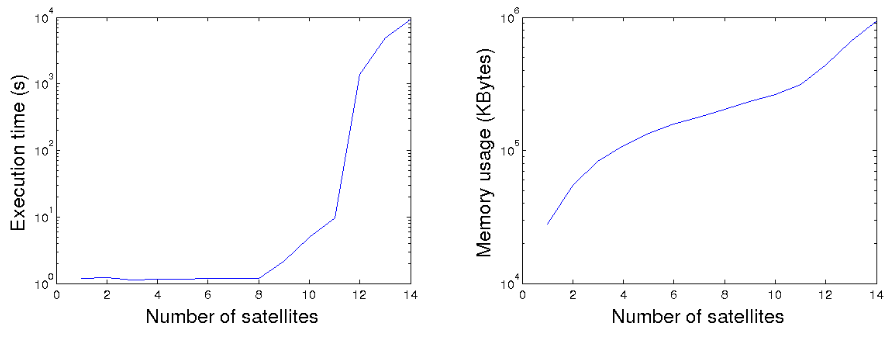

FIGURE 4.7: Execution time and memory usage (varying number of satellites)

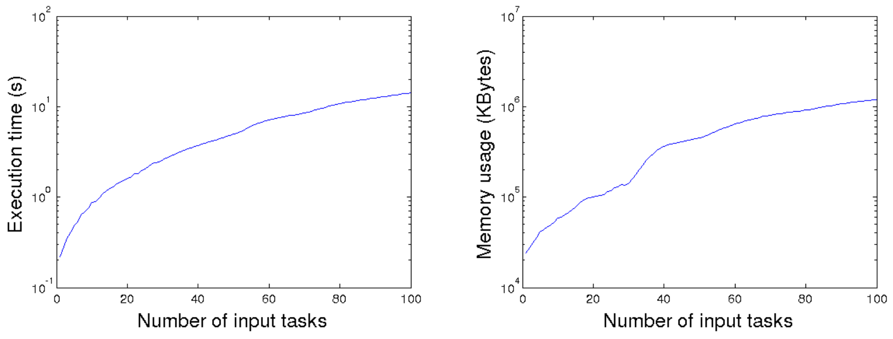

FIGURE 4.8: Execution time and memory usage (varying number of input tasks)

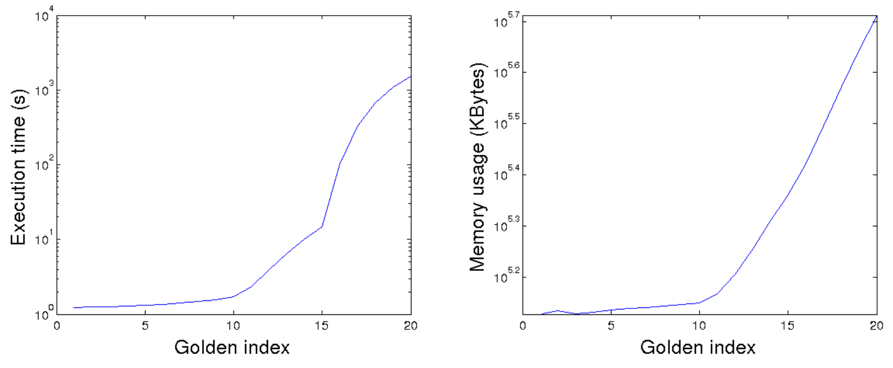

FIGURE 4.9: Execution time and memory usage (varying golden index)

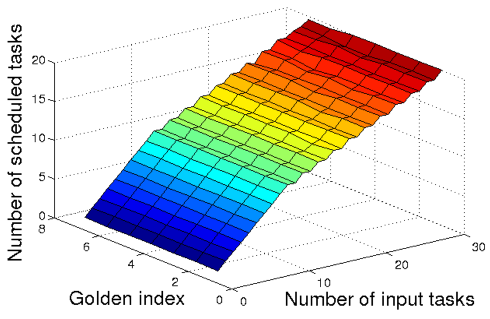

FIGURE 4.10: Execution time (Local-Global)

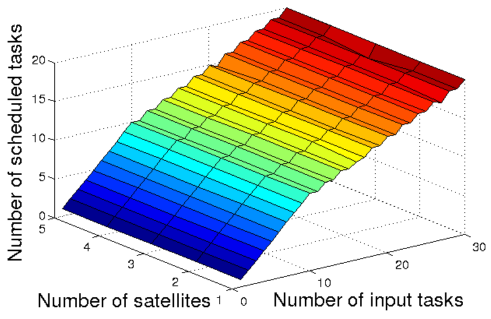

FIGURE 4.11: Scheduled tasks (Local-Global)

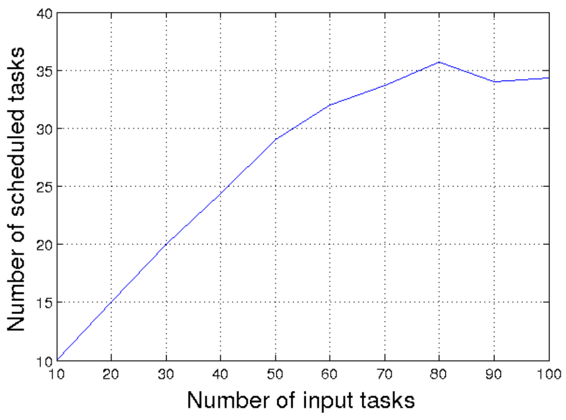

FIGURE 4.12: Large input tasks range (Local-Global)

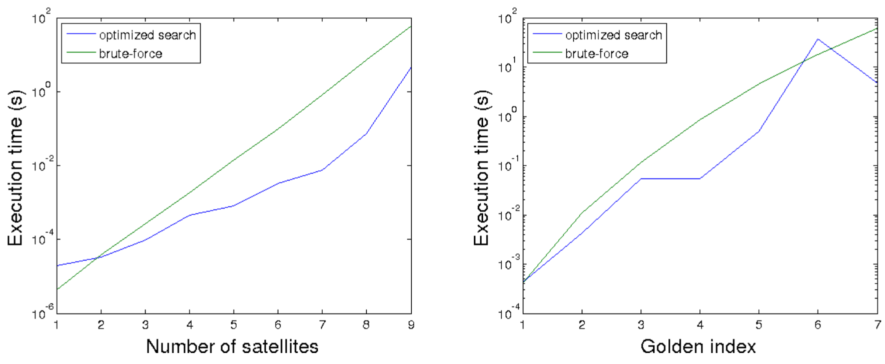

FIGURE 4.13: Execution time comparison (combinatorial search vs. brute force)

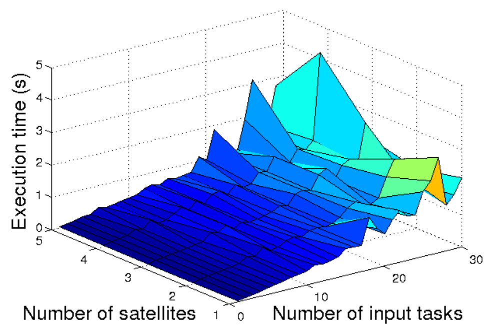

FIGURE 4.14: Execution time (price-based)

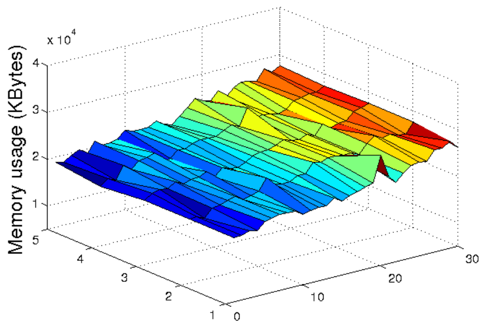

FIGURE 4.15: Memory usage (price-based)

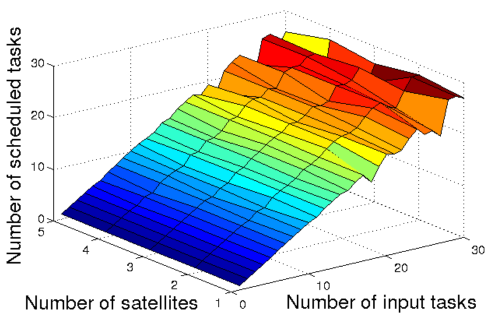

FIGURE 4.16: Scheduled tasks (price-based)

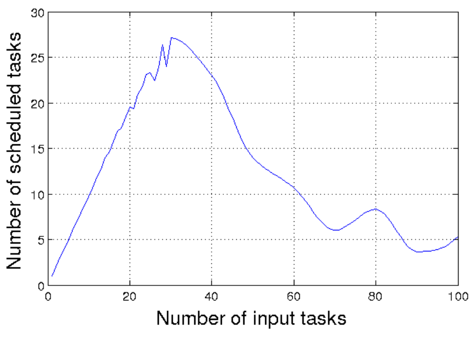

FIGURE 4.17: Large input tasks range (price-based)

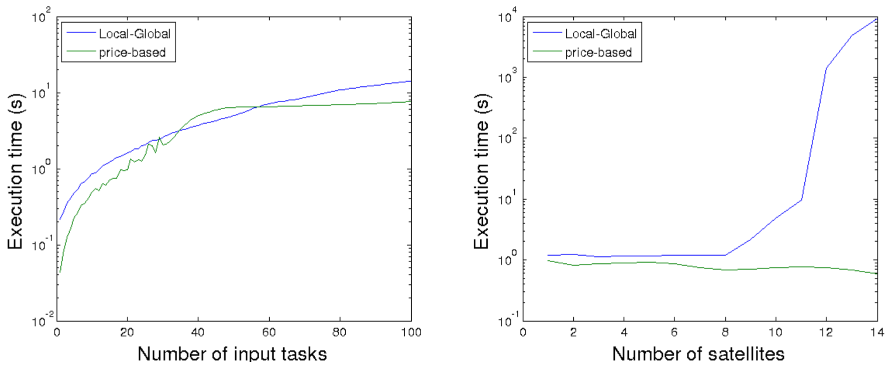

FIGURE 4.18: Execution time comparison

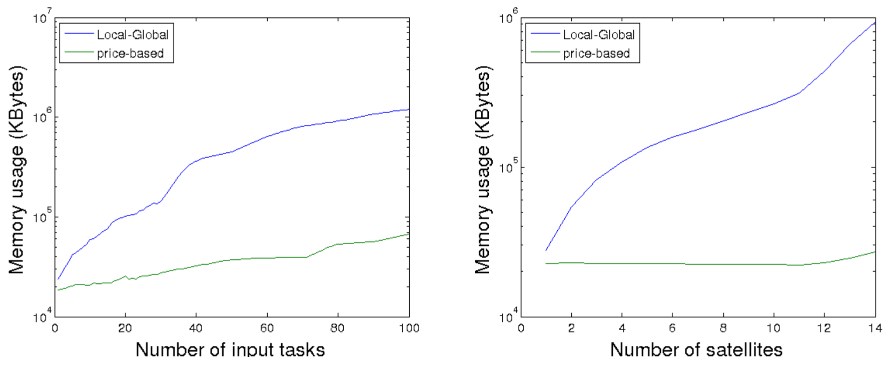

FIGURE 4.19: Memory usage comparison

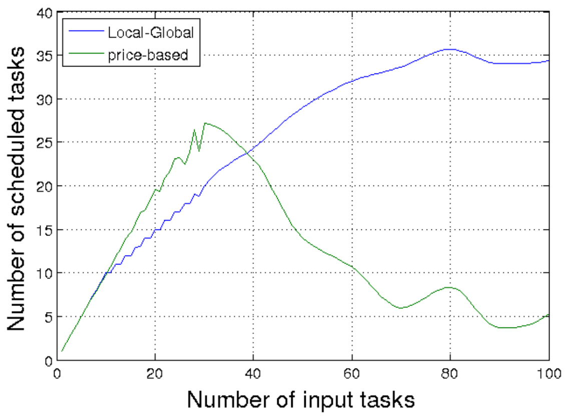

FIGURE 4.20: Scheduled tasks comparison

Chapter 5

Conclusions

5.1 Results analysis and solutions quality

In the last section the most significant results have been shown. However, some specificities of each algorithm that we may not observe in those figures must have been explained for a better comprehension of the behaviour, advantages and disadvantages of each algorithm.

Regarding the Local-Global policy, it must be said that there are many solutions quality aspects included in the schedule optimization it performs apart from how many tasks have been finally scheduled (remember 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 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 subsolutions are delivered to the *Global* from several satellites, giving no chance to find a good combination.

The price-based scheduler has also some characteristics that are not found on Local-Global: dependence between tasks and accounting the communication cost of passing the results from one task to another is included in the schedule calculation. Furthermore, the market model allows us to redefine the bid calculation for taking into account whichever parameter we want to adapt the performance to a particular system. This could be done also either statically or dynamically.

When both algorithms are compared, we find that the price-based execution times and memory usage is lower than the Local-Global's, but there are some handicaps to be observed: the price-based "sequentilaization" of the scheduling process is a bottleneck and causes a saturation point that decreases the performance for high number of input tasks, while the Local-Global's saturation is softer and it achieves to maintain the number of scheduled tasks for this situation. Price-based scheduler also requires good communication among the satellites forming the cluster, something not very realistic 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

In this work the Local-Global policy's design has been completed and fully implemented, finding out and implementing the most appropriate state-of-the-art system for comparing both and test the real Local-Global's performance. The parameters fitting most to the comparison's goals have been defined and large testing benchmarks have been performed for both algorithms, quantitatively measuring the behaviour of each one of them.

These complete work allows us to firmly state some conclusions about the Local-Global policy's future work to be done:

- As it has already been said, although the current performance can be considered as a very good result for the first approach, an improvement can be obtained by optimizing the *Local* scheduler. 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 the may contain repetitions among them.
- Also some optimizations could be included in the *Global* combinatorial search. Despite having very good performance when heterogeneous satellites form the cluster, for homogeneous cases it decreases dramatically. These are the cases to be studied for the improvements to be carried in the algorithm.
- 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.

To conclude, the results obtained in this Bachelor Thesis have allowed to characterize a theoretical sketched distributed task scheduler that could be implemented as a critical part of the autonomy system of future fractionated satellite systems.

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

[3] 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.

[4] 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.

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

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

[7] 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`.

[8] 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.

[9] 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`.

[10] 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`.

[11] 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`.

[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] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.