Escuela Politécnica Superior de Linares

Telecommunication Engineering Master

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Master Thesis

DEVELOPMENT OF A SOFT-COMPUTING BASED META-SCHEDULER FOR CLOUD COMPUTING TAKING INTO ACCOUNT THE "FOLLOW THE RENEWABLE" APPROACH

Student: Adrián Cotes Ruiz

Supervisors: Dr. Sebastián García Galán
Dr. Rocío Pérez de Prado

Department: Telecommunication Engineering Department

March 2020

Index

Figure index

Table index

RESUMEN

En este trabajo fin de máster se ha realizado un trabajo de estudio e investigación sobre los procesos de ejecución y planificación de tareas en sistemas cloud compuestos por múltiples datacenters. Se muestra como el principal objetivo de este trabajo de investigación consiste en mejorar el proceso de meta-planificación y reparto de tareas entre múltiples datacenters en un sistema cloud, así como como del proceso de planificación local dentro entre los diferentes recursos disponibles en cada datacenter con el objetivo de reducir el tiempo requerido para la ejecución de tareas y el consumo total de potencia y energía realizado por los datacenters debido a dicho proceso de ejecución de tareas, tratando de priorizar la ejecución de tareas en aquellos datacenters que cuenten o dispongan de suministro eléctrico por medio de energías renovables.

Para realizar este proceso de investigación se ha utilizado un simulador de cloud llamado *WorkflowSim with DVFS* and meta-scheduling que fue desarrollado en mi TFG. Este simulador permite una fácil creación de múltiples escenarios de cloud, permitiendo crear tantos datacenters como sean necesarios con tantos hosts por datacenters como sean requeridos, permitiendo además crear escenarios heterogéneos y homogéneos tanto entre datacenters como entre los equipos que componen cada uno de los diferentes datacenters. Además, este simulador permite la simulación de diferentes bolsas de trabajo que proporcionan mayores posibilidades de creación de diferentes escenarios de simulación. Adicionalmente, el simulador, que está programado en Java, presenta un alto nivel de flexibilidad, versatilidad y modularidad gracias a su estructura de clases que utiliza los procesos de herencia de Java, procesos que permiten extender fácilmente las capacidades de las clases ya implementadas añadiendo así nuevas funcionalidades. Gracias al uso de la herencia en el simulador es posible implementar nuevas funcionalidades con objeto de poder realizar procesos de estudio e investigación adicionales sobre la planificación y meta-planificación de tareas.

Finalmente, todo el proceso de desarrollo e implementación de las nuevas características y funcionalidades en el simulador han sido convenientemente protegidas en un repositorio de GitHub privado perteneciente al grupo de investigación TIC-188 de la universidad de Jaén, proporcionando una amplia documentación y descripción de todos los procesos llevados a cabo para la completa implementación de las nuevas características

implementadas. Este documento es concluido con todos los resultados alcanzados mediante la implementación de las nuevas funcionalidades alcanzadas tras el desarrollo de este trabajo fin de máster.

INTRODUCCIÓN

La demanda sobre los servicios cloud ha sufrido un significativo incremento en los últimos años. Actualmente, estos servicios no solo proporcionan almacenamiento remoto de datos, si no que además son utilizados para una gran cantidad de diferentes tareas. Empresas como Amazon y Microsoft todo tipo de servicios cloud por medio de sus plataformas cloud AWS [1] y Azure [2]. A través de estas plataformas se ofrece una gran variedad y amplitud de diferentes servicios como MPP online (Procesamiento Paralelo Masivo online), servicios multimedia (codificación y decodificación en tiempo real), kit de desarrollo online para compilación y depuración de código, servicios de hosting, etc. Además, otras empresas como Google incluso ofrecen servicios de juego online a través de su plataforma Stadia [3].

En [4] se especifica como la demanda sobre los servicios cloud está incrementando una media de un 12% anual, junto a un incremento al mismo tiempo del coste de la electricidad en un 4,4% anual. Además, en [5] es especificado que el consumo de potencia realizado por un datacenter es equivalente al realizado por 25000 viviendas. Este hecho refleja que el consumo de potencia realizado por los servicios cloud no debe ser ignorado, situación que se espera que empeore en los próximos años. Debido a esto hay una urgente necesidad de actuar en consecuencia de acuerdo con el problema descrito para alcanzar una solución al continuo incremento del consumo de potencia realizado por los servicios cloud y los datacenters.

CONCLUSIONES

Esta sección resume todo el trabajo realizado en este TFM, especificando los objetivos y metas alcanzados, así como los resultados obtenidos.

Para comenzar, uno de los principales objetivos consistía en extender las capacidades del simulador WorkflowSim with DVFS and meta-scheduling creado en mi TFG, así como realizar una limpieza de código en profundidad, objetivos que han sido realizados satisfactoriamente. Primero, como se muestra en la *sección 5.1* se ha realizado grandes limpiezas de código, limpieza que ha sido posible realizar gracias a la herencia de Java, ya que ha permitido crear de nuevo las entidades pertenecientes al escenario de *meta-planificador*, haciendo que estas entidades hereden de las ya existentes en el simulador *WorkflowSim with DVFS,* permitiendo así reutilizar gran parte del código previamente ya implementando solo requiriendo pequeñas extensiones de código para implementar las entidades del simulador con meta-planificación. De igual manera, los algoritmos de meta-planificación previamente implementados han sido modificados para eliminar parcialmente el uso de una clase java estática para el intercambio de información, facilitando así su uso y compresión. Segundo, una gran variedad de mejoras ha sido implementadas en el simulador como es especificado en la *sección 5.2*, entre las que se encuentran la implementación del método de planificación on-line, la extensión de las capacidades de los algoritmos FRBS previamente implementados para permitir el uso de términos negados y no negados así como los conectores AND y OR, la extensión de funcionalidades del simulador por medio de la implementación de los modelos de energías renovables, un nuevo algoritmo de meta-planificación FRBS que utiliza las nuevas capacidades FRBS implementadas previamente especificadas, la implementación de los sistemas de adquisición de conocimiento y la planificación multiobjetivo.

En la *sección de resultados* es mostrado como se han alcanzado importantes metas y objetivos. Primero, se muestra la efectividad de los sistemas de adquisición de conocimiento como Kasia y Pittsburgh para la fácil obtención y creación de bases de reglas óptimas que usadas sobre el nuevo y avanzado algoritmo de meta-planificación basado en FRBS permite obtener resultados que presentan grandes mejoras respecto a los algoritmos clásicos de meta-planificación, tal y como es mostrado en la *sección 6.2.1*.

Gracias a la inclusión e implementación de las capacidades extendidas de FRBS por medio de los términos negados y los conectores AND y OR ha sido posible alcanzar una significativa reducción en el número de reglas de las bases de reglas generadas por medio de los sistemas de adquisición de conocimiento. Como se especifica en la *sección 6.2.2*, esto permite un proceso de meta-planificación más rápido al disponer de menos reglas que evaluar en el proceso de meta-planificación, reduciendo así los requerimiento computacionales en el proceso de evaluación de los diferentes antecedentes por medio de la base de reglas. Además, esta reducción de reglas permitirá una reducción de los consumos de potencia y energía en la entidad meta-planificadora al requerir de menos tiempo en el proceso de meta-planificación de tareas. Adicionalmente, cabe destacar que la reducción del número de reglas permite conseguir un mayor nivel de interpretabilidad, lo que facilita significativamente la comprensión de la lógica y estrategia establecida en la solución propuesta por el sistema de adquisición de conocimiento.

Por otra parte, la creación de los escenarios de simulación de energía renovable proporciona al simulador nuevas capacidades, capacidad que ha sido implementada en dos métodos diferentes, uno, por medio de la inclusión de una batería en cada datacenter que almacena solo y exclusivamente energía renovable y otro, que supone que la fuente ilimitada de energía que alimenta cada datacenter dispone de una determinada proporción de energía renovable. A pesar de ser una primera y simple implementación y aproximación, la inclusión de esta nueva característica permite el estudio técnicas de meta-planificación más complejas por medio de Pittsburgh y Kasia con objeto de preparar estos algoritmos frente a los datacenters del futuro que utilizan energías renovables, permitiendo así priorizar la ejecución de tareas en aquellos datacenters que dispongan de alimentación por medio de energías renovables. Además, para facilitar el estudio de las energías renovables, estas han sido implementadas en diferentes modelos de energía renovable, creando un modelo base sobre el que los diferentes modelos heredarán características básicas para así extender sus capacidades y permitir diferentes escenarios de simulación de energías renovables, habiendo sido implementado hasta este momento 4 modelos de energías renovables.

Finalmente, con objeto de proporcionar una correcta y real implementación de la optimización multiobjetivo se ha implementado la *eficiencia de Pareto* sobre el algoritmo Pittsburgh. Tal y como es mostrado en la *sección 6.2.3*, esta nueva implementación muestra un gran potencial, mostrando como es capaz de proporcionar múltiples soluciones óptimas de manera simultánea para solventar los problemas de optimización en cuestión. Sin

embargo, tal y como se muestra ciertos resultados de los obtenidos por medio de la implementación actual son inconsistentes, mostrando como en ciertos escenarios es capaz de proporcionar resultados excepcionales mientras que, en otros escenarios, los resultados que han sido obtenidos han sido nefastos comparados con los proporcionados con otros algoritmos. A pesar de esto, los resultados proporcionados por la *eficiencia de Pareto* son prometedores, siendo de interés realizar futuras investigaciones sobre este nuevo algoritmo para tratar de explotar las capacidades y posibles beneficios que puedan ser obtenidos.

1. ABSTRACT

In this master thesis a work of research with respect the task execution within Cloud systems made of multiple datacenters is done. Is exposed that the main objective of this research is to enhance the way to schedule tasks involving the multiple datacenters at a Cloud system as well as tasks within the host/computers within a specific datacenter with the purpose of reducing both, the total execution time of tasks and the power consumption, trying to prioritize the execution of tasks to those datacenters that have renewable energy availability.

To do this, a simulator called WorkflowSim with DVFS and meta-scheduling is going to be used, which was developed in my previous final project degree. This simulator allows the creation of different Cloud scenarios made of as many datacenters as desired, both homogeneous and heterogeneous among datacenter and/or with the host that composed each datacenter. Additionally, it also allows the simulation of different bag of tasks to adapt it to the Cloud hardware specified scenario. Moreover, it presents a modular scenario based on classes, which provides a high degree of adaptability and versatility that allows the implementation/extension of new simulator functionalities. As a consequence of that, new tasks scheduling strategies considering these new capabilities are possible.

Finally, all the development/programming done to implement new functionalities to this simulator that allows the specified research process is securely uploaded to a GitHub private repository property of the TIC-188 research group, wide and in-depth documenting all the steps done for the implementation of the new functionalities, allowing newcomers to the project to easily understand how all the additional functionalities have been implemented. The document is concluded with all the achieved results from the implementation of the new achived functionalities through the development of this master thesis.

2. INTRODUCTION

In recent years, the demand for cloud services has risen significantly. Nowadays, these services not only provide data storage, but also are used for an incredibly high number of different tasks. Enterprises as Amazon and Microsoft offers all kind of different cloud services through their AWS [1] and Azure [2] services. The offered services vary from a wide amplitude of different services offering online MPP (Massive Parallel Processing), multimedia services (real-time media coding and decoding) cloud development kit for code compiling and debugging, hosting, etc. Moreover, Google is now offering gaming services through their cloud by means of Stadia [3].

In [4] is specified that the cloud service demand is increasing an annual amount of 12% on average, together with an average increase of the electricity cost of 4.4% yearly in the U.S. Moreover, in [5] is said that a data center power consumption is equivalent to up to 25000 households. These facts reflect that the cloud services power consumption must not be ignored, situation that is only expected to become worse through the years. Due to this, there is an important necessity to act accordingly to put an urgent solution to this increasing problem in order to reduce the power consumption developed by the cloud services and data centers.

3. OBJECTIVES

The whole master thesis degree development is aimed to the research of different ways to reduce the power consumption of data centers. For this purpose, a wide study/research about meta scheduling algorithms are going to be done to enhance how tasks are scheduled among different data centers so that tasks are delivered to the most appropriate data center to execute them.

To do this, the use of a previously developed cloud simulator called WorkflowSim with DVFS and meta scheduling is going to be used, avoiding the necessity of building complex and expensive physical scenarios to carry the meta scheduling algorithms/strategies research. The simulator must be correctly upgraded to allow the study of more advanced meta scheduling strategies.

Among the different aspects for which the simulator must be upgraded, it must be added the possibility to simulate basic renewable energy scenarios, made of a battery that store renewable energy as well as a direct power source made of a partial renewable energy proportion.

Related to the meta scheduling algorithms, its enhancement will be based in different aspects. To achieve advanced meta scheduling, the Fuzzy Logic (FL)/Fuzzy Rules Based System (FRBS) is going to be used as it allows the management of multiple variable to be considered for the scheduling process. To obtain the best result possible from the FRBS meta scheduling algorithms, knowledge acquisition system such as Pittsburgh [12] and Kasia [10] are going to be used for finding optimal Rule Base (RB) to achieve better results. Additionally, as the optimization process will imply the consideration of multiple variable, the multi-objective optimization is going to be enhanced through the Pareto efficiency.

Finally, last but not least, the reduction of complexity by mean of massive code clean up has also been carried out.

4. ANTECEDENTS

This section is aimed to introduce some basic concepts and knowledge that will be necessary for the correct understanding of this master thesis degree.

4.1. Cloud Computing types

A wide variety of different cloud computing services can be found. These cloud types are usually defined as XaaS (X as a Service). Depending on the cloud service the final user will have a different type of permissions. The higher the permissions the final user/administrator have, the higher the complexity will be and, at the same time, the more knowledge that will need to manage the whole system. The different type of cloud services is shown below:

- Application level:

Based on *SaaS* (Software as a Service), this type of cloud provides an application or group of applications as a service. These services are remotely executed in a remote cloud service rather than on a physical computer or server within the organization. As its main advantage, the service provider is responsible for the hardware and software maintenance as well as its correct behavior of the whole system, for which the final user or organization will pay a specific subscription or fee for its use.

- Platform level:

Based on *PaaS* (Platform as a Service), this type of cloud gives more privilege/control to the final user/organization, giving control over the *OS* (Operating System). As the final user has control over the *OS,* advanced user will be able to add new services to the contracted cloud service, although the hardware/server maintenance responsibilities will remain in charge of the service provider. Even though the final user has control over the *OS*, it cannot decide the destination server in which the provided service/OS that execute over a *VM* (Virtual Machine) must execute within the whole service provider *data center*.

- Infrastructure level:

Based on *IaaS* (Infrastructure as a Service), this type of cloud gives more privilege to the final user than the given in the *PaaS*. In this type, the user/organization that contract the service can choose the physical machine in where the *VM* or *VMs* that host its service must be execute within the service provider *data center*. Moreover, provide the ability to control the amount of physical resources that can be allocated to the *VMs*. It requires more advanced knowledge as it grant more privilege/control over the final user that contract the service.

- Hardware level:

Based on *HaaS* (Hardware as a Service), this type of cloud gives almost total control and responsibility to the final user. The service provider is going to provide to the end user/organization with all the necessary hardware and is the final user the one in charge of installing it on its own infrastructure, leaving only in charge of the provider the hardware maintenance. With *HaaS*, the final user has the highest level of control having total control of it and maximum flexibility.

- Public:

Cloud made of multiple resources from multiple sources that can be enterprises, home users, or any other kind of resource that voluntarily donate its computational resources. Its main benefit is the free-of-charge use, although its lacks security, its availability is not guarantee. Its disadvantages are given due to being third party resources provided for free, and so its connections may lack security and its availability may not be guarantee due to internet or power disconnection without a previous notice.

- Private:

Data centers made by the own organization that due to its specific necessities include the need of maximum security, guaranteed availability and maximized performance system throughput, scenario that can be achieved thanks to being specifically designed by the own organization on the need of the specific *data center*.

As its main disadvantage, the organization that creates this private *data center* oversees its maintenance as well as its modernization process or expansion in case of necessity.

- Hybrid:

Type of cloud that aims to provide the advantages of the public and private cloud systems. Its private part provides security, guaranteed availability and performance, and is aimed to be used to specific problems/applications that require maximum security. On the other hand, the public part provides an extra performance that can be used in case of necessity of more computational performance for those applications for which the security is not a necessity and additional performance must be a priority.

At the same time, it includes the disadvantages of both types, being necessary to ensure a proper maintenance for the private part and not having neither guaranteed security nor availability for the public part.

4.2. Cloud planning

The effectiveness, efficiency and efficacy of a cloud system is given by the cloud planning. The *cloud planning* defines the strategy to follow by a cloud system to deliver tasks among the different computational resources available in the whole cloud with the objective of achieving the most efficient execution for its soon completion. The *planning* process represents a key part for the cloud system having a high complexity. A bad cloud planning can lead to slow and inefficient tasks execution within the system.

The *cloud planning* is mainly divided into *planning strategies* and *scheduling algorithms*, although some basics components and concepts must be known for its correct understanding.

Basic concepts:

- Task: it is the minimum and most basic element in a cloud system that must be executed and cannot be divided into different *subtasks*. Each *task* may require different computational resources and time for its completion
- Job: refers to a complex *task* that is made of a combination of *tasks* which can be either homogeneous or heterogeneous and so require different computational resources for its completion. For a full *job* completion, all the *tasks* that made it must be completed.
- Application: a specific application with the purpose of executing a specific *task*.
- Resource: a computational entity with the ability of executing *tasks*. Its computational capabilities depend entirely on the *CPU* (Central Processor Unit),

amount of *RAM* (Random Access Memory) as well as other elements as the *OS* that made up the resource.

- Scheduler: cloud entity that stores *tasks* and *jobs* which will made use of the *scheduling algorithms* to decide which task must be executed by a specific *cloud resource*.

The planning stage followed up by the cloud system to deliver task to resources is as follows:

- First stage: the scheduler entity must gather information regarding each *task* or *job* that can be executed on the cloud system as well as the available computational *resources* and its characteristics
- Second stage: the *tasks* and *resources* are exhaustively analyzed to identify the best or optimal *resource* that must execute each *task* according to the *scheduling algorithm* implemented.
- Third stage: once each *task* has an assigned *resource*, it must be delivered to it.
- Fourth stage: *resources* start the *tasks* execution and the *scheduler* waits for the retrieval of the successfully executed *tasks*.

4.2.1. Planning strategies

There is a wide variety of different planning strategies that cloud systems can follow, although only the most important are going to be explained:

4.3.1.1. Static-dynamic

Static: corresponds to a single type where tasks are manually assigned to each resource. Its basic behavior allows to predict certain parameters as the required execution time, power consumption and energy beforehand. On the other hand, it does not have any adaptability to errors or changes on the infrastructure. If a *task* has been assigned to a *resource* that has failed, the execution will fail its execution.

Dynamic: unlike the *static* one, this type allows to adapt dynamically to the change on the cloud infrastructure. *Tasks* will not be assigned beforehand but will be assigned through a specific *scheduling algorithm* in use. If a *resource* is not available for *tasks* execution, the *task* will be schedule to a different available *resource*.

4.3.1.2. Local-global

Local: local cloud type only considers the state of the local *resources* inside the *data center* to make the *task* scheduling.

Global: the global type is found in more complex cloud scenarios. These scenarios are made of a group of *data centers* which their respective local *schedulers* for local *task* scheduling. Additionally, an upper level entity on this type of cloud infrastructure is present to control how the *tasks* are divided among the different *data centers*. This upper level entity is called *meta scheduler*, and it gather knowledge enough to be able to perform the *task* scheduling among the different *data centers* present in the scenario.

4.3.1.3. On-line – on-batch

On-line: in this type of cloud, a *task* is schedule to be executed to an available *resource* as soon as it arrives to the *scheduler*. If the *task* cadence of arrival to the *scheduler* is low, it provides a better performance than the *on-batch* type.

On-batch: type of cloud that, rather than *scheduling tasks* as soon as they arrive, it waits for a period or for several *tasks* to arrive before *scheduling* them. This process is aimed to create a *BoT (Bag of Tasks)*, which allows a better *scheduling* process as avoid assigning prematurely a specific *task* to a powerful *resource* that may be found later busy when its really needed for the execution of a demanding *task*.

4.3.1.4. Single/Multiple criteria

Tasks can be scheduled by a *single or multiple criteria*. The *single* one considers one single aspects for which a *task* should be *schedule* to one *resource* or another, as the execution time, the power consumption, the energy, the *resource* CPU load, or any other. On the other hand, the *multiple* one considers multiple parameters for the *scheduling* process.

On the one hand, a *multiple* criterion consideration can lead to significant better results, but its complexity can produce the opposite desired result, where worse result can be obtained. It is due to the generally opposite criterion for the optimization of two or more parameters, where optimizing one can lead to the worsening of one or all the rest parameters. For this reason, a *single* criterion must be also considered as its optimization is significantly easier providing results not so much worser than the one obtained in a complex to optimize *multiple criterion* scenario.

4.3.1.5. Distributed/centralized

In a *distributed* scenario the *scheduling* process is developed by multiple *schedulers* aiming a higher failure tolerance and a better performance when a high *tasks* arrival occurs. On the other hand, its complexity is significantly higher than the *centralized* one, which correct behavior/optimization is significantly easier as there is no synchronization among multiple *schedulers*. Most *meta-scheduler* cloud scenario use the *centralized* planning strategy.

4.3.1.6. Workflow oriented

A *workflow* defines a specific *task* execution dependency scenario, where some *tasks* will have a strict dependency over the execution result of another *task*, which limit the capacity of free order *tasks* execution. *Workflows* are defined by *DAG* (*Directed Acyclic Graph*), which establish the *task* execution order dependency. In *figure 3* is shown an example of *DAG*.

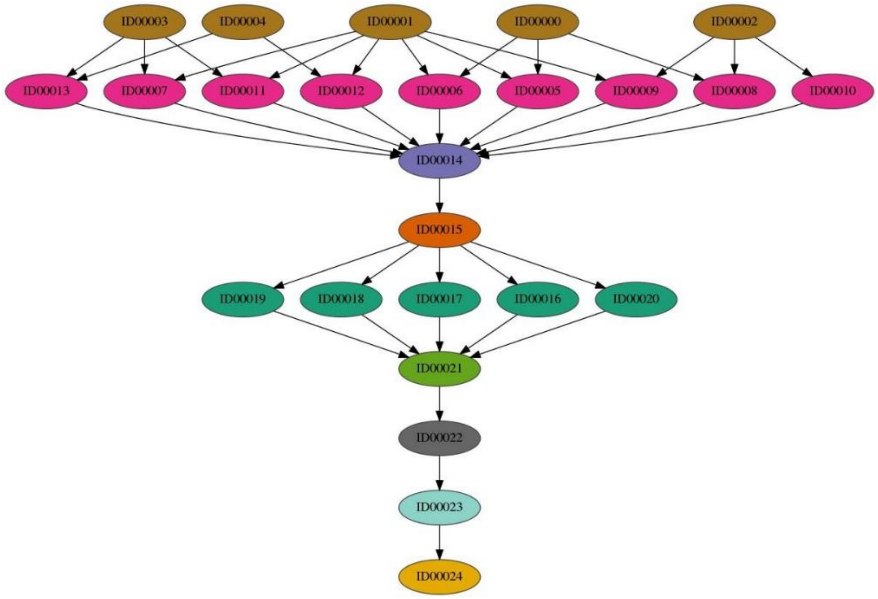

Figure 1: DAG

4.2.2. Scheduling algorithms

The *scheduling algorithms* [9] specify the strategy to follow in the process of *task* scheduling/delivering among the different available resources in the cloud system. These algorithms allow to implement certain level of intelligence for the *scheduling* process, although it is not a mandatory characteristic of these *algorithms*. Some algorithms may consider certain parameters to be optimized for the *task scheduling*, while others may follow a simple algorithm that will not implement or prioritize any type of *task* delivery among the different available *resources*.

Speaking of intelligence on the *scheduling process*, *algorithms* are divided into two groups, those that provide intelligence, which implement the *on-batch* planning method, and those that does not implement any kind of intelligence that use the *on-line* planning method.

Moreover, there is a wide variety of *algorithms*, from simple to complex, although only the most common are going to be explain here.

4.2.2.1. On-line scheduling algorithms

FCFS: the *FCFS algorithm (First Come First Served)* consist in a simple algorithm that add tasks to the execution task queue as soon as tasks arrives. The first task to arrive, will be placed on the first queue position, the second arriving, will be placed on the second queue position, and so on.

RR: *RR (Round Robin)* algorithm schedule task in a Round Robin strategy following a *FIFO (First In First Out)* strategy. As soon as a task arrives, it will be scheduled to be executed to the first computational *resource* as soon as it is not busy. In the case where the *resource* for which the task will be scheduled is busy, the *scheduler* will check if the next *resource* is in an idle state and will continue this process until an idle *resource* is found.

4.2.2.2. On-batch scheduling algorithms

MinMin: algorithm that prioritize the execution of smaller tasks on the most power available *resource*, in other words, aims to execute the smallest *tasks* in the lest time possible. To execute a *task*, a certain number of computational operations will be done, which are usually measured in *millions of instructions*. A *resource* computational power will be measured in the *MIPS* that is able to execute. In this way, to compute the time required to execute a *task*, it will be necessary to just divide the required *task millions of instructions* by the *resource MIPS*, which is going to provide an execution time required for its completion.

MaxMin: opposed to the *MinMin algorithm*, the *MaxMin algorithm* aims to guarantee that the greatest *tasks* are going to be executed in the least time possible, prioritizing its execution over smaller *tasks*.

It must be pointed that as these algorithms work over the *on-batch planning*, *tasks* will not be *schedule* as soon as they arrive to the *scheduler*, but will have to wait a certain for a certain period or until a certain number of *tasks* is reached before the *scheduler* initiates the *scheduling* process. Thanks to this behavior, multiple *tasks* will be analyzed simultaneously

to, depending on the algorithm used, order them in ascending or descending order and start analyzing the most optimal *resource* to execute each of the *tasks*.

4.3. Fuzzy Logic

Used in Fuzzy Rule Based System (FRBS) [6], the Fuzzy Logic (FL) introduce human-like logic to computing systems. While in the classic computing logic the output of a condition will either be true or false, FL allows a wide variety of numerical outputs within a specified numerical range depending on the defined FRBS in used. Not only that, but it also breaks the classic computing logic where the input condition must fulfill a strict condition, allowing the use of input condition where this strict condition is no longer considered.

Due to this new FL, both input and output variables can be classified into different groups called Membership Functions (MF) that, as previously specified, breaks the strict conditions previously specified by the classic computing logic. Now, an input or output variable can belong to more than one MF at the same time.

To try an ease the understanding of the FL and the MF, an example is provided bellow. Supposing a person classification system by their ages, where people are divided by three different groups, being these groups *young*, *adult* and *elderly*.

Classification	**Age**
Young	$< 30\ years$
Adult	$\geq 30\ years\ and \leq 60\ years$
Elderly	$> 60\ years$

Table 1: age classic membership function

In *table 1* it is shown a basic classification system based on the classic computing logic. According to this logic, a person will belong to one single group, which will lead to abrupt changes in the group in which a person is classified according to its age. As an example, supposing a person with an age of 29, will be classified as *young*, and once this person turns 30, it will be strictly classified as *adult*. Following human logic, it is seen that a person with

an age of 29 could be considered as *young* but also *adult* in some degree, and in the same time, once a person turns 30, will not necessarily be considered as *adult*, but still be considered as *young* in some degree.

Thanks to the FL/FRBS, this problem previously specified is solved. To solve it, the FRBS make use of the MF. The MFs can be made of different shapes, and according to its shape, the degree of membership for a value for a MF will vary. Depending on the FRBS system used to implement the FL, the available shapes to be used can vary, where it can be found triangular, trapezoidal, gaussians or defined point by point.

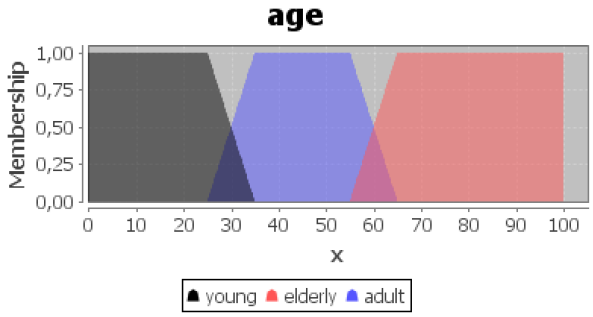

Figure 2: age membership function

In *figure 1*, a membership function example generated through jFuzzyLogic [7] [8] is shown, which is a FL library for java systems. This figure, denots how each group *young*, *adult*, and *elderly*, are represented by its own MF, where it can be shown how each MF, at certain point, start to fall its belonging degree slowly until its reach a 0 value. In the same way, the belonging degree of MF does not rise from 0 to its maximum value abruptly but experiment a slow and soft rise. This behavior allows any age input value to belong to more than MF at the same time, as specified in the example above.

Additionally, in this example the input supports values from 0 to 100, which is done on purpose to ease its understanding. However, in real scenarios to ease its implementation the input values are normalized to a range between 0 and 1, and so the MF range must be adapted in the same way to fit the total input range of 0 to 1. In the same way, some applications may require normalized input and/or output values from -1 to 1. A problem in which the output value provided must allow the increasing or decreasing of, for example, a vehicle speed depending of some input parameters, the output value can be ranged from -1 to 1 depending of it is desired to accelerate or brake the vehicle.

It's important to point that in FRBS, the input variables are given the name of antecedents, while the output variables are given the name of consequents. There can be as many

antecedents and consequents in the FRBS scenario as necessary, although the most common scenario is multiple antecedents and one single consequent. The antecedents or input variables comprise the input data of the FRBS system and are going to be considered to obtain the result of the FRBS, result that is provided by a single or multiple consequent. The consequent or consequents result are always computed by using the antecedents, for which a series of conditions are going to be used for the process. These conditions are called Rules, which make up the Rule Base (RB), which is a database that store knowledge to be used to provide information about how to deal with the antecedents to obtain the consequent/s results.

Once the basic FRBS concepts have been properly explained, it is necessary to specify the whole FRBS structure. This structure is made of 4 main components, which are the *fuzzifier, defuzzifier, inference engine and knowledge base*. At the same time, the *knowledge base* is made of the *database* and the *Rule Base (RB)*.

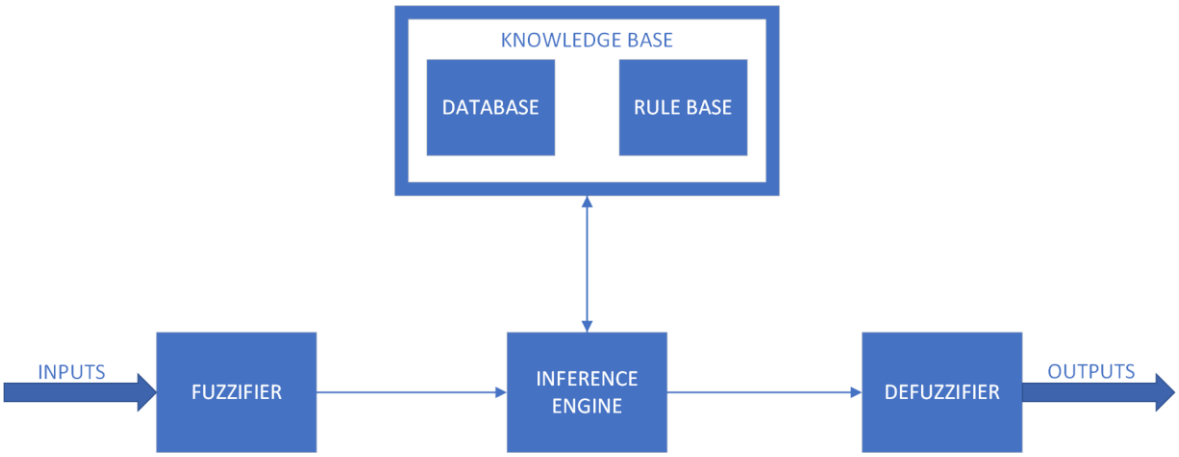

Figure 3: FRBS structure

In *figure 2*, the whole FRBS structure is shown. All its components develop a specific function and works together to provide one or multiple output/consequents from the inputs/antecedents.

- Fuzzifier:

For a normalized input, determine the level of membership for all its MF for which this input/antecedent belongs. Note that one single antecedent may have some level of membership to more than one MF. For example, using the *age* FRBS system above, an age of 26, it would determine a high level of membership for the *young* MF while determining a low level of membership for the *adult* MF.

- Knowledge Base:

The *knowledge base (KB)* provides the required knowledge to establish the relationship between the antecedents with the consequents. This *KB* is used by the *inference engine* to reach a solution for a specific problem.

The *FRBS* are found in two different types, the MISO (Multiple Input Single Output) and the MIMO (Multiple Input Multiple Output), although *MISO* is the most common scenario.

At the same time, the *KB* is made of two additional components, called *Database* and the *Rule Base (RB)*.

The *Database* specify the *MF* for each of the antecedents and consequents of the system, specifying the range and shape of each *MF*.

On the other hand, the *Rule Base* define the rules that are used to obtain the outputs. These rules establish the conditions among the different *antecedents* and the *consequents* to obtain the problem output or result. The *rules* follow an *IF THEN* structure to interconnect all the desired *antecedents* with the *consequents*. A *rule* may be made of one or all the *antecedents* of the system, not being necessary to include all the antecedents in a single rule. Conditions within a *rule* can be negated or not. If a *rule* is not negated, its condition will be set as *IS*, whereas *IS NOT* is used for negated conditions/terms. If a *rule* is made of more than one *antecedents*, these *antecedents* are connected through *connectors*. These connectors are divided into two types, which are *AND* and *OR* connectors. Connectors define the way the *consequent* value is computed. If *AND* connector is used, the lowest value of all the *antecedents* is going to be used to compute the *consequent* result, whereas the highest value is used when *OR* connector is used. Below, a few *rules* examples are provided.

- Non-negated terms, *AND connector*

```
IF X1 IS A11 AND X2 IS A22 AND X3 IS A33 THEN Y1 IS B1
```

- Non-negated terms, *OR connector*

```
IF X1 IS A11 OR X2 IS A22 OR X3 IS A33 THEN Y1 IS B1
```

- Negated terms, *AND connector*

```
IF X1 IS NOT A11 AND X2 IS NOT A22 AND X3 IS NOT A33 THEN Y1 IS NOT
B1
```

- Negated terms, *OR connector*

```
IF X1 IS NOT A11 OR X2 IS NOT A22 OR X3 IS NOT A33 THEN Y1 IS NOT B1
```

Where X1, X2, X3 are the inputs variable, and A11, A22 and A33 are the *antecedents* MF to which the input/*antecedent* is compared. Y1 represents the output variable, while B1 is the output MF to the specified *consequent*.

Note that a *rule* does not necessarily have to include all of its terms and *consequent* negated or not, and so it can include some of them negated or not negated.

- Inference Engine:

The *inference engine* is in charge of performing the antecedents evaluation for obtaining an output value from the evaluation of all the defined *rules* in the *Knowledge Base*. For this purpose, the *inference engine* makes use of the information stored within the *Knowledge Base*.

- Defuzzifier:

For all the outputs provided by the *inference engine*, the *defuzzifier* is going to, according to the MF defined at the consequent or consequents, provide a specific understandable output or outputs to the system that must be controlled through the FRBS.

To ease the understanding of the whole *FRBS* and *FL*, an example is going to be provided. It is desired to create an automated system for car/vehicle movement. The system must move the vehicle from one point to another point going straight forward without any kind of bend. For this, it must be considered the vehicle distance from the desired end point as well as its current speed. The automated system must increase or decrease the vehicle speed according to its current distance and speed. For the resolution of this problem, a *FRBS* is going to be used.

The first step consists of finding the number of required *antecedents* as well as *consequents*. It is known that two mandatory variables for the system are the vehicle *distance* and *speed* which are the system inputs or *antecedents*. On the other hand, the *FRBS* must provide an output from these two *antecedents*. It is known that the *speed* must be modified along time, which means that the vehicle must have the capacity to accelerate and brake. To create the *consequent*, it must be understood that an acceleration can be either positive or

negative, representing the positive acceleration the increase of speed whereas the negative represents the vehicle brakes. Taking into account the above-mentioned ideas, a single *consequent* can be used to represent the acceleration and braking of the vehicle.

The second step consist of analyzing the *antecedents* and *consequents* normalization process/range. It is important to appropriately specify the correct values range to be used for each normalized variable to be used in the system. For the *antecedents*, the *distance* one is known to be only positive, the more the vehicle approach to its destination, the highest the value will be, and so, the normalized *distance* value range will be between 0 and 1. For the speed/velocity antecedent the same scenario is found, the higher the speed is the faster the vehicle is going to approach to its destination, and as the car will not move backward, the speed will be always positive or 0, and so, the normalized range will be between 0 to 1. On the other hand, the acceleration consequent is a different a special scenario. As it must allow to accelerate and brake, the braking process can be represented as a negative acceleration, and so, the normalized consequent range will be from -1 to 1.

The final process, once the number of antecedents and consequents is decided, it is necessary to define the number of MF for each antecedent and consequent. Usually, the consequent must have a higher resolution, which consists on having more MF than the antecedents. For the given problem where there is not a high number of antecedents and consequents, a total of 3 MF for each antecedent and 5 MF for the consequent should provide good results.

The distance MF will be *close, intermediate* and *far*, while the speed MF will be *slow, normal* and *fast*. For the consequent, the acceleration MF will be *brake, soft brake, keep speed, soft accelerate* and *accelerate*. In *table 2* the respective MFs to each antecedent and consequent are shown.

Antecedents		**Consequent**
Distance	**Speed**	**Acceleration**
Close	Slow	Brake
Intermediate	Normal	Soft brake
Far	Fast	Keep speed
		Soft accelerate
		Accelerate

Table 2: antecedents and consequent MFs

Once the whole FRBS has been defined, it will be necessary to properly define the RB. The RB is what is going to provide the FRBS the knowledge/intelligence to properly provide a correct output from the provided input to obtain a functional and optimal solution to the given problem.

It is important to know that to obtain an optimal result from a FRBS will be necessary to create an initial RB and properly and continuously execute this RB to test the obtained solution. From the obtained solution, modifications/corrections will be necessary to be applied to enhance the results in a way that it will finally possible to obtain the optimal or desired result for the given problem. An example of some rules for this FRBS problem is shown below:

1. **IF** distance IS close **AND** speed IS fast **THEN** acceleration IS brake
2. **IF** distance IS far **AND** speed IS slow **THEN** acceleration IS accelerate
3. **IF** distance IS far **AND** speed IS fast **THEN** acceleration IS brake

The required process to obtain an optimal RB for a given problem can be complex and difficult, which can require plenty of hours of manually testing different RB until the desired solution is obtained. To avoid this tedious process, automatic process to obtain a proper RB for a given problem can be used. This automatic process is obtained through knowledge acquisition systems.

4.4. Knowledge acquisition systems

One key aspect of the *FRBS* is the necessity of an optimized *RB* which knowledge enough to provide the desired optimal result for the specific scenario. Without an optimized *RB* the results may significantly differ from the desired result, which could lead to obtaining worser results that the one obtained through a basic computational logic. Obtaining an optimized *RB* is a must if *FRBS* is in use, however, obtaining it implies an arduous job.

The *knowledge acquisition systems* [10] significantly ease the *RB* optimization process, providing an automated and usually fast system for finding optimized *RB*. Typically, these systems generate randomly *RB* for which with a continuous process of evaluation, will

classify them and generate new ones/evolve them until a certain objective is achieve. Its behavior consist on, once an initial or group of initial *RB* are generated and evaluated to classify all of them and, depending on the strategy/*knowledge acquisition system* in use, create new one or evolve the previous one, for which this new group of *RB* will be evaluated and classified once more, process that will be repeated until a specific condition is achieved, which could be achieving a specific result or until a certain number of execution has been done.

The *knowledge acquisition systems* are divided into two groups, the *genetic algorithms* and those based on a *particle-swam optimization*.

4.4.1. Genetic Fuzzy systems: Pittsburgh

The *genetic fuzzy systems* are a type of *knowledge acquisition system* that make use of *genetic algorithms* [11] for the process of either finding an optimal *RB* or optimizing them.

The *genetic algorithms* are systems that represent problems as particles in a population to try to solve or find a solution to a given problem. The population is made of a group of particles where each particle represents a solution for the problem.

As each particle represent a solution, all of them are evaluated and given a specific fitness that tells how good each solution is. Once done, for the given solutions, a process of solution selection will be done to use them for the *crossover* process to create a new child population from the parent particle. Although the best solutions are preferably used as the parents, elitism should be avoided as bad particles may have useful information not achieve better solution, and so, a certain probability of selection should be given to the worse solution, although in a minor degree.

Once parents have been selected, it must be divided into pairs to use them to create child particles that will represents new solutions. In the same way, the new solutions must be evaluated and subsequently joined to the original population and classified/ordered. Once classified, the worst solution must be erased from the candidates to reduce the whole population to the exact initial population size. Additionally, during the *crossover* process, a *mutation* probability must be added to allow some level of *exploration* to help the new solutions avoid local optimum solutions.

The previously specified process excluding the initial random solution generation, represents a whole *iteration*.

Finally, a condition to stop the *algorithm* must be specified, as specifying the number of *iterations* that the system must execute.

The *genetic algorithm* provides a way for finding optimal or close to optimal solutions for a given problem in a short period of time. Although the solutions provided may not be the best, they will be good/optimal enough and will significantly ease the solution finding process. Due to it, these *algorithms* are only valid when a certain level of error or non-optimality can be accepted as a solution.

An example of *genetic fuzzy algorithm* is *Pittsburgh* [12]. *Pittsburgh* represents a *knowledge acquisition system* for *FRBS* that use *genetic algorithms* to find optimal *RB* as solution. *RB* are now represented by the population, where each particle represents a specific solution or *RB*.

As specified in *section 4.3.* each *RB* will be made of a group of *rules* that will specify through a group of conditions the output for a given input or group of inputs. Through each *iteration*, *Pittsburgh* will find new solutions/*RB* evolving the previous *RB* until the stopping condition is reached, where the system should provide a *quasi-optimal* solution.

4.4.2. Particle Swarm Optimization Fuzzy systems: KASIA

Represents an alternative method for finding solutions to problems to the *genetic fuzzy systems*. Opposed to the *genetic fuzzy systems*, Kasia and the *particle swarm optimization fuzzy systems* are based on the *particle swarm intelligence* [13].

The *particle swarm intelligence* is made of a swarm of particles that works either individually or, without a main entity that leads them, with a common objective to be solved.

Each particle finds solutions locally, for which will store the best solution found and provides it to the rest of the swarm's particles. Particles moves freely move through a search space based on specific parameters and knowledge. These parameters and knowledge are the particles *velocity*, its current *position*, the particle's best position and fitness found, as well as the best position found by the whole swarm. The particle's best position is usually called

as *pBest*, while the best position/result found by the whole swarm is called *global best* or simply *gBest*. Lastly, the *velocity* defines the direction towards the *particle* moves to explore the whole search space.

The parameters previously defined before will be stored in the following vectors:

- **X vector:** defined for each particle, stores the current position for all the axis that composed the search space.
- **pBest vector:** defined for each particle, stores the best position found for all the axis that composed the search space.
- **V vector:** defined for each particle, stores the current velocity.
- **gBest vector:** defined as a single vector shared among the whole swarm, stores the global best position found by the swarm in the search space, storing the position for each axis where the best has been found.

Additionally, additional parameters must be defined to store the fitness/mark for the *X vector*, *pBest vector* and *gBest vector*.

- x_fitness: stores the fitness/mark for each particle current position *(X vector)*
- p_fitness: stores the fitness/mark for each particle best position found *(pBest vector)*
- g_fitness: stores the fitness/mark for the best global position found by the whole swarm *(gBest vector)*

Once explained some basic concepts regarding to the *particle swarm* is necessary to explain three additional aspects which are the *initialization process*, the *updating process* and the *particle movement logic*.

At the *initialization process* the particle swarm are initialized. This process involves the creation of the particles, establishing their location as well as their initial velocities. Both the particles location and *velocity* can be initialized randomly, although some constraints must be established. The particles location must be set within the whole range that corresponds to the search space to the related problem. In the same way, a maximum *velocity* must be established to avoid particles from moving long distances along the search space, which will lead to an aggressive exploration process and no exploitation. Once particles initial location is established, they must be evaluated to obtain the initial fitness *(x_fitness)*. Once the *x_fitness* is obtained, all particles best *(p_fitness)* must be set with the same value as

(x_fitness) as the reference value. Finally, from all the *p_fitness* values, the best one will be selected and set as the global best solution into the *g_fitness vector*.

The *updating process* updates the particles current position, evaluates the new positions, and updates the *fitness* vectors. At first, the *velocity* vector *(V vector)* for each particle is updated, followed by the updating of the particle's positions *(X vector)*. Once done, particles are evaluated by their new positions, updating their *x_fitness* parameter. It is followed by the updating of each particle's *p_fitness* and *pBest*, only if the new $x_{fitness} > p_{fitness}$, process that is at the same time followed by the updating of the *g_fitness* and *gBest* only if the condition stated in *(1)* is fulfilled.

$$p_{fitness_i} > g_{fitness} \; for \; i = 1,2 \ldots n. \qquad (1)$$

The final concept to be explained consists of the *particle movement logic*. This process establishes how particles *velocity* and *position* are updated. *Velocity* vector is updated through *equation 2*, while *position* vector is updated through *equation 3*.

$$v_i = v_i + \varphi_1 * rand * (pBest_i - x_i) + \varphi_2 + rand * (gBest_i - x_i) \qquad (2)$$

$$x_i = x_i + v_i \qquad (3)$$

As it is shown in *equation 2*, two colors are used to distinguish two parts within the equation. The gold colored one represents the *cognitive* part, while the blue colored one represents the *global* part. The *cognitive* refers to the local knowledge acquired by the particle, which refers to the best particle position, while the *global* one refers to the best global result found by the whole swarm. Both *cognitive* and *global* part are going to try to approximate the particle to the best position that have been found, and as expected, both may try to direct the particle to opposite positions. To solve this, the *weights* play an important role for the *velocity* vector update.

To clarify both equations, some parameters are explained:

- *i:* represents the particle parameter selected.
- *v:* refers to the current selected particle velocity
- *x:* the current particle selected position
- φ: parameters that represents a weight. Usually the weights are given a small value (1, 2, or even less). Weights are used to establish the importance of the cognitive and global knowledge. It may be desired to give one weight a higher value than the other to consider either the cognitive or the global knowledge more important than the other, or even give both weight the same value. Consider that a good weight parameter adjustment is a key aspect to obtain optimal results, and inaccurate weight may lead to far from optimal results.

  rand: refers to a simple random value which gives the movement to the velocity particle. The rand must allow to obtain both positive and negative values, with a real numerical range. A range between -1 to 1 is usually a good option to be used, although the final optimal rand range may vary depending on the problem to be solved.

It must be pointed that the above equations are the general equations applied for the original *particular swarm intelligence* and while Kasia use the most of them, it uses a slightly modified velocity equation (*eq. 2).* The only difference with respect to the original equation is the addition of an additional weight that multiply the velocity vector at the equation, as shown at *equation 4*:

$$v_i = w * v_i + \varphi_1 * rand * (pBest_i - x_i) + \varphi_2 + rand * (gBest_i - x_i) \qquad (4)$$

The purpose of the *w* weight is no other than forcing the process of exploration at the initiation of the algorithm, with a later forced exploitation process. At the exploration process, the algorithm allows the velocity vector/particles to make large movements within the problem search space, allowing it to quickly obtain a general view of this search space, obtaining useful information with regard to promising, interesting that may provide optimal solutions. Once the search space has been properly explored, the exploitation process avoids large velocities movements, forcing the particles to make a hard search on those locations of the search space where the best solutions has been found. This behavior is achieved by the implementation of a decreasing exponential within the new *w* weight that allows the initial

big velocities movements at the few first iteration, gradually descending to make the maximum velocity value to allow only small movements on the search space to explore the surrounding of the previously best found locations.

The *particle swarm optimization/algorithm* allows the adjustment of several parameters. If these parameters are incorrectly set, it may provide incorrect or far from optimal solutions, and so it must correct set. Moreover, there is not a single correct configuration, and the correct configuration varies depending on the problem to be solved.

Once all the concepts related to the *particle swarm optimization (PSO) algorithm* are explained, *KASIA* can be explained.

As well as *Pittsburgh*, *KASIA* provides a *knowledge acquisition system* to solve and provide optimal *RB* for *FRBS* to try to reach optimal solutions. In *KASIA*, each particle of the swarm represents a full *RB*, that, as previously explained at *section 4.3.* may be made of several rules. It follows the same exact concept explained at the *particle swarm optimization*, where it is going to be executed through several iterations, for which at each iteration, *KASIA* is going to move each of the *RB* to different positions according to the previous knowledge acquired by particles individually and as a whole entity, thanks to the *cognitive* and *global velocity* knowledge.

4.5. Multi-objective optimization

The *Multi-objective optimization* objective is to provide a solution to a *multi-objective optimization problem (MOP)* [14]. In an optimization problem, most of the scenarios found will requires the optimization of multiple objectives/parameters rather than one single parameter, which implies the presence of a *MOP*. While optimizing and solving a one single objective/parameter problem, solving/optimizing a *MOP* may supposed an arduous job. Optimizing a single objective problem requires little effort, as either if it is a maximization or minimization problem, only one single parameter must be optimized, and so, any solution or modification of the current scenario that suppose a deterioration of the previous results could be directly dismissed. On the other hand, *MOP* implies the optimization of multiple parameters or objective, even if all parameters share the same final objective as enhancing

better results. Due to the presence of multiple parameters, most of time the optimization of one single parameter will directly imply the deterioration of one or all the not optimized remaining parameters, which may lead to achieving worse results after the optimization process attempt.

A simple, first approach to solve these problems could be a *weighted sum of rated solution* for each parameter. The *weighted sum* can present an easy to implement logic for the *MOP*, which may provide close to optimal solutions if and only if the *MOP* is extremely easy, as a two-objective/parameter problem. However, due to the previously explained problem, if a high number of objectives is tried to be optimized, it will not be possible to obtain optimal or close to results due to the optimization incompatibility among different parameters.

An alternative method for the *MOP* is the *Pareto efficiency*, which provides a logic to decide when a solution is or is not consider as optimal avoiding scoring each solution.

4.5.1. Pareto efficiency

Pareto efficiency [14] is an originally created concept used for economy efficiency as a new way of resource allocation. This introduced new concepts that can be implemented/used to solve *MOP*. The concepts introduced by it are the *pareto frontier/front/set*, *pareto dominancy/non-dominancy*, *pareto efficient/optimal*, *pareto improvement* and *pareto optimality*.

Correctly used, these new introduced concepts implement an alternative way for solving *MOP*. As opposed to the *weighted sum*, the *Pareto efficiency* provides a better behavior/performance when a high number of objectives/parameters are involved in the problem, which allows its use for the solving of complex *MOP*. To understand how it works, the previously introduced concepts are explained bellow:

Pareto dominancy/non-dominancy: imply how a value/solution or group of is positioned against another solution or group of. A solution is *non-dominated* when there is not any other solution that provides a better result than the specific selected solution. In the same way, if a solution is *non-dominated* it implies that the specific solution shows *dominancy* against others or is *dominant*. On the other way, a solution is *dominated* when

there is at least another solution that *dominates* it, or in other words, there is at least one solution better than the selected solution. A *dominated* solution does not imply that it cannot *dominates* any other solution, but means that there is better solutions than the one provided by this specific one, being possible that although it is *dominated,* it can *dominates* others at the same time.

Pareto improvement: it refers to the possibility of finding a new solution that improve the previous obtained. To consider a *Pareto improvement*, the new solution must at least improve one parameter from all the parameters that compose the problem to solve without worsening any of the other parameters involved.

Pareto efficient/optimal: a specific solution is considered as *pareto efficient/optimal* when no more improvements can be made for any solution to outperform the solution considered as *efficient*, in other words, when it cannot be found any other solution that dominates the last found solution.

Pareto frontier/front/set: composed by a set of *pareto efficient/optimal* solutions to provide a variety of solutions that are considered as optimal but without directly establishing a solution as best.

To better understand the previous concepts, the objective of *Pareto efficiency* is to provide a group of optimal solutions to *MOP*. *Pareto efficiency* will avoid specifying a single solution as best but will provide a group of that are considered as optimal solutions to solve the *MOP*. The optimal solutions or *pareto optimal* are provided by means of the *Pareto frontier*, where all the *Pareto efficient* solutions/*non-dominated* solutions are shown. The final user will be the one to decide which of the provided solutions is best or most ideal to be used for the specific *MOP*.

To use *Pareto efficiency* to solve *MOP*, is necessary to implement it over a *knowledge acquisition system* to evolve the solutions to better ones. However, *knowledge acquisition system* needs methods to classify solutions from best to worst. Although *Pareto efficiency* already provides methods for this purpose as the *dominancy* and the *Pareto fronts*, these systems need more advanced ways to successfully provide a constant evolution/improvement to the original solutions. For this purpose, the *Pareto ranking* [16] methods are used.

At [15] is shown a work of research over the *Pareto efficiency* together with the *Goldberg* ranking [16][17] method for cloud computing optimization, using it for the simultaneous optimization of multiple parameters, showing the effectiveness of this strategy for the optimization of multiple parameters simultaneously.

4.5.1.1. Pareto ranking

Pareto ranking represents a key tool that must be used at Multi-objective evolutionary algorithms. These algorithms require further classification than the one provided by the *Pareto efficiency* by its own. The classification provided by the *Pareto ranking* allows to classify solutions in different levels of *dominancy*. This new solution allows to give a probability of selection to the different obtained solutions depending on their level of dominancy.

Evolutionary algorithms as *Pittsburgh* requires to classify the different solutions with a probability of selection to develop the different process related to these algorithms such as selection of population, recombination and mutation.

The ranking classifies the different solutions according to their level of *dominancy*. Those that have a higher level of *dominancy* will stay in a lower classification and so will be assigned with a higher probability of selection.

A good example of basic and functional *Pareto ranking* is the *Goldberg's ranking/AM2* [16][17]. This ranking works in the following way. For an initial population of solutions, it will test the *non-dominancy* for the whole population. Those that are not dominated and so dominates the rest of the population will be assigned with the initial *rank of 1*. The number of *non-dominated* initial population particles is counted and so removed from the population to avoid considering them for the next stage. In the new stage, the original *rank 1* is summed by the number of previously counted *non-dominated* particles. As an example, if it was found that 2 particles were *non-dominated* then the new rank will be 3 (original *rank 1* plus 2 *non-dominated* particles). Once this is done and the previously *non-dominated* particles has been excluded, the *dominancy* must be tested again for the new remaining population, where a new set of *non-dominated* particles will be found that dominates the rest of the solutions

presents in the population. This process must be done until all the population has been classified according to its level of *dominancy*.

Once the population is correctly ranked, it will be possible to decide the probability to be assigned to the different ranks, as establishing different probability of selection according to the particle rank.

4.6. Power models

To achieve a better power consumption/reduction and energy, it is mandatory to obtain a proper way to estimate the power consumption develop by any computational resource. For this, *power models* aim to achieve a good, easy to compute estimation that approximate to the real obtained power consumption and energy done by resources. If a good estimation can be achieved, it will be possible to be implemented in a *cloud service scheduler* with the objective of reducing the power consumption and energy considering the estimation in the scheduling process. The energy is given by product of the whole power consumption and the total time where the whole power consumption has been consumed *(5)*.

$$E = P * t \qquad (5)$$

In the computation scenario, the time is given by the specific amount of time a specific resource has been in execution. For this case, it is considered that a resource is in execution while it is executing a specific task, and so, the time is given dividing the total task length (L) and the resource *MIPS (6)*.

$$t = \frac{L}{MIPS} \qquad (6)$$

In this way, it is easily seen how the energy is directly related to the task length and the computational resources of the resource/host that execute the task. The higher the resource *MIPS* is, the lower the time will be, and so, the lower the energy *should be*. However, not

only the *MIPS* must be considered in this process, as the *power* consumption rely on the *resource*. The computation of a computer/resource *power consumption* depends on several parameters, as the *dynamic energy*, which at the same time depends on the *dynamic power*.

The whole energy developed by a computer depends on its *dynamic energy* as well as its *static energy* [18] *(7)*. The *dynamic* component represents the *CPU* (*Central Processing Unit)*, which can alter the consumption according to its *frequency* and the *voltage* required to bias and maintain the selected *frequency*, while the *static energy* represents which will not vary the energy.

$$E = E_d + E_s \tag{7}$$

The *dynamic energy* is given by the product of the *dynamic power* and the execution time, as specified in *(8)*.

$$E_d = P_d * t \tag{8}$$

At the same time, the *dynamic power (9)* depends on parameters that are directly related to the *CPU*, which are the *voltage*, *frequency*, its *capacitance* and a constant.

$$P_d = k_d * C * V^2 * f \tag{9}$$

At the same time, the *static energy* [19] is considered to depend on the *dynamic energy* multiplied by a *static constant* k_d.

4.7. DVFS

Dynamic Voltage and Frequency Scaling (DVFS) refers to the technology that allows modern *CPUs* to dynamically adjust their voltage and frequency under certain scenarios/situations. The *CPUs* will be able to adjust these parameters depending on the *CPU load* so that it will only raise its working *frequency* when necessary.

Voltage and frequency are directly related. To maintain a stable high frequency, a certain amount of voltage will be required to be used to correctly supply the *CPU*. This means that to obtain a high frequency, it will be required to increase the voltage supply, and so, the power consumption will increase, while on the other hand, if the *CPU* is working in a low frequency, the voltage supply may be reduced which implies a lower power consumption.

DVFS represents a key tool to reduce the power consumption. As *CPUs* can vary its working frequency and so its voltage supply according to the load or amount of work they must do, it will be possible to dynamically reduce the power consumption. If a *CPU* is in idle state, the frequency can be reduce, and so the power consumption will as well, and, at the same time, if it is necessary to execute a task, the *CPU* will be able to increase its frequency to the necessary one only the required time to execute the task, making a high power consumption only when necessary.

The use of *DVFS* relies on two conditions, on the one hand, the CPU must support it, and on the other hand, it must be implemented at a *kernel* [20] level within the *OS*.

Additionally, the *kernel* side *DVFS* control relies on *governors*. *Governors* specify the strategy to follow for the *CPU* frequency to be used. Can be divided in two different groups, static and dynamic. The static ones fix a frequency that will be use, not allowing any variation of it over time. On the other hand, the dynamic governors follow specific algorithms that allow the variation of the frequency and voltage over time.

Moreover, there are five basic *governors* that are explained to further clarify the behavior of *DVFS*.

Static:

- **Performance:** aimed to provide the highest performance possible, this *governor* set the *CPU frequency and voltage* at its maximum possible. As the main advantage, the CPU will always provide the highest performance possible, although as the main disadvantage the consumption level will be the highest even when the CPU is idle. Thanks to its behavior, tasks will not be executed as soon as arrive and completed in the least time possible.
- **PowerSave:** provide the opposite behavior as *performance*. Set the CPU *frequency and voltage* to the lowest available to ensure the lowest power consumption. Due to this, tasks will require more time to be executed compared to *performance*.
- **UserSpace:** allows the user to manually set the *frequency and voltage* desired to be used from the available/supported by the CPU.

Dynamic:

- **OnDemand:** aimed to provide the best response possible to provide the maximum performance allowed by the CPU when needed but allowing it to reduce its frequency. It works with a single threshold that delimit, according to the CPU load, if the frequency must be increased or decreased. Usually set at 95%, if the CPU usage goes above it, the CPU will count 100 iterations and if after it the CPU usage stays over it, the frequency will be increased along with it voltage to the highest one supported by the CPU if it is not already at its maximum. On the other hand, if the CPU usage goes below this threshold, it will count once again 100 iterations and if after it the CPU load continues below it, the frequency and voltage will be decreased one single step only if it is not already at its minimum value.
- **Conservative:** contrary to *OnDemand*, this *governor* tries to reduce the power consumption to the least possible while allowing to dynamically modify the frequency and voltage depending on the CPU load. It works with two thresholds, the *setUpThreshold*, usually set at 80% of the CPU load, and the *setDownThreshold*, usually set at 20% of the CPU load. If the frequency stays above the *setUpThreshold* after 100 iterations, the frequency and voltage will be increased one single step if possible. On the other hand, if the frequency stays below the *setDownThreshold* after 100 iterations, the frequency and voltage will be decreased one single step if possible as well.

4.8. Available cloud simulation platforms

For the development of this master thesis a simulator previously developed at my bachelor thesis is used, called *WorkflowSim with DVFS and meta-scheduling.* This simulator supposes an evolution over previously developed cloud simulators. The evolution of the simulators is shown in *figure 4*.

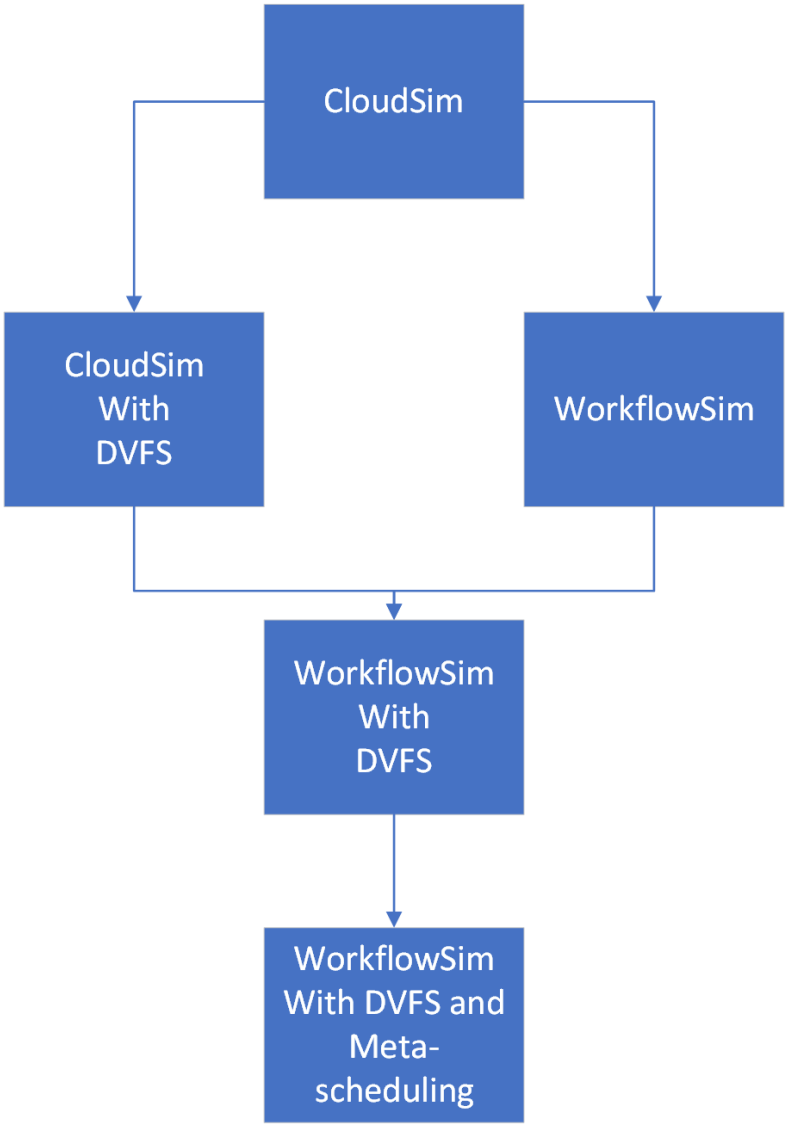

Figure 1: simulator evolution

Each of the new evolution introduce new functionalities to enhance the capabilities of the simulator, increasing its flexibility, versatility, but at the same time its complexity. This simulator family count with a really high level of complexity, being made of different

components that must be widely studied and understood to be able to correctly use them. At the same time, as the whole simulator family is an evolution of the previous one, concepts that are explained/introduced in earlier versions are valid for the newer one.

In the following sub-sections, each of the simulators are going to be explained as well as its functionalities, capabilities and new features introduced at each version.

4.8.1. CloudSim

ClodSim [21][22] is a very first cloud simulator written in Java that allows the creation of basic simulation scenarios. Even though it allows the simulation of basic scenarios, its structure is complex, being made of several different elements that works together to allow the creation of different scenarios, presenting a *modular* structure. However, thanks to its *modular* structure, it allows the creation of different scenarios and configuration for testing purpose. Moreover, its *modular* scenario allows the inclusion of newer, more complex elements to increase the simulator capabilities providing an easy evolution that gave birth to the creation of newer versions.

CloudSim is mainly made of 4 type of elements which are the *basic elements*, *entities*, *core*, and the *event system*.

4.8.1.1. Basic elements

CloudSim basic elements provide basic functionalities to the simulator, working with entities to contribute to create the whole cloud simulation system. The main basic elements in the simulator are the *host, Pe, VM and cloudlet.*

- Host: represents a host within the cloud scenario. Several parameters can be established to specify its characteristics as the amount of RAM, storage, and the processor. The processor is specified by an object of the class *Pe*, where one or several of this basic element can be set up within one single *host*. Additionally, a *host* by itself cannot execute task, and a *VM* must be built over it for this purpose.

- Pe: the *processor element (Pe)* represents the CPU of a *host*. This element is made up of two attributes, an identifier and its *MIPS* capability.
- VM: *Virtual Machine (VM)* represents elements that run over a selected *host*. The *host* must have at least one *Pe* to have processing capabilities. A *VM* can allocate all the *host and Pe* resources or just a part of them, allowing the execution of multiple of them on a single *host*. Once a *host* is running a *VM*, tasks (*cloudlet)* can be scheduled for its execution. At the same time, each *VM* will have its own identifier which allows the scheduler to schedule tasks to the desired *VM* where must be executed.
- Cloudlet: the *cloudlet* represents the basic task in the simulator. Cloudlets are made of an identifier and its length. On the one hand, the identifier is used to identify the task by the scheduler, while on the other hand, its length identifies the amount of resources or millions of instructions necessary for its execution.

4.8.1.2. Entities

Entities are big, complex elements that take an important role within the simulator. Entities is what provide the simulator a high level of versatility and allows to build all kind of simulation scenarios. Each entity provides a specific function to the simulator, having a total of 4 called *CloudSimShutdown, CloudInformationService, Datacenter* and *DatacenterBroker*.

Moreover, entities use a complex *event system* to communicate among them, based on different *tags* that identify the type of event.

- CloudSimShutdown:

Its behavior consists on, once there are no more events to be delivered to any entity and the previously delivered one has been received, inform about this situation to each entity to ensure that all are properly stopped to correctly finished the simulator execution.

- CloudInformationService:

In charge of the registration of all the *datacenter* and *datacenterbroker* in the system. *Datacenterbroker* are going to request the *datacenter* characteristics through this entity.

- Datacenter:

Representing a *datacenter* in the simulation system, this entity oversees providing *cloudlet* execution capabilities. For this purpose, this entity is going to gather a set of *hosts* with its respective *Pe* and *VM*.

- DatacenterBroker:

Represents the *cloud scheduler*. This entity oversees, from an initially assigned bag of task *(cloudlets)*, its scheduling to the available *hosts* within a *datacenter* to complete the execution. These *cloudlets* are assigned to the *VMs* that are in execution over the *hosts* within the *datacenter*. Once a *cloudlet* is delivered for its execution to a *VM*, its waits for its completion and, once finished, the *VM* will return it to the *datacenterbroker*. Moreover, *datacenters* and *datacenterbrokers* must be paired. One *datacenterbroker* must behave as the *scheduler* for a *datacenter*, and so, it will have control over the assigned *datacenter* and so use the available resources as convenient to complete the tasks execution.

Some additional concepts regarding to the simulator must be explained. As this simulator represents a very basic first approach to the cloud simulation, even though it allows the creation of quite different scenarios, its capabilities are really limited. It requires the manual creation of tasks, hosts, pe, VM, datacenters and brokers (schedulers), not only that, but it also requires the manual scheduling of tasks to make the brokers send tasks to the VMs running over the hosts gathered in its associated datacenter. This behavior creates a not real scheduling scenario, as neither automatic nor smart scheduling can be done, and it completely rely on the user that creates the simulation scenario. However, thanks to this entities and basic elements scenario it allows to implement additional and more complex elements and entities which lately resulted in the evolution of this simulator to the newest one detailed in the following sections.

Finally, entities are not created right from the start, but they all evolve from a basic entity called *SimEntity*. The *SimEntity* represents the root entity from where all the entities evolve, making use of the Java Inheritance. This basic entity implements some basic parameters and functionalities that are used by all the system entity. It implements the entity identifier *(id)* methods to obtain it, and a complete set of functions to create and send *events* to communicate entities among them. The entities dependency is shown in *figure 5*.

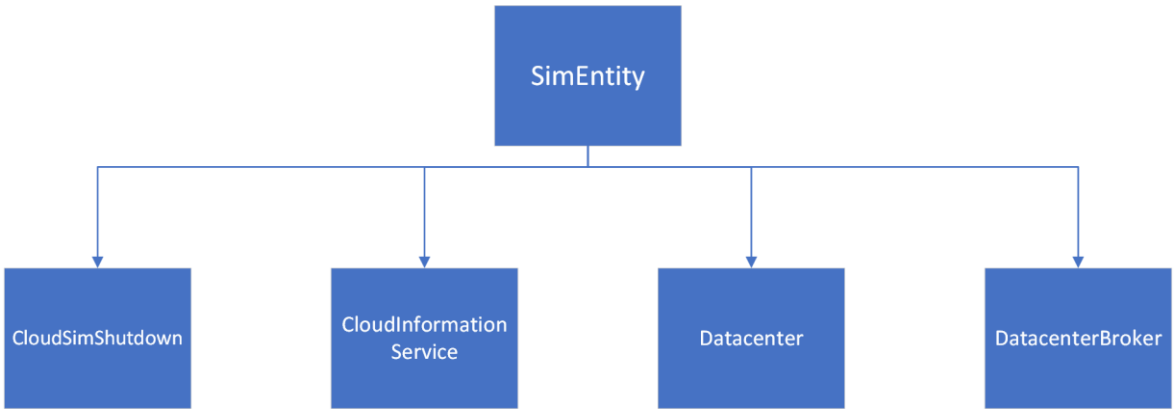

Figure 2: CloudSim entities dependency

4.8.1.3. Core

CloudSim simulator relies into a *core/kernel* that oversees controlling all the simulation environment, managing entities creation, execution, and the existing system *events*. This core/kernel is given the name of *CloudSim* as the simulator. In *figure 6* is shown how entities and events rely over the *CloudSim* entity for its communication, its correct behavior and execution.

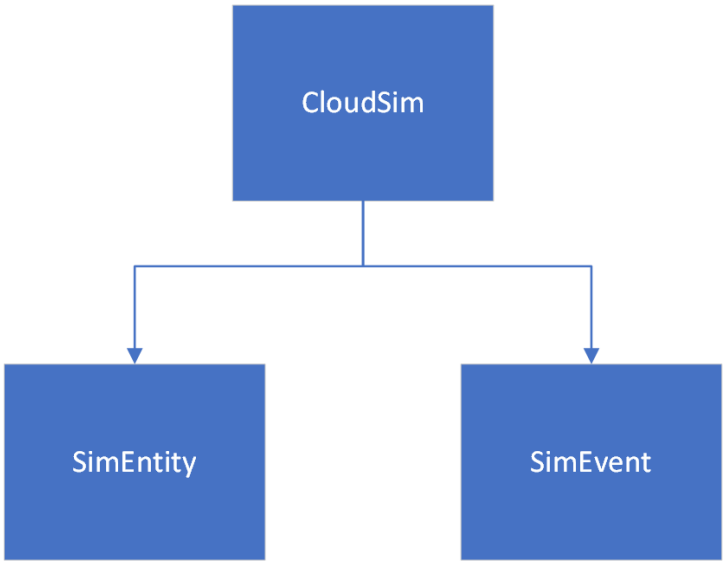

Figure 3: CloudSim core

4.8.1.4. Event system

Probably one of the most complex elements of this simulator, the event system is used to communicate the different entities in a really controlled way. The simulator is made by a series of different events and so each one will be numerically identified, so that each time an entity wants to communicate with any other, it will use a specific event for a specific purpose, as each event has its own specific purpose. Understanding the event system is a key element if it is desired to correctly managed, control and understand the way this simulator is implemented and works. For this reason, this section is written to try and ease the understanding of this essential part of the simulator.

TAG	Source	Destination	Event Name
-1	Broker	Broker	END_OF_SIMULATION
2	Datacenter	CIS	REGISTER_RESOURCE
6	Broker	Datacenter	RESOURCE_CHARACTERISTICS
6	Datacenter	Broker	RESOURCE_CHARACTERISTICS
15	Broker	Broker	RESOURCE_CHARACTERISTICS REQUEST
20	Datacenter	Broker	CLOUDLET_RETURN
21	Broker	Datacenter	CLOUDLET_SUBMIT
32	Broker	Datacenter	VM_CREATE_ACK
32	Datacenter	Broker	VM_CREATE_ACK
33	Broker	Datacenter	VM_DESTROY
41	Datacenter	Datacenter	VM_DATACENTER_EVENT

Table 3: CloudSim events

At *table 3* it is shown the most used events within *CloudSim*. The simulator includes more events than the show, although those explained are the one that is normally used. Additionally, *tags* and hence events are defined in a specific class for its purpose called *CloudSimTag*s.

END_OF_SIMULATION: the only purpose of this event is to inform the schedulers about the finalization of the simulation, for which the schedulers will need to execute a VM_DESTROY event before stopping their own execution. This event is generated by the

engine entity once all the sent tasks to be executed to the schedulers/datacenters have been received and so there is no more tasks to be executed, for which it will inform of this situation to these entities to stop their execution.

REGISTER_RESOURCE: event created/used by the schedulers and datacenters entities to register at the CloudInformationService entity at the beginning of the simulation. It is used to register at the CloudInformationService the different schedulers and datacenters equivalent entities, for which later a discovery process is done to find the available datacenters within the simulation scenario.

RESOURCE_CHARACTERISTICS: this event is used by the datacenter to response to a RESOURCE_CHARACTERISTICS_REQUEST sent by the scheduler. Through this event, the datacenters are going to provide its associated schedulers with information related to themselves. This information is gathered into a DatacenterCharacteristics object that will store information related to the CPU architecture, the operating system used at the datacenter, as well as other information about monetary cost produced by the task processing that is not currently in use at the current implementation.

RESOURCE_CHARACTERISTICS_REQUEST: used by the schedulers to request information about the datacenters in the simulation environment.

CLOUDLET_RETURN: used by datacenters to return a completed task that was assigned to a VM for its execution.

CLOUDLET_SUBMIT: used by schedulers to deliver a task to be executed to a VM within its assigned datacenter.

VM_CREATE_ACK: generated from a scheduler to a datacenter and vice versa. When generated by a scheduler, it requests the creation of a VM within its associated datacenter into a specific host. When generated by a datacenter, it responses with a positive or negative acknowledgement to the creation of the VM.

VM_DESTROY: once there are no more tasks to be executed within a datacenter and all the delivered tasks have been executed and returned to the scheduler, this entity will send it to the datacenter to request the destruction of the VM to free up the resources used by the VM.

VM_DATACENTER_EVENT: generated and delivered by a datacenter to itself, this event is created with the aim of, once it has tasks to execute, order its execution. Once tasks

are completed, the datacenter will send a CLOUDLET_RETURN event to its associated scheduler.

To finish with *CloudSim*, its two major drawbacks must be highlighted. The simulator *does not implement a real automatic and smart scheduling process*. Its schedulers, called brokers, must be manually programmed a provided with the tasks that must be retrieved to the datacenter, having to specify which VM must execute which task. Additionally, it only provides task's time execution information, and *make no energy estimation*. For a given group of tasks, once it has been properly configured, it will estimate the execution time required to complete each task based on its characteristics and the VM/host capabilities.

4.8.2. CloudSim with DVFS

CloudSim with DVFS [23] is a first evolution of *CloudSim*. This evolution seeks to solve one of the two major drawbacks, being it the non-possibility to estimate the task execution power consumption. As *specified before*, the main advantage of *CloudSim* is its high modularity thanks to its division in *basic elements, entities* and *events*. Thanks to this, its evolution to *CloudSim with DVFS* was possible. Most of the simulator is as specified at the *CloudSim section*.

To evolve the simulator to support the energy/power consumption estimation, both *power models* and *DVFS* have been implemented within it. This was achieved by evolving two entities and two basic elements.

4.8.2.1. CloudSim with DVFS entities evolution

In *figure 7* it is shown the evolution of the simulator entities. The datacenter and broker have evolved to the *PowerDatacenter* and *PowerDatacenterBroker*. These two new entities use java inheritance to extends the capabilities and functionalities of their root entities.

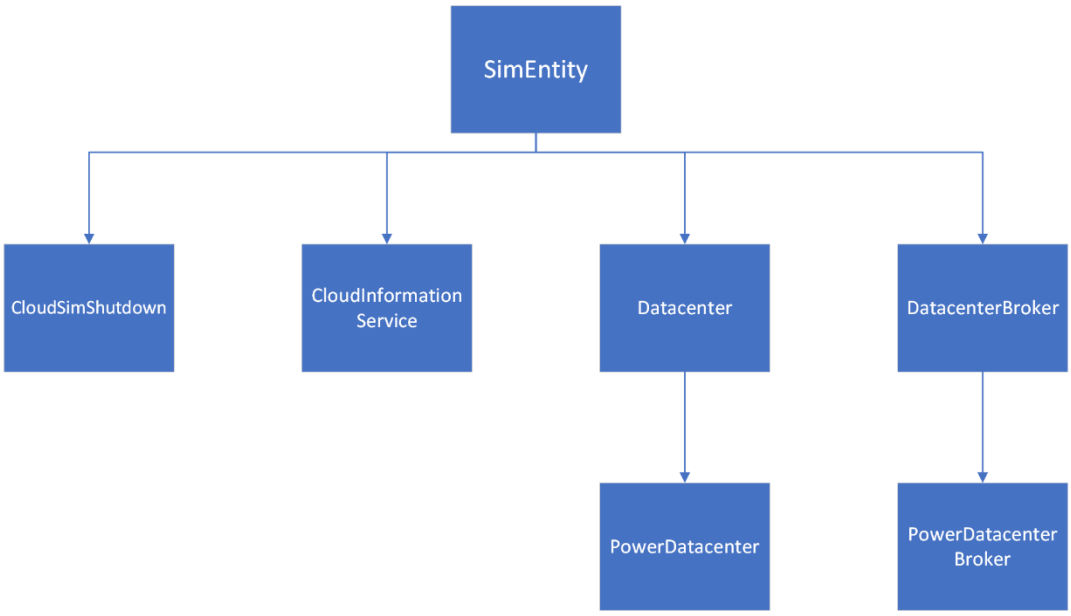

Figure 4: CloudSim with DVFS entities evolution

4.8.2.2. CloudSim with DVFS basic elements evolution

In *figure 8* is shown the evolution of the basic elements. Both the Host and Vm elements have evolved into the PowerHost and the PowerVm. As the entities, java inheritance has been used to extends the original capabilities and functionalities.

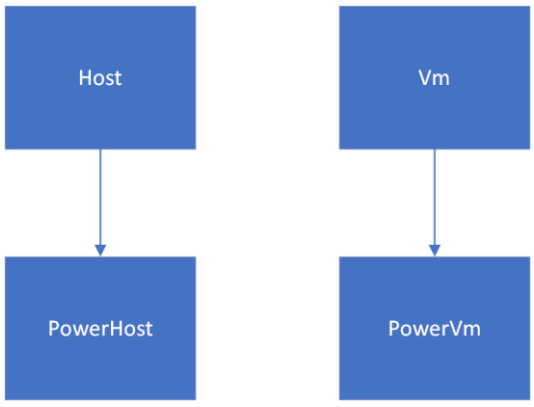

Figure 5: CloudSim with DVFS basic elements evolution

4.8.2.3. CloudSim with DVFS capabilities

Thanks to the implementation of the power models, the simulation of real CPUs was introduced in the simulator. For this, power consumption measures must be done over the real CPU to be simulated to gather information about it. This process consists on manually fixing the CPU frequency and voltage to each of the supported values, and for each one, measure the consumption both with the CPU at idle (0% of use) and under stress (100% of use). Once all the measures are done, it can be used to create a CPU simulation to be used at the simulator. It allows a proper study and estimation for real cloud scenario, as it allows to build real hardware scenarios and obtain a total estimation of the power consumption and energy due to task scheduling.

On the other hand, at this stage, the tasks execution simulation differs from its implementation in CloudSim. In CloudSim, as soon as a task is assigned to a VM for its execution, it divides the task length by the VM MIPS capacity and report the necessary time for its execution. In the later version, it is necessary to report power consumption and energy due to the CPU usage at a certain frequency, and so, tasks must be executed in small portion of time to allow the CPU frequency and voltage variation due to the DVFS implementation. For this, an interval of time called *SchedulingInterval* is implemented. This new parameter, with a default value of 0.01 seconds, simulates the task execution for the specified interval of time, which allows a close to real scenario power consumption and energy estimation. Additionally, the division of the task execution simulation in small intervals allows to simulate the *DVFS* behavior for which a CPU will modify its frequency and voltage according to the CPU load and the selected governor.

Moreover, each datacenter is going to record the total consumption developed by their hosts/VMs, and so, it will provide when requested the total power consumption. For the study of consumption, the parameters showed at *table 4* are provided after a simulation is finished.

Information	**Units**
Total simulation time	Time in seconds
Power sum	W
Power average	W
Energy consumption	Wh

Table 4: CloudSim with DVFS time and consumption report

Finally, this evolution to CloudSim does not implement new events for its behavior and use the one already implemented within CloudSim.

4.8.3. WorkflowSim

WorkflowSim [24] consist in another evolution of CloudSim, alternative and parallel to CloudSim with DVFS evolution. Opposite to CloudSim with DVFS, this alternative evolution does not provide any of the improvements introduced on it, and so, there is not any possibility to estimate/simulate power consumption and energy at tasks execution, and so will only provide information related to the tasks time execution.

This CloudSim evolution introduce several improvements over its original simulator. WorkflowSim includes the simulation of workflows by the implementation of DAGs, and automated/smart task scheduling.

DAGs are implemented in the simulator through DAX files, which make use of XML to define a list of tasks and its dependencies, each one with its ID and length.

On the other hand, the smart, automated task scheduling provides schedulers entities with the ability to automatically assign the tasks to be executed defined in the DAX to VMs within its associated datacenter for its execution according to the strategy specified in the scheduling algorithm used. The simulator supports a wide variety of scheduling algorithms, being the most important from the implemented one, those specified at *section 4.2.2*.

The simulator behavior with respects to its planning algorithm/strategy is reduced to a single behavior. All tasks are going to be treated in an on-batch strategy, not allowing any other strategy to be used. It makes all tasks to be grouped in *BoT*, and so tasks will be scheduled in groups rather than individually among all the available resources at the datacenter.

To achieve these new functionalities, further implementations were done in the simulator with respect CloudSim. The modifications consist in the inclusion of further entities as well as a new basic element and a new event system.

4.8.3.1. WorkflowSim entities

The new functionalities implemented in CloudSim to create WorkflowSim requires a big an in-depth modification of the old entities structure. Some entities have been evolved, while additional ones have been introduced.

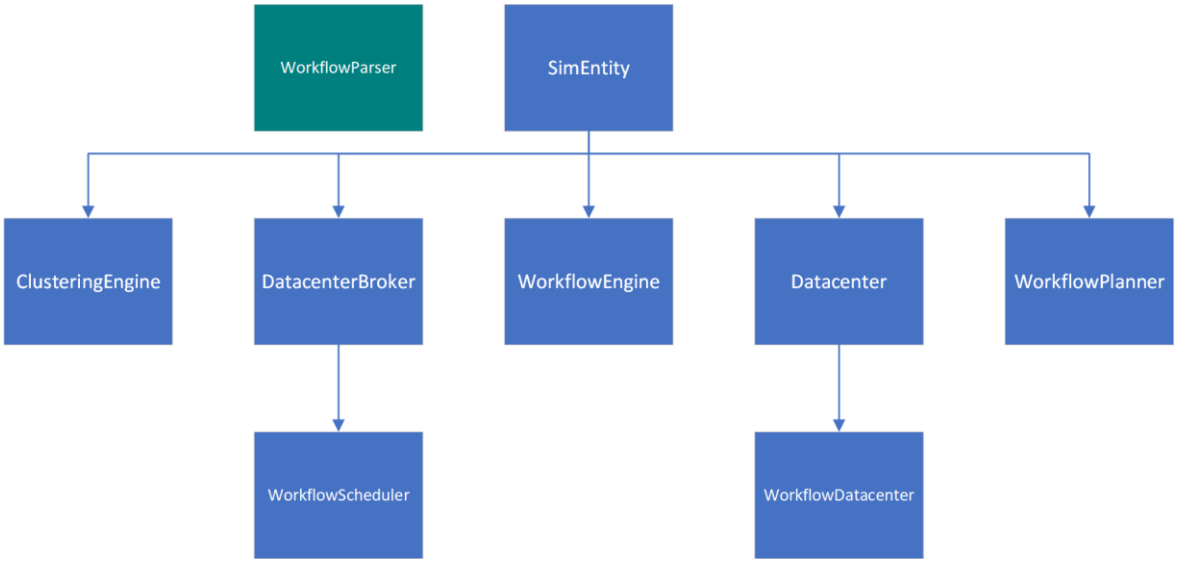

Figure 6: WorkflowSim entities dependency

In *figure 9* is shown the entities dependency for WorkflowSim simulator. As shown, a total of 4 entities has been added, while other 2 have evolved. All entities inherit either from SimEntity or other entities that inherit from it apart from the WorkflowParser entity, which is a special case. This new entity did not inherit from SimEntity as it does not behave as a normal entity with the ability to execute by itself but must be executed through the WorkflowPlanner entity. Additionally, the WorkflowParser does not have the ability to send events. Bellow the entities purpose are explained in detail.

WorkflowPlanner: the purpose of this entity is, at its instantiation, it will create the *ClusteringEngine* entity, which at the same time creates the *WorkflowEngine*, that creates the *WorkflowScheduler*. At the instantiation of this entity it will be necessary to indicate the number of WofklowScheduler entities to be created. Additionally, it will create the *WorkflowParser* entity, request it to parse the tasks defined in the DAX and obtain them to later send them to the *ClusteringEngine* through an event.

WorkflowParser: entity with the inability to send events. It only purpose is to take the DAX file that define all the tasks to be executed and parse them into simulator tasks.

ClusteringEngine: its objective is to receive the tasks from the *WorkflowPlanner* and, after creating the *WorkflowEngine*, send deliver the tasks to this entity through an event.

WorkflowEngine: the objective of this entity is to make a proper control of all the tasks to be executed and its execution restrictions. Due to the use of DAG, tasks execution is restricted and conditioned to the prior execution of their parent's tasks, and so, a task cannot be executed until all its parent tasks execution is completed. This entity controls it and so will only send tasks to the *WorkflowScheduler* entity for its scheduling process if the tasks has no parent to be executed or all its parent's tasks have been executed. For this, the entity will receive all the completed tasks from the schedulers entity to which tasks have been retrieved.

WorkflowScheduler: its objective consists in, once it receives tasks from the WorkflowEngine to be executed, it will schedule them for their execution to the available VMs within its associated WorkflowDatacenter entity. To deliver/assign a task to a specific VM, it must be idle (not executing any task). The scheduling process is defined by the scheduling algorithm specified by the scheduling algorithm used at the scheduler entity.

WorkflowDatacenter: the new implementation of the datacenter entity provides the ability of using a new type of VM called CondorVM for the execution of tasks.

4.8.3.2. WorkflowSim basic elements

WorkflowSim introduced additional basic elements to provide the new functionalities specified. The new basic elements introduced are shown in *figure 10*, being the *CondorVM*, *Task* and *Job*.

CondorVM supposes an evolution over the classic Vm element that represents the Virtual Machines within the simulator. Implemented through java inheritance over the Vm, this new entity introduces a new parameter to be used within the VM, which is its state. The state allows to specify whether a VM is busy or idle, representing the busy state a VM that is executing a task, and idle the opposite.

Task and Job both evolve from the Cloudlet through java inheritance. Task introduce the ability to establish dependency and execution restrictions among tasks, to add the possibility of representing a DAG. On the other hand, Job, which inherit from Task, is just a simple

container/group of Task to represents a bigger task that is composed by multiple tasks at the same time.

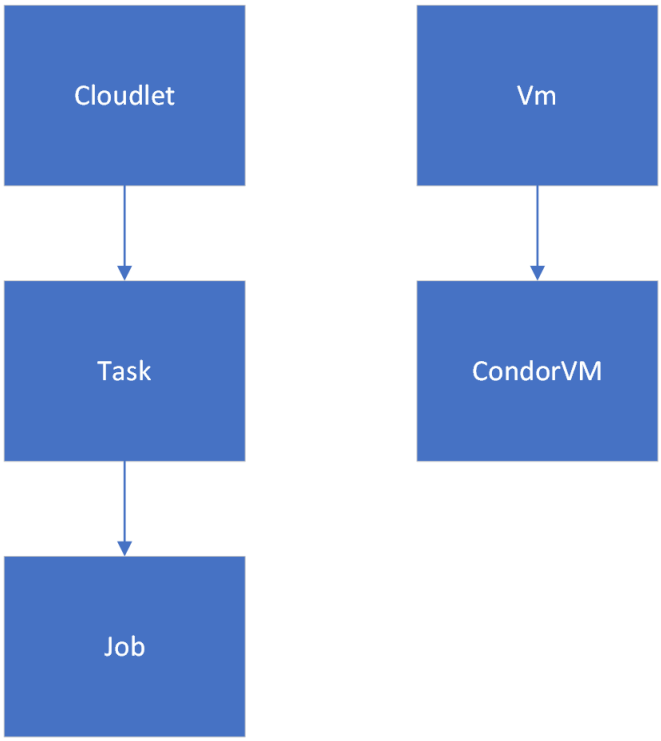

Figure 7: WorkflowSim basic elements evolution

4.8.3.3. WorkflowSim events

WorkflowSim introduced new events used by the new entities to communicate among them and use their new capabilities. In *table 5* is shown the new introduced events.

TAG	Event type
1000	START_SIMULATION
1001	JOB_SUBMIT
1005	CLOUDLET_UPDATE

Table 5: WorkflowSim new events

START_SIMULATION: this event is aimed to be sent to the planner entity, for which it is going to develop the action of taking the DAX file in where the different tasks are

defined and translate them into tasks that can be understood by the simulator and so the other entities. Once done, it will trigger a JOB_SUBMIT event.

JOB_SUBMIT: this event is sent by the planner entity with the objective to provide the engine entity with all the tasks that must be executed in the simulation scenario. Once received, it will store all the tasks into a job list for later use when receiving CLOUDLET_RETURN.

CLOUDLET_UPDATE: this event is implemented at the scheduler. It will trigger the use of the selected scheduling algorithm to make the scheduler to, according to the strategy implemented within the algorithm, decide for each VM which VM within its associated datacenter is the proper one to execute the task.

4.8.4. WorkflowSim with DVFS

WorkflowSim with DVFS [25] emerged as a new evolution for the CloudSim simulation family. It was made by the union of the WorkflowSim and CloudSim with DVFS, and provides the benefits introduced in both simulators. The new simulator has the capabilities to simulate and work with workflows defined by DAG, smart, automatic scheduling as well as power consumption and energy estimation capabilities due to the integration of the power models and the DVFS.

WorkflowSim was used as the base for the creation of this new simulator, for which the classes that provided the power/energy estimation capabilities in CloudSim with DVFS has been added. Due to this, the main simulator class and entities structure remains almost unchanged, although some minor modifications were done to provide the new required capabilities to achieve the DVFS state.

The main advantages obtained through the implementation of DVFS over WorkflowSim is a better, closer approach to a real datacenter scenario. With this simulator, thanks to the inclusion of all the benefits introduced on the two simulator on which this one is based, it is possible to make a proper study over power aware scheduling algorithms, so that new ones can be created through researching to try and reduce the power consumption and energy developed by datacenters due to the execution of tasks, as the scheduling algorithms will follow strategies for this purpose.

4.8.4.1. Entities

The simulator maintains almost the same structure as the one found in WorkflowSim, presenting some minor modifications at the entities structure to achieve the new capabilities. In *figure 11* is shown the entities dependency for this simulator.

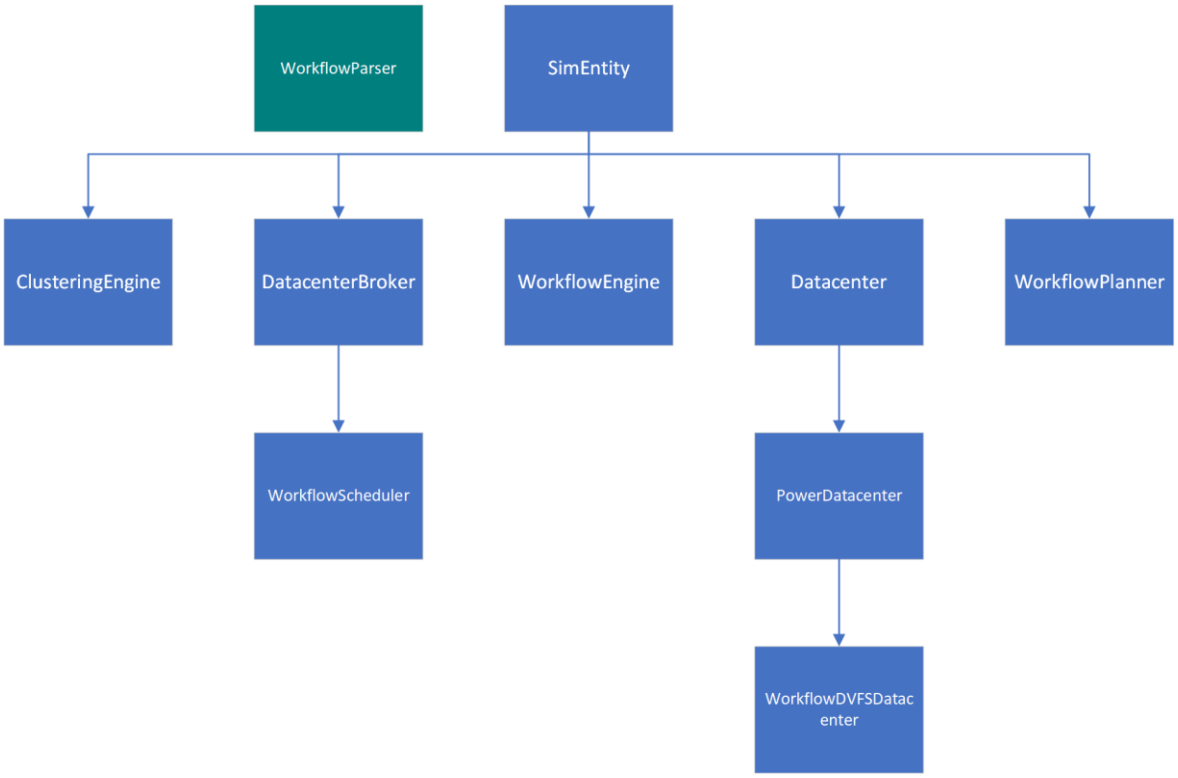

Figure 8: WorkflowSim with DVFS entities dependency

Comparing the entities dependency between WorkflowSim and WorkflowSim with DVFS there is one single modification which relates to the *datacenter entity* implementation. Before, the datacenter entity was implemented over the classic *Datacenter* implemented in CloudSim, for which a new entity called *WorkflowDatacenter* inherited from it. Now, this last entity has been modified to inherit from the *PowerDatacenter* entity, which corresponds to the one introduced in CloudSim with DVFS. From this entity that inherit from *Datacenter*, the *WorkflowDatacenter* entity is built to inherit from it and create the *WorkflowDVFSDatacenter* entity.

The new entity, due to the inherit process provides both, the benefits of the datacenter entity from *CloudSim with DVFS* and from *WorkflowSim*. From the CloudSim with DVFS implementation, it provides capabilities to simulate real CPUs, which provides the DVFS capabilities and so, the power consumption and energy estimation due to the processing of tasks. On the other hand, the mixing with the WorkflowSim datacenter entity provides VM estate capabilities to specify whether a VM is busy or idle, which is necessary to implement the automated tasks scheduling, as the scheduler entity must know at every time each VM state to know if it is possible to schedule a tasks to it or not.

4.8.4.2. Power model for research purpose: the PowerModelAnalytical

As specified in *section 4.6*, PowerModel introduces an effective way to simulate the power consumption developed by a CPU. Implemented within CloudSim with DVFS, certain PowerModels are provided that simulates, together with the DVFS, specifics real CPUs. However, in some scenarios, especially for the study and research of better scheduling process aimed to reduce the power consumption and energy, the proposed and provided power models within CloudSim with DVFS provide a homogeneous scenario, which means that, within a datacenter, all the available hosts will have the same characteristics, and so, if all the VM allocates the same resources for each host, the obtained results will be the same independently of the scheduling algorithm used. This creates a situation where it is not possible to properly study new scheduling algorithm, whether its objective is to minimize the tasks execution time, the power consumption/energy, or both. For this, a new power model was created and provided within WorkflowSim with DVFS simulator.

The newly provided power model called *PowerModelAnalytical* aims to solve the specified problem. It allows to manually specify the CPU parameters that define its power consumption as well as processing capabilities, which is defined by several parameters. In *table 6* is shown the PowerModelAnalytical parameters.

Parameter name	units	Type of data
maxFrequency (f)	Hz	double
maxVoltage (V)	V	double
dynamicConstant (k_d)	%	double
staticConstant (k_s)	%	double
capacitance (C)	F	double
percentages	%	Array of doubles
IPC	-	double

Table 6: PowerModelAnalytical parameters

To clarify, the *dynamicConstant, staticConstant* and *percentages* parameters represents percentages specified from 0 to 1. Additionally, the *percentages* parameter represents all the frequencies that a CPU is capable of working, computed from its maximum frequency (*maxFrequency*). As an example, if the maximum frequency of a CPU is set to 2 GHz, and its percentages parameters indicates values of 0.5, 0.75 and 1, the frequencies at which the CPU can work are 1, 1.5 and 2 GHz. As specified at the PowerModelAnalytical code, the dynamic constant represents the active of gates within the processor. Moreover, the IPC represents the Instruction Per Cycle, which tells, for each CPU frequency Hz, the number of instructions it can compute. The IPC parameter depends on the CPU architecture, which generally, the newer a CPU architecture is, the higher its IPC will be.

From these parameters the CPU power consumption as well as its maximum performance expressed in MIPS can be computed. For this purpose, the following equation are used.

In *eq. 9* is shown how the CPU dynamic power is computed. As shown, the power dynamic component depends on the dynamic constant, the CPU capacitance, voltage and current frequency

$$P_d = K_d C V^2 f \tag{9}$$

Once the power dynamic component is computed, it is possible to compute the total power consumption, shown in *eq. 10,* which depends on the power dynamic component as well as the static constant.

$$P = \frac{P_d}{1 - K_s} \tag{10}$$

And finally, the final parameter to be computed is the MIPS as shown in *eq. 11*, which represents the CPU performance capabilities and depends on its frequency and IPC.

$$MIPS = \frac{f * IPC}{10^6} \quad (11)$$

4.8.4.3. Power aware scheduling algorithms

Historically, scheduling algorithms have always been focused on following strategies to minimize the tasks execution time required to compute them. At this simulator, its author focused efforts to not only reduce time, but also to reduce the power consumption and energy due to tasks processing, and most important, focused on minimizing both parameters at the same time. For this purpose, new scheduling algorithms were developed, dividing them into simple algorithms and complex, that make use of FRBS.

Initially, two new, simple power aware scheduling algorithms were introduced, that seeks to reduce the power consumption called PowerAwareMaxMinSchedulingAlgorithm and PowerAwareMinMinSchedulingAlgorihtm. Both suppose an evolution from the classics MaxMin and MinMin scheduling algorithm, with the peculiarity that rather than seeking for the host/VM that guarantees the minimum execution time, the algorithms looks for the host/VM that guarantees the lowest power consumption to execute a task.

As well as MaxMin and MinMin, these two news algorithms follow the same behavior, which divides their behavior in two different stages. At the first stage, the power aware MaxMin algorithm prioritize the scheduling/execution for the biggest task, those that need the highest amount of MI (Millions of Instructions), while the power aware MinMin provides the opposite behavior, prioritizing the lowest one for its execution and scheduling. The second stage remains the same for both algorithms, for which for the selected task to be schedule and executed, it seeks for the host/VM that guarantees the lowest power consumption.

At first sight, the two newly introduced algorithms seems to works properly, following a logical thinking to reduce the power consumption, although some problems related to the specific scenario state can easily make it perform even worse than on the classic algorithms and provide the opposite desired result. By prioritizing the tasks execution to the available host/VM that guarantee the lowest power consumption can provide incredible undesired

results, where the scheduling of a task to a VM may force another task to be scheduled later on to another resource and may make the second task execution to last a higher amount of time than necessary, and so, finally could increase both, the total execution time as well as the energy.

Due to this, the author explored the possibility of developing more advanced scheduling algorithms to prevent the problem exposed before.

4.8.4.4. Power aware FRBS scheduling algorithms

Advanced scheduling process can be an arduous work. To achieve a more advanced and smarter scheduling process, the author made use of FRBS. Thanks to this, multiple parameters were possible to be used at the scheduling process to implement a proper scheduling process that were able to enhance the results obtained through both, the time-oriented and the new power aware scheduling algorithms. A total of two new scheduling algorithms were implemented, both making use of the FRBS, and using as the base scheduling algorithms for their implementation MaxMin and MinMin algorithms. The two new algorithms were given the following names:

- **PowerAwareFuzzyMaxMaxSchedulingAlgorithm**: based on MaxMin and sharing its first stage behavior.
- **PowerAwareFuzzyMinMaxSchedulingAlgorithm**: based on MinMin and sharing its first stage behavior.

At the second stage, the FRBS make presence, allowing to manage multiple parameters simultaneously. For the implementation of the FRBS in Java, the *jFuzzyLogic* library was used. The FRBS implemented in both algorithms is made of 5 antecedents and 1 single consequent, providing below the list of antecedents and consequent used:

Antecedents:

- MIPS: refers to the MIPS capacity for the selected VM.
- Power: refers to the power consumption done by the selected VM that run over a host for the execution process.

- Length: refers to the task length, or in other words, the million of instructions necessary for the task completion.
- Time: refers to the time required for the execution of a tasks on a specific VM/host. It is computed using the *eq. 6* specified at *section 4.6*, using for this purpose the selected VM MIPS and the task length.
- Energy: refers to the total energy (product of execution time and power) that will be made by a selected VM for the processing of the task to be executed. Its computation is done using the *eq. 5* provided at *section 4.6*, using the power consumption made by the selected VM for the task execution and the time required for its execution. Take into account that as the energy is measured in *Wh* (Watts per hour) which requires to use the execution time in hours and not in seconds.

Consequents:

- Selection: refers to the probability of selecting a VM to execute a task, where the higher this probability is, the more suitable the VM is for the task execution. It is computed by evaluating the antecedents through all the rules within the FRBS RB. Moreover, only those VMs that are available or *idle* are evaluated as candidates for their selection to execute a task. Once all the *idle* VMs have been evaluated, the one with the highest probability of selection will be chosen to execute the task.

Additionally, all the antecedents are defined with a total of 3 MFs while the consequent is defined with 5 MFs to provide a higher resolution at the selection process as the result after the antecedents evaluation. Below is provided the MF names for the antecedents and the consequent.

Antecedents MFs:

- Low
- Normal
- High

Consequent MFs:

- Very_low
- Low
- Normal
- High

- Very_high

Finally, it must be highlighted that to achieve the desired results, a proper RB must be made for the specific scenario to be tested. The results obtained through the FRBS scheduling algorithms completely rely on the quality of the RB, so that a bad RB may provide undesired results, while a good, optimal RB should easily improve the results obtained through the classic scheduling algorithms.

4.8.5. WorkflowSim with DVFS and meta-scheduling

Implemented by myself using *WorkflowSim with DVFS* as the base simulator, *WorkflowSim with DVFS and meta-scheduling* [26] simulation introduced the ability to automatically manage multiple datacenters in the same simulation scenario in a smart way through the introduction of the *meta-scheduler* entity. Before this simulator, it was possible to create scenarios with multiple datacenters/schedulers, although its behavior was basic. In *WorkflowSim with DVFS*, the WorkflowEngine entity was the one in charge of managing all the datacenters/schedulers. If a simulation scenario with two or more datacenter/scheduler was built, the WorkflowEngine would use a simple Round Robin algorithm to deliver tasks to the different available schedulers which, at the same time, schedule the received tasks among the different available resources within their associated datacenters.

With the inclusion of the meta-scheduler entity a smart tasks delivery among the multiple schedulers/datacenters can be made. The meta-scheduler entity allows to implement meta-scheduling algorithms that, like the scheduler one, provide strategies to define the strategy to follow to correctly deliver tasks among all the available schedulers/datacenters. The smart meta-scheduling provides the ability to focus the task scheduling among different schedulers/datacenters aiming to reduce execution time and the energy.

The implementation of the meta-scheduler and its capabilities required the modification/implementation of additional elements, modifying the entity structure, adding additional events as well as implementing the meta-scheduling algorithms.

4.8.5.1. Entity tree structure modification and implementation

The implementation of the meta-scheduler entity supposes, at first sight, a simple modification of the entities structure as presented at the *WorkflowSim with DVFS* simulator. In *figure 12* is shown the modification of the entities structure in a basic logic.

The simple logic defines that the entity structure must be modified as shown in *figure 12*, where between the *Engine* and the *scheduler* entity the new *meta-scheduler* entity is placed. To remember, the engine is the entity that holds all the tasks, understand the DAG and so knows the tasks that can be executed due to the restriction defined within the DAG and so provide the executable tasks to the scheduler entity to follow with the scheduling process to execute tasks. If a smart tasks delivery is desired among the engine to the different schedulers, it is necessary to implement the meta-scheduler and locate it between the engine and schedulers communication. By this way, the entity provides the new entity with the tasks that can be executed and so, through the meta-scheduling algorithms used, the new entity delivers the executable tasks among the available schedulers according to the strategy specified by the algorithm.

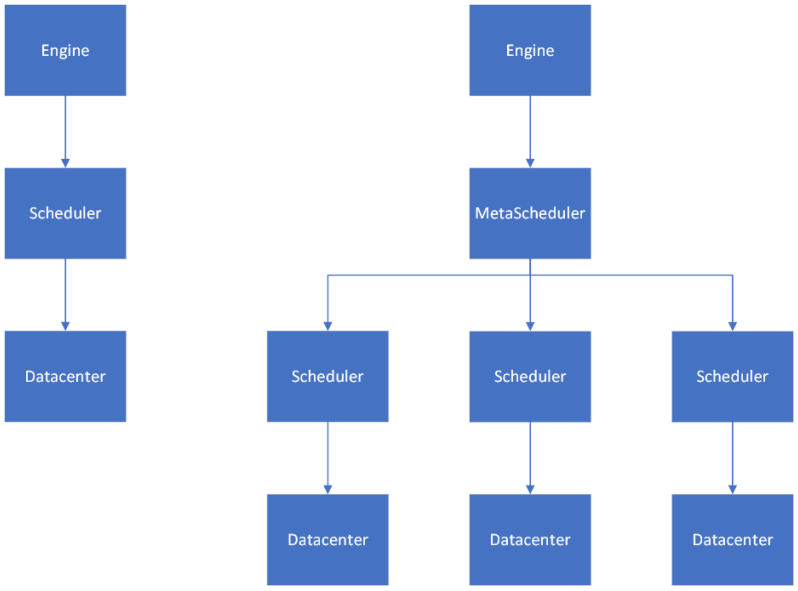

Figure 9: scheduler vs meta-scheduler structure

However, for its implementation further modification were required, modifying not only its structure in a further way than the specified, but requiring a modification of the entities to work with the new entity. The entities in *WorkflowSim with DVFS* are not prepared to work with a new entity, as its structure and specific communication between entities is not

flexible enough to easily place a new entity. For this reason, the internal behavior and implementation of the entities were modified to allow the integration of the new entity.

In *figure 13* it is shown the new entities structure for the meta-scheduler simulator, where all entities ended by the suffix "_META" refers to entities that have been modified to allow the integration of the meta-scheduler entity. All the new entities that end with the suffix were evolved from their equivalent ones in WorkflowSim with DVFS, directly modifying them to be adapted to work with the new entity. Moreover, the WorkflowMetaScheduler entity make use of the WorkflowScheduler class to evolve and implement the new class.

The new entities count with a wide variety of modifications, which are summarized next. At *section 4.8.3* was specified how the first WorkflowPlanner entity oversees the creation and instantiation of the rest of the entities. Due to the inclusion of the meta-scheduler entity, the behavior is modified to the one shown in *figure 14*, were the WorkflowEngine will no longer creates the scheduler entities, but will create the meta scheduler entity which oversees the creation of the scheduler entities.

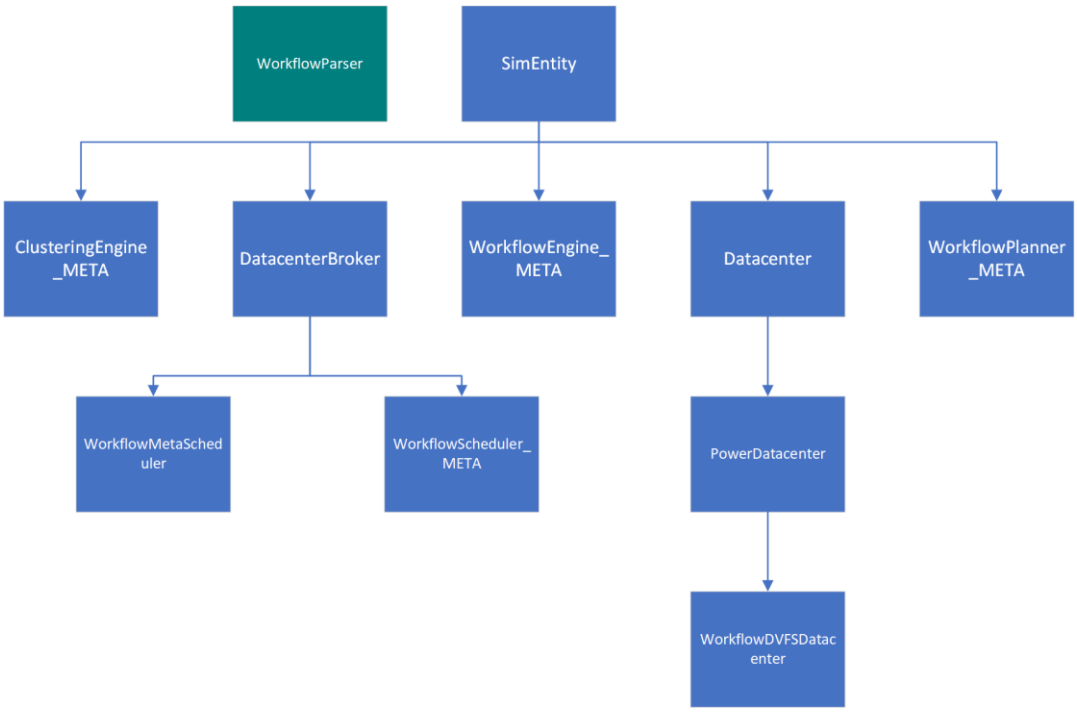

Figure 10: WorkflowSim with DVFS and meta-scheduling entities structure

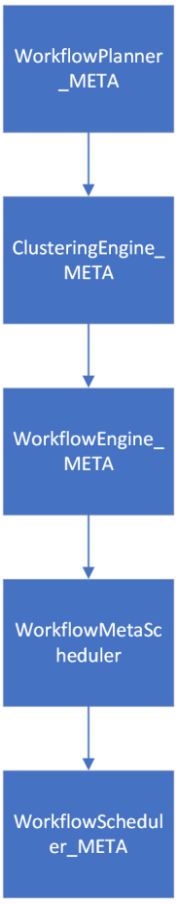

Figure 11: WorkflowSim with DVFS entities creation tree

The second most important modification is done at the WorkflowEngine, WorkflowMetaScheduler and WorkflowScheduler entities. The original behavior made the WorkflowEngine to directly communicate with the WorkflowScheduler entity, whether there were just one or more, in the scenario. Now, with the inclusion of the meta-scheduler, all the communication made between those entities must be done through the meta-scheduler entity.

The previous behavior was made possible by making the Engine entity to store a list with all the schedulers identifiers (ID) to know at each moment all the available schedulers in the scenario, and the schedulers hold the identifier for the unique Engine entity. By using both sides the ID of the other entity, the Engine was able to communicate with the schedulers and vice versa, always using the event system. Every time a scheduler needed additional tasks to be executed, it would request it to the Engine, and the Engine would send additional tasks to execute to the scheduler that requested it, only if more tasks to be executed existed. In the

same way, once a task is completed by a scheduler, the result with the task would be sent to the Engine entity.

As the meta-scheduler entity was introduced to provides a smart meta-scheduling of tasks among the different schedulers within the simulation scenario, all this process had to be done through the meta-scheduler entity. For this purpose, the Engine would no longer know either the scheduler entities in the scenario nor their identifiers (ID), which avoid this entity to directly communicate with the schedulers. At the same time, the schedulers entities will no longer have direct communication with the Engine and so will not no its identifier to avoid it. To solve it, the Engine will know the meta-scheduler identifier, and the schedulers entities will know it as well. At the same time, the meta-scheduler entity knows the identifiers for both the Engine and all the scheduler entities, which allows to establish communication between the Engine and the schedulers through the meta-scheduler. As all the tasks will be delivered from the Engine to the meta-scheduler entity, it is possible to implement smart meta-scheduling process by implementing meta-scheduling algorithms with different strategies to focus on different optimization purpose.

4.8.5.2. Additional event

Opposite to the previous implementation of the simulator, this one does not include significant modification to the event system, including just one single additional event to complement the additional entity.

The new event called CLOUDLET_SUBMIT_META use a tag = 1007, is used by the schedulers entities to request tasks to be executed to the meta-scheduler entity. This event is triggered after the scheduler entity successfully creates VMs within its associated datacenter. Once received by the meta-scheduler, it will request tasks to the Engine entity which is going to send tasks to be scheduled to the scheduler entities to the meta-scheduler.

4.8.5.3. Basic meta-scheduling algorithms

As specified, the meta-scheduling entity supposed an evolution of the scheduler entity which aims to make a meta-scheduling process to deliver tasks to the different schedulers in a smart way. For this, the scheduling algorithms were updated so that they provide the scheduling process at the meta-scheduler entity.

It is important to highlight the differences with the meta-scheduling algorithms with respects the scheduling algorithms. On the scheduling algorithms, the scheduler has complete control over its associated datacenter, for which it knows the exact resources capabilities. These resources, represented by VMs, have the capability of executing tasks and, following the strategy specified at the scheduling algorithm, the scheduler analyze for a selected task all the VMs within the datacenter and decides the optimal VM to execute the specific task. On the other hand, the meta-scheduler does not have direct control over the VMs within each datacenter, but have a general, summarized view of each datacenter characteristics. As the meta-scheduler is not aimed to substitute and replace the scheduler job, it must work only with summarized, general information of each datacenter in the scenario, for which must decide, with the summarized information, the optimal datacenter to execute a task. Once this is done, the meta-scheduler will deliver the tasks to the associated scheduler to the datacenter that must execute the task and so, the scheduler will be in charge of executing a proper and optimal scheduling process to decide the best, optimal VM to execute the task. Note that, the whole scheduling process in the meta-scheduling scenario is a lot more complex than the one found in a single scheduler scenario. For this case, both, the meta-scheduling and scheduling algorithms used must be fully optimized/optimal for the given problem. If one of both algorithms used make a bad scheduling process, the whole scheduling process could be considered as broken or not optimal, providing bad results.

Several meta-scheduling algorithms were implemented to achieve a smart scheduling process among the different schedulers/datacenters, and are divided between basics, and advanced which use FRBS.

Additionally, due to the entities structure, the meta-scheduler has no direct communication with the datacenter entities, which supposed a problem. If the meta-scheduler has no communication with the datacenters, it will not be able to know their capabilities, which will avoid making smart meta-scheduling as information about the

capabilities and the current state of the different datacenters must be known. The problem was solved by the implementation of a static class called GeneralParametersMeta, which is an evolution over the GeneralParameters static class implemented within WorkflowSim with DVFS used for the FRBS scheduling algorithm.

The GeneralParametersMeta hold information related to each datacenter and its current state, so that this information can be directly obtained by the meta-scheduler entity each time it needs it for the meta-scheduling process. Additionally, it gathered additional information that is used at the *FRBS meta-scheduling algorithm.*

Once some basic concepts related to the meta-scheduling algorithms is explained, is time to specify the algorithms that were implemented in this simulator.

RoundRobinMetaSchedulingAlgorithm: represents a basic Round Robin algorithm adapted for the meta-scheduler scenario. From the available datacenters/schedulers, the meta-scheduler delivers tasks to each scheduler following a Round Robin strategy.

MinMinMetaSchedulingAlgorithm: represents an adapted MinMin algorithm for the meta-scheduler entity. As the MinMin, it prioritizes the execution of the smallest tasks, ordering tasks from smallest to largest. At the second stage, it schedules the selected task to the datacenter that, in average, have the highest MIPS capability if there is idle VMs

MaxMinMetaSchedulingAlgorhtm: represents an adapted MaxMin algorithm for the meta-scheduler entity. At the first stage, order tasks from largest to smallest to prioritize the execution of the largest one first. At the second stage provides the same behavior specified at its equivalent MinMin version for the second stage.

4.8.5.4. FRBS based meta-scheduling algorithms

The FRBS meta-scheduling algorithms follow the objective introduced at the *WorkflowSim with DVFS* by its author, where the focus was to reduce the time, power consumption and energy due to the task scheduling. At this simulator, the power aware FRBS algorithm was adapted to work at the meta-scheduler side. For this purpose, the base algorithm used for its adaptation is the *PowerAwareFuzzyMaxMaxSchedulingAlgorithm*, explained at *section 4.8.4.4*.

The new algorithm called *PowerAwareFuzzyMaxMaxMetaSchedulingAlgoritm* is an adaptation of the previously mentioned algorithm to work at the meta-scheduler entity. The main difference with respects its original implementation for the scheduler entities is the inclusion of an additional antecedent, and so, the total number of antecedents pass from 5 to 6, maintaining a single consequent as before.

The additional antecedent represents the *utilization* of the datacenter. At the meta-scheduler side, it is not enough to know the MIPS capabilities and the power consumption of datacenters, and so, if it is desired to avoid wrong scheduling, it is mandatory to know at any moment each datacenter utilization. Task scheduling to datacenters that are full or close to full load must be avoided, as it would make tasks arriving to these datacenters to wait for their scheduling if there is no idle resources, for which their execution would be delayed. Moreover, if a large task is delivered to a datacenter where only the slowest VMs are idle, the task will be forcibly scheduled to be processed on a slow one, which could significantly delay the completion of the whole tasks to be executed, especially if the task length scheduled for its execution is high.

Moreover, although the other 5 antecedents and the single consequent used are the same as specified at *section 4.8.4.4*, their use differ from its original implementation. Below is, once more, provided the antecedents with their purpose and behavior for the meta-scheduler entity:

Antecedents:

- MIPS: refers to the average MIPS capacity of the selected datacenter. For its computation, the MIPS capacity of each VM is sum and divided by the number of VMs at the datacenter.
- Power: refers to the average power consumption of the selected datacenter. Its computation is done like the MIPS antecedent, but obtaining the power consumption of each VM rather than its MIPS capacity.
- Length: same as the length parameter specified at *section 4.8.4.4*.
- Time: refers to the estimated average time required to execute the selected task at the selected datacenter, using the *eq. 6* specified at *section 4.6*, where the MIPS parameter refers to the datacenter average MIPS capacity.
- Energy: refers to the estimated average energy (product of the execution time and power) for the execution of the selected task at the selected datacenter, using the *eq.*

5 specified at *section 4.6*, where the power parameter refers to the datacenter average power consumption.

Consequent:

- Selection: at the meta-scheduler entity, the selection probability does not specify the probability of selection for a VM at a datacenter but refers to the probability of selecting a datacenter as the possible candidate to execute the selected task. Once a datacenter has been selected to process a task, the task is provided to the associated scheduler to the selected datacenter that must execute this one for its subsequent local scheduling.

Additionally, the MFs defined for the antecedents and the consequent are the same as defined at *section 4.8.4.4* for the FRBS scheduling algorithm.

Finally, it must be pointed that, as every FRBS system, for its optimal behavior it is mandatory to create or obtain a properly optimized RB for the scenario where it is used, otherwise, the results obtained may be suboptimal and could worsen the results obtained through the classic/basic algorithms.

Moreover, at this stage, the RB was manually defined which is not the best approach to obtain it. However, it was possible to achieve better results that the provided by the meta-scheduling algorithms based on the classic scheduling strategies.

5. PROJECT DEVELOPMENT

The development of the project has been divided in two parts, the first part which corresponds to major code cleanups as well as modifications to the code to ease its use and understanding, and the second part, where the objectives specified at *section 3* that have been implemented are explained.

Moreover, the development of this project consists on improving the simulators explained at *section 4.8*, specifically the simulator previously done at my bachelor thesis as explained in *section 4.8.5*, for which this specific simulator is used as the source to implement the new capabilities and achieve the desired results.

At the code clean up section/part several modifications have been done to try and ease the understanding and complexity of the simulator. First, at the development of the *WorkflowSim with DVFS and meta-scheduling*, the original WorkflowSim and WorkflowSim entities were simply copied into a new Java package and directly modified to achieve the desired behavior. Although it made up a fully functional simulator where the desired behavior and functionalities where achieved, it was code inefficient. It has been modified to follow the implementation found with previous simulators, where newer versions make use of the Java inheritance to add additional functionalities to new classes. Following the Java inheritance, the entities implemented at the last simulator version have been reimplemented once more using as the base code the entities introduced at WorkflowSim and WorkflowSim with DVFS, directly inheriting from them, creating a newer and lighter version of the new entities introduced at the *WorkflowSim with DVFS and meta-scheduling simulation*.

At the same time, due to the modification done to the entity structure, the meta-scheduling algorithm have been adapted to this new entity structure. Additionally, the previous meta-scheduling algorithm implementation relied on a static Java class called *GeneralParametersMeta* which store some basic information needed for the scheduling process. Again, although this previous implementation was fully functional, it significantly raised the complexity of the simulator, requiring a high effort to understand the simulator. Due to this, additional steps have been done to remove the use of this static class for the basic meta-scheduling algorithms, which are those that does not use the FRBS.

On the other hand, plenty of modifications and implementations to improve the simulator have been done.

First, an additional planning method has been implemented at the simulator. On previous version, the *on-batch* planning was the only planning method implemented, and so, to extends the simulator capabilities, the *on-line* planning method has been implemented.

Second, the FRBS meta-scheduling algorithms capabilities have been extended. Up to this point, the previous versions used an old implementation of the *jFuzzyLogic* library that implements the FRBS into Java. This old implementation had some restriction as neither allowing the use of negated terms nor the use of the OR connector. To solve this, a new version of this library has been implemented which allows the use of the negated and non-negated terms as well as the AND and OR connectors.

Third, a new simulator power/energy approach has been implemented which allows the consideration of renewable power/energy sources that supply the datacenters. This new renewable energy scenario is implemented through two different approaches, one that implements batteries that store renewable energy at each datacenter and other that consider that the unlimited power supply at each datacenter consist of a certain amount of renewable energy and non-renewable energy. The new scenario allows the study of additional scenarios where some datacenters may or may not have renewable energy either from their attached batteries or through the unlimited power source, with the possibilities of creating more complex, advanced meta-scheduling algorithms that prioritize the scheduling of tasks to those datacenters that have renewable energy available.

Finally, to ease the complex process of optimal RBs computation, a few knowledge acquisition systems have been implemented. The implemented algorithms are the *Pittsburgh*, *Kasia* and *Pareto* over *Pittsburgh* which provide different approaches/solutions for finding optimal RBs under different scenarios. Thanks to these algorithms, it is possible to intelligently and easily adapt the RBs used at the complex meta-scheduling algorithms to optimize the execution of tasks on different scenarios.

5.1. WorkflowSim with DVFS and meta-scheduling code cleanups

At the first stage of development of this project, before beginning with the development of new capabilities for the simulator, it was decided to properly analyze the previous work done aiming to detect if the previous work should be modified or enhanced to provide a clearer, easier to understand and use code and simulator.

After a first analysis, it was identified that some parts of the previous work and code were messy, confusing and untidy, for which was decided to put a solution. It was identified that some modifications where done to the simulator that could break the backward compatibility as well as a heavy entity structure modification that broke the inheritance/extends logic among different classes. Moreover, it was detected that static classes were used to ease handling with information exchange among classes aiming to ease its use and understanding, procedure that was used even in the most simple scenarios were its use created the opposite result, making the code understanding even more complex than if it were not used.

Due to all this, it was decided to dedicate a part of the development of this project for code cleanups and enhancing the old code.

Moreover, all the work developed at my bachelor thesis was neither correctly documented nor securely saved. Due to this, it was decided to redo all the work developed again to securely store it as well as correctly document all the modifications done to the *WorkflowSim with DVFS* to achieve the *WorkflowSim with DVFS and meta-scheduling* state. To do this, all the work was replicated step by step, uploading it commit by commit to a GitHub repository where at each commit was correctly and in-depth explained all the modifications done to finally implement the meta-scheduler simulator scenario.

Finally, not only all the *WorkflowSim with DVFS and meta-scheduling* redo process is uploaded to GitHub, but also all the cleanups process as well as all the new implementation specified at *section 5.2*.

5.1.1. Entities code cleanups

The first stage for the simulator cleanups consists in a complete entity redone. As specified at *section 4.8.5*, the entities for the meta-scheduling simulator where not done by using inheritance from the entities implemented in *WorkflowSim with DVFS*, but the entities implemented in this simulator were copied with a different name and directly modified to implement the meta-scheduler structure.

As this work broke the entities inheritance implementation logic, it was decided to redo all this work to create a better, easier to understand structure using the benefits of Java inheritance. For this purpose, the original entity structure implemented in WorkflowSim with DVFS has been taken and evolve to create the new required entities as shown in *figure 15*.

To achieve this, the original entities from where the newer ones inherit were modified to allow the inheritance, as the classes were declared as *final* with some attributes and methods declared as *private*, which avoided its use to inherit from them. However, in spite of these modifications, the behavior of the original entities was not modified and so the backward compatibility with the original version is maintained.

Additionally, other minor modifications were done to reestablish the CloudSim java class to its original state, as it was modified on the development of the simulator to allow the correct behavior for the meta-scheduling algorithms. The current simulator implementation does not need any longer to modify the CloudSim core to make the meta-scheduling algorithms to work, and the modifications done to achieve this state are explained at *section 5.1.2*.

Thanks to this entity structure modification, a lot of major cleanups has been achieved. As it uses the inheritance, it has been required to only implement at each new entity the bare minimum attributes and methods necessary to make it work, while some other methods have been overridden by newer ones adapted to the new scenario.

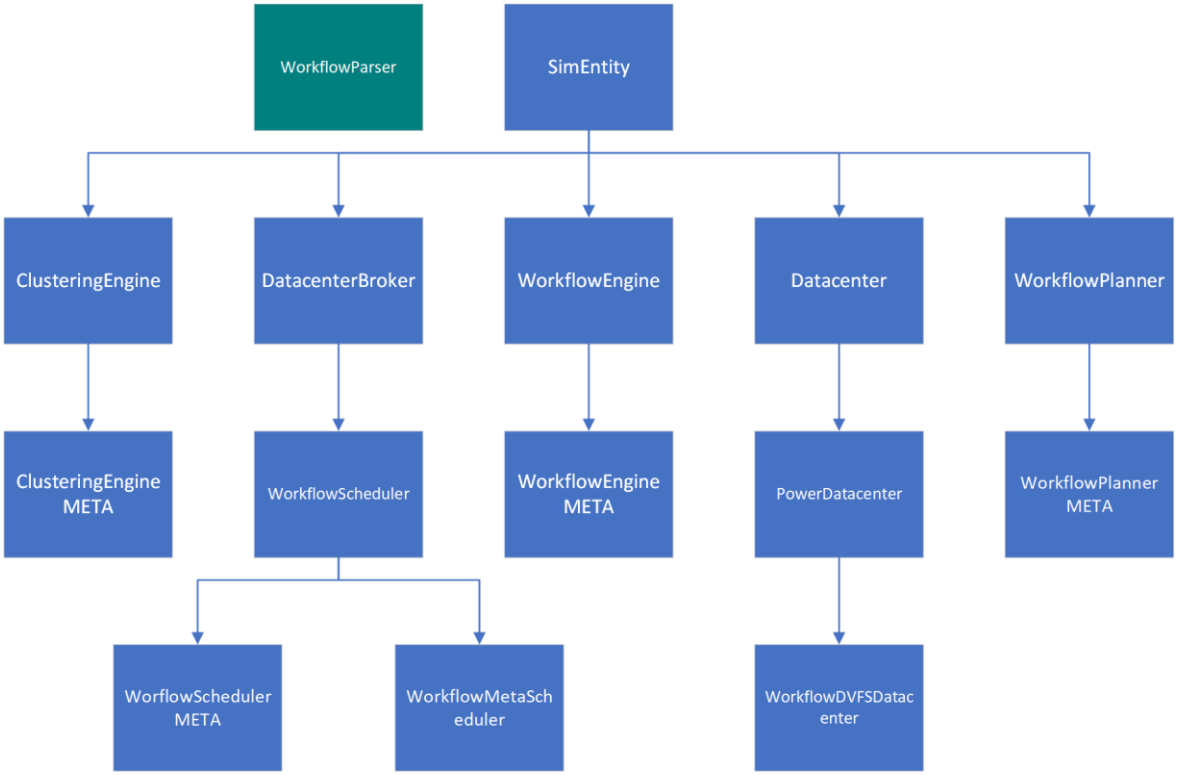

Figure 12: WorkflowSim with DVFS and meta-scheduler entities redone

Finally, to improve the entities storing structure, a new java package is created for the specific new entities called org.workflowsim.dvfs.meta, which helps to maintain every simulator implementation entity ordered and easy to locate. Not only that, but also the old meta-scheduler entities are relocated at the new package.

5.1.2. Meta-scheduling algorithms adaptation to the new entities

The creation of the new entity structure and entities required some minor modification to the meta-scheduling algorithms due to the change on the name of the entities. As before, to maintain a backward compatibility, rather than modifying the meta-scheduling algorithms, new ones has been created and adapted to be used by the new entities. Just as has been done with the new entities' implementation, the old algorithms are renamed with the suffix "Old". Additionally, to better locate and store the meta-scheduling algorithms, a new package called org.workflowsim.scheduling.meta is created to separate the meta-scheduling from the scheduling algorithms.

The modification and adaptation of the algorithms to the new entity structure does not alter the simulator behavior at all, obtaining the same simulation results for the execution of the same scenario at both, the old and the new implementation.

5.1.3. Static class dependency removal

The meta-scheduling algorithms, for its implementation, made use of a static class called GeneralParametersMeta as specified at [26] and *section 4.8.5.3*. This static class, even though allowed the meta-scheduling entity to have a direct access of the datacenter for the meta-scheduling process, increased significantly its understanding and configuration/programming. Despite of working correctly, it was detected that the use of the static class could be removed from the basic meta-scheduling algorithms and so ease its configuration and understanding, not needing an external class to access to the datacenter information anymore. However, to achieve it, a wide study and modifications were done.

In the old implementation, scheduling algorithms directly relied on the static class to obtain datacenter information state and characteristics as the power consumption and the performance through the host/VM MIPS. Due to this, the meta-scheduling behavior was achieved by implementing it directly between the CloudSim core, the static class, and each of the algorithms. The first, main problem, implied the direct modification of the CloudSim core, which partially broke the backward compatibility with older simulators and a workaround was introduced to bypass the problem introduced at the core simulator when the meta-scheduling scenario was not use as stated at [26]. As the CloudSim core posses' control over all the created entities and managed them and their events, a portion of code was implemented on the *runClockTick* method to copy to the static class the current datacenters state. The *runClockTick* method refers to a CloudSim method that is executed continuously after the whole processing of events. If the CloudSim core has more events to be processed, it will execute itself continuously to process the remaining events, checking from the first to the last entity if there is event for each of them. The workaround to solve the problem consisted on implementing a simple null check at the newly introduced code where it was checked if the static class list that stored the datacenter IDs was set, and, in case of null, it skipped the process of searching for the datacenters entities by their IDs to create a copy of them to the static class.

Although functional, it was desired to suppress this old code as well as the static class dependency in the greatest degree possible. After some in-depth study, it was analyzed that several changes had to be done to suppress this dependency and remove the necessity of altering the CloudSim core code. On the one hand, it was detected that the CloudSim class is a static class by itself. Thanks to this, the necessity of altering its code was no longer needed, as it was possible to call this class from any other class to retrieve the datacenters entities and so remove the dependency with the static class or, at least, remove it partially.

Secondly, a deep modification to the meta-scheduling algorithm implementation was required to be done. In order to ease the implementation and to automatize the process of accessing to the datacenters information, it was decided that, rather than implementing redundant code on every meta-scheduling algorithm, detecting some common part of the code and making use of the java inheritance was more convenient.

To understand the modifications made, it is required to make further explanation of the way the scheduling and meta-scheduling are implementing. As well as occurs with the entities, the scheduling algorithms inherit from a common parent class that implements all the common parameters and methods that are shared among the different algorithms. The parent scheduling class from where all the scheduling and meta-scheduling algorithms originally inherit is called BaseSchedulingAlgorithm. In *figure 16* is shown the algorithms structure. Note that only a few algorithms are shown in the figure.

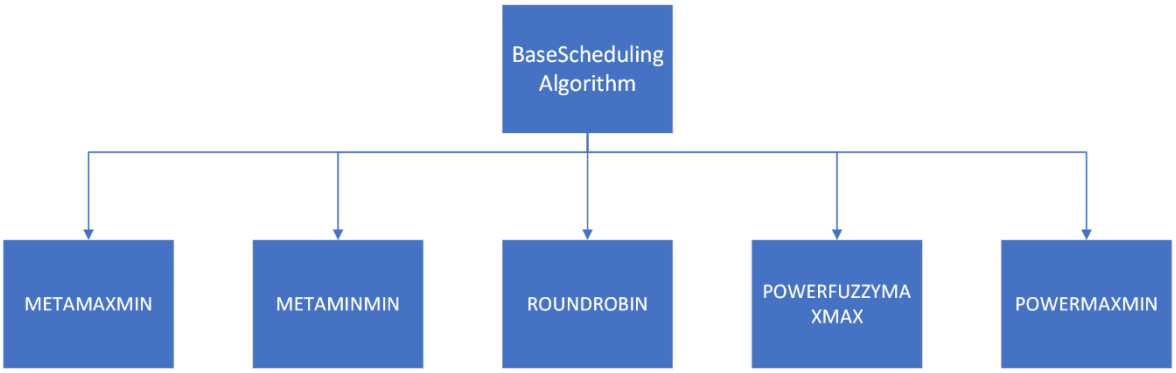

Figure 13: algorithms inheritance

Schedulers and meta-schedulers do not directly use the desired algorithm to be used for the scheduling or meta-scheduling process but create a BaseSchedulingAlgorithm object that extends its functionalities to the desired algorithm to be used. This process allows to establish a common functionality among all the algorithms, which allows to define at the schedulers

and meta-scheduler a single way of managing/using the algorithms through the base algorithm that extends its functionalities to behave as the desired algorithm. Due to the desired suppression of the GeneralParametersMeta static class, it was necessary to extends the base scheduling algorithm functionalities which would probably affect the backward compatibility, which as specified before, is a not desired scenario. For this reason, a new base scheduling algorithm called the BaseMetaSchedulingAlgorithm was created to properly separate the scheduling from the meta-scheduling algorithms.

The new BaseMetaSchedulingAlgorithm is used by the new implementation of the meta-scheduling algorithms to inherit common parameters. The parameters implemented within this base meta-scheduling algorithm are a list of datacenters called *datacenterList*, a *Map* called *bindedDatacenterToScheduler*, and a *list of lists of jobs* called *scheduledListMeta*.

The *datacenterList* is used to store a copy of the datacenters within the simulation scenario to provide the *scheduling* algorithm with the current state for each datacenter at the time of making the meta-scheduling process.

The *bindedDatacenterToScheduler Map* is used to know the relation among the datacenters and their associated schedulers. This *Map* stores the ID of each datacenter attached with the ID of its associated scheduler. At the process of scheduling, it is only required to know the ID of the scheduler that must receive the tasks for the desired datacenter that must execute the task. The ID is used to send event with the associated Job/Task/Cloudlet to be scheduled.

The *scheduledListMeta* consists in a *list of lists of Jobs*. During the meta-scheduling process, once the meta-scheduler decides through the used algorithm the datacenter that must execute a specific task will store at the associated list to its associated scheduler the task that must be executed, process that is repeated as long as there are available tasks to be meta-scheduled. This process allows to send tasks to each datacenter for its execution in the on-batch strategy.

Once the BaseMetaSchedulingAlgorithm class was implemented, all the meta-scheduling algorithm were modified to inherit from this class and, as well, the meta-scheduler entity was modified to use the BaseMetaSchedulingAlgorithm rather than the old one. Thanks to this, the meta-scheduling algorithms can obtain the required datacenter information from the class that they inherit, only remaining the way to initialize these parameters from the meta-scheduler entity.

As the CloudSim entity was discovered to be a static class that can be called from any entity or other java class, it was possible to access to it at the meta-scheduler entity at the process of meta-scheduling tasks, which allows to take the datacenter entities state and initialize the previously specified lists attributes at the new base scheduling algorithm. However, to retrieve the datacenters information from the CloudSim core it is necessary to know the datacenter IDs. Moreover, the it is not only required to assign the datacenter information to the base scheduling algorithm, but it must be also assigned/provided the *bindedDatacenterToScheduler Map* information. To do this, the *Map* was also implemented at the meta-scheduler entity as well as the *datacenterList.* Additionally, as it is necessary to know the datacenter IDs to obtain the datacenters from the CloudSim direct access, a *datacenterIds list* was also implemented at the meta-scheduler entity.

Once this was done, before providing the BaseMetaSchedulingAlgorithm with the required data from the meta-scheduler entity, it is required to initialize the two new attributes, *Map* and *datacenterList* with the required information. At the creation of the simulation scenario where all the different entities must be manually defined and created, entities information can be retrieved and, at the same time, their attributes can be modified and initialized. Thanks to it, the *bindedDatacenterToScheduler Map* and the *datacenterIds list* within the meta-scheduler list can be initialized at the meta-scheduling process. By using the *datacenterIds list*, the datacenters current state can be obtained by directly accessing to the CloudSim entity list and identify through the datanceter IDs the datacenter entities. Once this is done, before executing the meta-scheduling algorithm, the BaseMetaSchedulingAlgorithm is provided with the *datacenter* current state as well as the *bindedDatacenterToScheduler Map* which is required to allow the algorithm to, once identified the datacenter that must execute a task or group of tasks, provide to its associated scheduler the list of tasks that must be later scheduled within the available VM inside the datacenter for its execution.

Thanks to these modifications, it was possible to successfully remove the GeneralParametersMeta static class dependency from the basic meta-scheduling algorithms, and so, RoundRobin, MaxMin and MinMin, no longer need its use to work. However, while trying to remove it from the FRBS meta-scheduling algorithm several problems were found that make it not possible to remove its dependency. The reason for which it is not possible to avoid its use is due to the DVFS. As specified, the DVFS alter both, the CPU frequency and voltage. As the CPU performance (MIPS) and the direct power consumption rely on

these parameters and the DVFS change them along time, it creates not predictable performance and power consumption changes at the datacenters that severely affect the FRBS system. To avoid this result, the best solution found was to keep using the GeneralParametersMeta static class, so that the same values are used to normalize the MIPS, power, energy and time parameters and prevents values out of the normalized range which creates error at the meta-scheduling process at the FRBS.

5.2. WorkflowSim with DVFS and meta-scheduling enhancements

After the major code cleanups and initial modifications to ease the simulator use and understanding, several modifications and additions were done to improve the simulator capabilities. Among these modifications, it can be found both, modifications done to add additional capabilities such as the on-line planning, and others done to enhance capabilities that were already implemented, like those aimed to improve the FRBS implementation within the simulator. Moreover, to additionally improve the simulator capabilities, external knowledge acquisition systems has been implemented to achieve an automated RB generation adapted to the simulation scenario used, aiming to obtain optimal or close to optimal results easily and simplifying users work.

5.2.1. On-line planning

As explained *at section 4.2.1*, cloud systems counts with a high variety of planning strategies to follow. At this simulator, since the WorkflowSim version, the unique and only planning strategy implemented is the on-batch strategy. This strategy implemented seems to be the smart choice and best solution to be adopted, as thanks to it, tasks will not be scheduled individually, but in groups, which helps schedulers and meta-schedulers to achieve better results due to the tasks execution. If tasks are scheduled individually, undesired results can be obtained, as a small task can be delivered to a powerful resource and so, once a big task arrive, it may be delivered to a weak, slow resource as all the powerful resources may be already busy executing tasks, which can easily severely delay the execution time for the big task. Thanks to the on-batch strategy, as tasks as scheduled as a whole, this undesired

behavior is easily avoided. However, the on-batch strategy is far from idoneal in certain situations. In on-batch strategy, tasks are not delivered after one of two different events occurs. This strategy waits a certain amount of time to receive the highest amount possible of tasks at either the meta-scheduler or the scheduler scenario before scheduling tasks or, on the other hand, waits for a certain number of tasks to be received to perform the scheduling process. Either of these two different scenarios aims to analyze tasks for its schedule to resources simultaneously to avoid the previously described problem. Now, if the task arrival rate is so low, a task arriving may take a significant big amount of time until it is finally delivered to a resource to be processed due to the necessity of waiting a certain amount of time or waiting for a number of tasks to be receive. This scenario can overall delay the tasks execution, which can be easily solve. For this case, if tasks are scheduled as soon as they arrive, the total execution time required to finish a task can be significantly reduced. This behavior is known as on-line strategy, and although in normal scenarios where the arrival rate is moderate to high can easily worsen results compared to the on-batch strategy, is worth considering its use for the previously described scenario.

As the on-line strategy is not implemented within the simulator in any of its version, actions to change this situation were done. At first, a wide study of the simulator was done to find the proper way to implement it, followed by its implementation. Its important to understand that, as the simulator works by events, tasks are sent as attached data inside the specific events to make the schedulers and meta-scheduler entities to execute the scheduling process. In the on-batch scenario, schedulers receive tasks in groups from the meta-scheduler entity. The tasks are received in one single event, from where the scheduler takes all of them and proceed to schedule them according to the algorithm used.

To implement the on-line strategy, a simple and single action was required. If tasks are delivered from the meta-scheduler to the schedulers individually rather than in groups, the schedulers are going to receive the tasks individually and so will schedule locally at their datacenters individually, which creates the on-line strategy. For this, it was required to allow to implement the possibility at the meta-scheduler entity to send tasks one by one, creating an event for each task to be sent.

For its implementation, only the meta-scheduler entity was modified. At the method where the meta-scheduling process is done, if the on-line strategy is selected to be used, it will take just the first task from the list of tasks to be meta-scheduled and will be passed to

the meta-scheduling algorithm for its schedule. On the other hand, if it is set to use the on-batch strategy, all the tasks available at the list of tasks to be meta-scheduled at the meta-scheduler will be passed to the algorithm to schedule them among the different available schedulers.

5.2.2. FRBS negated terms and AND-OR terms connectors

Since WorkflowSim with DVFS, the simulator has counted with FRBS algorithms that provides advanced scheduling process and the ability to obtain better simulation results. Although the FRBS implementation is functional, further steps could be done to improve it. The FRBS implementation only allowed to use the AND connectors between terms and neither the OR connector nor negated terms. The main reason for using and implementing the OR connector and negated terms is the possibility to achieve even more advanced scheduling, and not only that, but the possibility to achieve more efficient RBs. This new FRBS options allows to obtain smaller RBs, made of less rules that provides at least the same results as obtained before on even better. Due to the benefits that this can provide, it was decided to implement it. Note that most of the code adapted to support the OR terms connector and the negated terms were done aiming the future utilization and connection of the Java simulator with *knowledge acquisition systems* implemented in MATLAB as shown in *section 5.2.4*.

For the implementation, as always, one of the main objectives is to maintain the backward compatibility with previous versions, for which new files needed to be created. The first problem encountered at its creation was that the jFuzzyLogic library included in previous versions did not support either negated terms or the OR term connector, although it was easily fixed by simply replacing the used library for an updated version. The second problem is related to the original implementation. The code provided in the original version was not made/adapted to support neither the OR connector nor the negated terms, and so, some modifications has been done to achieve a working implementation.

First, it is necessary to understand how the FRBS is structured within the simulator through *jFuzzyLogic*. The main component in a FRBS implemented through the jFuzzyLogic is the *FCL* (Fuzzy Control Language). On the FCL file is where the main system is declared. It is made of a *function block* where the whole system is declared, which consists on the

input variables, output variables and the rule block. It is important to understand that up to this point, if the whole FRBS is declared into de *FCL* file there is no need to modify any code on the simulator to implement the negated terms and the OR connectors, and just by simple updating the jFuzzyLogic library is enough for its correct implementation.

At [27], the authors of the jFuzzyLogic library provides a wide guide about its utilization and the required concepts for its understanding, although to clarify it, some basic information is added below about the composition of an FCL.

- Input variables: declared as VAR_INPUT, encompassed just by name all the input variable (antecedents) that will make up the whole system
- Output variables: declared as VAR_OUTPUT, encompassed just by name all the output variable (consequents) that will make up the whole system
- Antecedents: declared as FUZZIFY, is used to declare each antecedent at the system specifying and defying each of its MFs. Note, a FUZZIFY must be created for each antecedent.
- Consequents: declared as DEFUZZIFY, is used to declare each consequent and as well as the antecedents, it is necessary to specify each of its MFs. Note, a DEFUZZIFY must be created for each consequent.
- Rule base: declared as RULEBLOCK, is used to declare the whole rule base that provide the intelligence to the FRBS. The rule base or RULEBLOCK is made of multiple rules declared within it as RULE.

It is important to understand that it is not necessarily required to declared within the FCL file the RB to be used, as it can be dynamically generated by through Java code. This dynamic generation is what requires further implementation to support the OR connector and the negated terms. The main objective of generating RBs outside of the FCL is to provide the ability to connect the simulator to a *knowledge acquisition system* that generates RBs seeking to create optimal ones adapted to the simulation scenario.

On previous version of the simulator, code for this purpose was provided, although, as specified, it did not support the OR connector and negated terms and requires to be adapted.

At its original implementation, an RB or Rule block was dynamically created by using three different classes for its definition, which are the *FuzzyEntity*, *FuzzyRule* and *FuzzyVariable*.

- FuzzyEntity: is what define the whole RB and is made of a list of rules defined by the *FuzzyRule* class objects.
- FuzzyRule: define a rule that is made of a list of antecedents and a consequent, both made of *FuzzyVariable* objects.
- FuzzyVariable: used to define either an antecedent or a consequent. It provides two attributes, *name*, to specify the antecedent or consequent name, and *weight*, to specify the antecedente/consequent weight, representing the weight the MF specified.

These classes provided the necessary tools to create an RB dynamically, although, as seen, it did not include any way to specify either the connector or if the term is negated or not. These classes were later used at another different class that contains a decodeJSONRules created by the author to create from a JSON string the rules dynamically. At this decodeJSONRules method, all the rules are directly connected with the AND connector and considering non-negated terms.

To implement the negated terms and the OR connectors, it was necessary to, in the first hand, modify the 3 classes used to define RB, and, in the second hand, modify the decodeJSONRules method. As it is desired to preserve backwards compatibility, rather than modifying the three classes to define the RB, newer ones are created based on those and as well, rather than modifying the class that hold the decodeJSONRules method to use the new classes called WorkflowSimTasksFUZZY, a new one was created base on this last and adapted to use the new classes. Below, the new classes to define the RB are specified.

- ExtendedFuzzyEntity: based on FuzzyEntity class, provide the same behavior but using the new *ExtendedFuzzyRule*.
- ExtendedFuzzyRule: based on FuzzyRule class, provide the same behavior, but using the new *ExtendedFuzzyVariable*.
- ExtendedFuzzyVariable: based on FuzzyVariable, contains modifications to allow specifying the negated term through a Boolean attribute called *negatedTerm* and another integer attribute called *connection* to specify whether the AND (attribute set to 1) connector or the OR (attribute set to 2) connector must be used.

On the other hand, a new class called *WorkflowSimMetaRenewableFUZZY* based on the *WorkflowSimTasksFUZZY* which contains an adapted version of the decodeJSONRules. The whole class is adapted to make use of the new classes that defines the RB. Additionally, the decodeJSONRules method is modified to test the *negatedTerm* and *connection* attributes

implemented in the new ExtendedFuzzyVariable class to create the new RB with the possibility of using the negated terms and the AND or OR connectors according to what is specified within the JSON string provided to the method to be decoded into a RB.

As the final step, the PowerAwareFuzzyMaxMaxMetaSchedulingAlgorithm is updated to make use of the new classes to allow the final implementation and use of the negated terms and OR connector. Code to test and obtain whether a term is negated and if the OR or AND connector is to be used to connects terms within a rule.

5.2.3. Renewable energy

As specified at *section 2* and [3][4], the energy and power consumption made by cloud systems and datacenters can be considered as high and experience a yearly increase of its consumption due the continuous raising demand. As specified, one of the main objectives of this master thesis is to provide a simulator that allows to properly study ways to reduce the energy and power consumption done by datacenters through the implementation of advanced scheduling algorithms. On the way to providing a better simulator, it has been considered the implementation of a renewable power/energy source to electrically supply the datacenters used. As renewable power source causes less harm to the planet, it is recommended to raise or prioritize its use against non-renewable power source.

The objective of adding the renewable power source to the simulator is no other than providing a tool to create even more advanced scheduling algorithms. In a scenario where a datacenter has access to a renewable power source, whether it is limited or not, the scheduling of tasks should be prioritized to deliver tasks for its execution to those datacenters that have renewable power source against others that does not have it.

Further steps were done to increase the simulator capabilities. To start, two different renewable energy models are implemented at the simulator, one based on the use of batteries installed at each datacenter that holds renewable energy and another where it is supposed that the renewable energy is directly provided as a unlimited power supply along with no renewable energy, specifying the proportion of renewable energy that is provided to the datacenter power source.

5.2.3.1. Battery implementation

As already specified, the renewable battery implementation consists on the implementation of batteries on each datacenter that will be charged with an only renewable energy. This creates a scenario were those datacenters that have batteries implemented possess a limited renewable energy to be used to supply their resources and, by this way, avoid using non-renewable energy source/supply. This new scenario supposed at the same time that every datacenter that have a battery is also connected to an unlimited, non-renewable power supply that can be used once the battery is fully discharged to continue powering the datacenter resources, allowing its correct functioning and uninterrupted tasks execution. At the same time, datacenters with no batteries are directly connected to non-renewable power source, so that their correct behavior is guaranteed.

Once decided how it had to be implemented, it was necessary to find the proper way to do it. To remember, the energy and power consumption estimation was implemented back at the *CloudSim with DVFS* simulator as specified at *section 4.8.2* and so, it is at these classes where the implementation of the batteries must be done.

After some study, it was found that the power consumption is computed within the updateCloudetProcessingWithoutSchedulingFutureEventsForce method within the PowerDatacenter entity, which corresponds to the datacenter entity implementation at the *CloudSim with DVFS* simulator, and so, it is here where the modifications should be done to implement the batteries. As always, to preserve backward compatibility, directly modifying older simulator classes is completely forbidden and so a different approach was done. To solve it, a new package was created called org.workflowsim.dvfs.meta.renewable where a new entity has been created called WorkflowSimDVFSDatacenterRenewable, which inherit from the WorkflowDVFSDatacenter created at the WorkflowSim with DVFS simulator, which, at the same time, inherit from the *PowerDatacenter* entity implemented at the *CloudSim with DVFS* simulator. Due to this inheritance process, it is guaranteed that all the methods necessary to provide a correct datacenter entity behavior are implemented. As only the previously specified method is necessary to be modified, at the new entity this method was created overriding the original one implemented at the *PowerDatacenter*, allowing to securely implements the modifications required to achieve the implementation of the renewable batteries.

Additionally, five additional attributes are implemented at this new datacenter class for the battery implementation, which are specified below:

- mCapacity: the maximum battery capacity. The value specified corresponds to energy in Wh (watts/hour)
- iCharge: the initial battery charge, specified in percentage (0 to 1)
- cCharge: the current battery charge. Contains the current capacity specified in Wh
- cpCharge: the current battery charge in percentage. The value stored will be between 0 and 1. Implemented in this way to provide a normalized value to be used if desired at a FRBS meta-scheduling algorithm.
- nrenewPower: refers to the non-renewable power consumption. Attribute aimed to store the power consumption once batteries are exhausted.

At the same time, all the new attributes have their own methods to modify and obtain their current values.

Its implementation within the specified method is done in the following way. Each time this method is called is due to the processing of tasks and so, as it is here where the power consumption is computed, it is necessary to compute ways to discharge the battery and so, once the batteries are discharged proceed to record all the non-renewable power consumption. Considering this, its implementation is made of 3 simple conditionals.

The first conditional test if the battery is fully discharged, and if fulfilled, all the power consumption done by the execution of tasks is recorded on the *nrenewPower* attribute.

The second conditional is the most complex one. Only tested if the first conditional is not fulfilled, it tests if after subtracting to the current battery charge the power consumption done, the resulting value is below 0. If this fulfill, it means that, although previously the battery had charge, the power consumption done is higher than the energy stored. In this case, it sets the battery charge to 0 and compute the difference between the battery charge and the consumption done to add the new consumption to the *nrenewPower*.

The last conditional, which is only triggered if neither of the two previous conditionals are fulfilled, simple subtract the power consumption done to the current battery charge.

Note that, for this process, a variable implemented called *timeFrameDatacenterEnergy* has been used to compute the battery charge substraction and the non-renewable power

consumption. This variable stores the whole energy used by all the datacenter resources in the last interval of time since the previous execution.

5.2.3.2. *Renewable power supply implementation*

Renewable power supply supposes an alternative implementation to renewable energy implementation within the simulator. Unlike the battery implementation, this one makes no use of batteries to store a limited renewable power source but implement an unlimited renewable power source. This unlimited renewable power source is directly provided along the non-renewable power source, supposing that the datacenter is powered by both power sources at the same time. Both are differentiated by a renewable power proportion implemented that specified at each datacenter the amount of power source/supply that corresponds to renewable power supply.

For its implementation, the previously created datacenter entity has been updated with two additional attributes as shown below:

- renewPower: the renewable power consumption. This attribute is used to store the quantity of renewable power that has been used from the unlimited power source. The objective is to implement a way to differentiate the renewable power consumption done from the unlimited power source from the one made from the renewable batteries.
- cARenewPSourceP: the current alternative renewable power source proportion. It is aimed to store the current proportion of renewable energy in the unlimited power supply. This parameter is specified in a percentual way from 0 to 1 to ease its later use at the advanced *FRBS meta-scheduling algorithms*.

It is important to point that the new renewable implementation is implemented providing complete compatibility with the previous method so that it is possible to use the new renewable power supply along with the batteries.

Moreover, additional modifications have been done within the method updateCloudetProcessingWithoutSchedulingFutureEventsForce to make use of this new attributes and compute the renewable energy and power consumption done from the unlimited power supply.

As specified before, the battery implementation is made with three simple conditions. The conditional inner code/behavior is modified to make use of the renewable proportion. The last conditional is not modified, as it is used when the battery has renewable charge to obtain the power to supply the datacenters directly from it.

The first conditional which is used once the battery is discharged is now modified to store the renewable energy and non-renewable energy consumption. At the renewable energy consumption, the power consumption is multiplied by the renewable proportion to store the amount of renewable energy used while at the non-renewable energy consumption attribute the power consumption done is multiplied by $(1 - renewProportion)$ to store the amount of power consumption used that corresponds to the non-renewable part.

The second conditional is modified as well. Once the remaining battery energy is subtracted, the remaining power consumption is divided between renewable and non-renewable and stored in its attributes as explained before.

5.2.3.1.1. Power source models

The renewable energy is obtained through solar, wind and other sources that due to its nature, presents a high level of uncertainty and are unpredictable. Due to this, it must be considered that the renewable power proportion implemented as specified at *section 5.2.3.2* should be able to vary.

To provide a better and more realistic simulation capabilities it has been implemented what has been called the *power source models*. This power source models aims to provides different behaviors to the renewable proportion established at each datacenter. As it is desired to maintain the highest versatility possible at the simulator, the new characteristic is implemented in a similar way as the scheduling and meta-scheduling algorithms are implemented that inherit from a basic java class which stores some common attributes and basic behavior shared among all the algorithms.

By this way, a basic power source/supply model is implemented called *BasePowerSourceModel*. Along with this basic power source model, another three ones are implemented. Below is shown the whole power source models implemented:

- BasePowerSourceModel: the basic power source model. It provides no functionality by itself and cannot be executed. Provides basic tools (attributes and methods) to be used by the power source models that inherite from it. This allows to easily use different models without the need to modify any code for its use.
- StaticPowerModel: a basic power source model. Simply maintains the initially specified renewable power proportion along the simulation (static behavior).
- RandomPowerModel: provides a renewable proportion variation along the simulation. Modify the proportion randomly along the simulation.
- SlopePowerModel: provides a predictable variation to the renewable proportion value.
- ParametricPowerModel: provides a predictable variation to the renewable proportion value.

The *RandomPowerModel* works in the following way. It allows to specify the amount of time for which is desired to modify the proportion, in other words, if specified 10 seconds, every 10 seconds (time in the simulation scenario), the renewable proportion is modified. Additionally, it is possible to specify the maximum value the proportion is modified each time it must be modified. This maximum proportion is multiplied by a random number between -1 to 1 to allow the proportion to increase or decrease. Obviously, the proportion is limited to not allow values below 0 or above 1.

The *SlopePowerModel* works following the mathematical slope equation. It allows to specify the initial and final values for the renewable proportion and time and every what amount of time the renewable proportion must be modified. It computes the new renewable proportion value by using the slope value and the current simulation time.

The *ParametericPowerModel* works by using two lists. The first list is aimed to store all the renewable power proportion that must be used along the simulation while the second one stores the times at which each of the proportion must start to be used.

Finally, models are implemented within the *WorkflowDVFSDatacenterRenewable* entity on the updateCloudetProcessingWithoutSchedulingFutureEventsForce method. The model is execute at each time the datacenter have to compute tasks so that it continuously tests whether it is necessary or not to update the current renewable proportion used according to what is specified at the model algorithm and the parameters specified at it.

5.2.4. Additional FRBS meta-scheduling algorithm

Once implemented the two different renewable energy approach into the simulator, the next step was to provide the FRBS algorithm the ability to use theses parameters at the meta-scheduling algorithm. For this purpose, a new meta-scheduling algorithm called *RenewablePowerAwareFuzzyMaxMaxSchedulingAlgorithm* has been created. The new algorithm implements the possibility of using both, the renewable energy stored at the batteries and the renewable power supply proportion from the unlimited power supply, individually or simultaneously.

The new algorithm uses the *PowerAwareFuzzyMaxMaxSchedulingAlgorithm* as the base code to implements the new functionalities. For its implementation, minor modifications have been done to the original code, simply adding the antecedents for both, the battery capacity and the renewable proportion for the FRBS evaluation. For the battery implementation, the *cpCharge* parameter specified at *section 5.2.3.1* is used while, on the other hand, for the renewable proportion the *cARenewPSourceP* parameter specified at *section 5.3.3.2* is used. To remember, these two parameters are implemented already normalized, providing values from 0 to 1 that are adapted to be used at the FRBS.

Due to the additional antecedents the new meta-scheduling algorithm has a total of 8 antecedents, the previously 6 explained at *section 4.8.5.4* which share the same behavior, and the two additional ones for the renewable energy consideration, while still using a single consequent for the selection. Below is provided the whole list of antecedents and consequent.

Antecedents:

- MIPS
- Utilization
- Power
- Length
- Time
- Energy
- Battery: refers to the datacenter battery capacity, using the *cpCharge* parameter.
- RPS (Renewable Power Supply): refers to the renewable power supply at the datacenter, using the *cARenewPSourceP* parameter.

Consequent:

- Selection

Moreover, the defined MFs at the antecedents and the consequent are the same as specified at *section 4.8.4.4* and *4.8.5.4* for the FRBS scheduling algorithm and the previous meta-scheduling algorithm implementation.

On the other hand, to make the FRBS implementation used on the new meta-scheduling algorithm functional it was necessary to implement a new FCL file. The new FCL file, called *RenewablePowerAwareFuzzyMETASchedulingAlgorithm* implements two additional inputs for the battery charge and the renewable proportion supply called *battCapacity* and *rPS* as specified at the *antecedent list*, both defined with three MFs defining a total input range from 0 to 1, to match the implemented at the *cpCharge* and *cARenewPSourceP*.

5.2.5. Advanced tasks scheduling: Pittsburgh and Kasia

The advanced task scheduling/meta-scheduling process is not only achieved through the implementation of a FRBS meta-scheduling algorithm, and, as already specified at *section 4.3*, an optimal RB is mandatory. The quality of the RB used will define the results that will be obtained so that if a bad or not adapted to the scenario is used, the results can be significantly worse than the those obtained through the classic scheduling and meta-scheduling algorithms. To solve this, as specified at *section 4.4*, knowledge acquisition systems are used to ease obtaining optimal RBs adapted to the scenario or problem.

The knowledge acquisition systems must be implemented so that they generate random initials RBs and evaluates them through the problem to be solved, in this case, executing all the RBs on the simulator, for which the simulator must provide the necessary feedback to the knowledge acquisition system to qualify/rank each RB for its later use to create/obtain newer, better RBs.

There is a wide variety of knowledge acquisition systems, although at this stage only two are implemented which are the Pittsburgh and the Kasia. The platform chosen to implement these systems is MATLAB.

5.2.5.1. Simulator and MATLAB integration

The first problem found when implementing the knowledge acquisition systems in MATLAB was its later use along the simulator. Both systems work as separate and individual entities and it was required to allow communication between the both to, on the one hand, evaluate the generated RB at the knowledge acquisition system on the simulator and, on the other hand, provide RB evaluation feedback to the knowledge acquisition system to proceed to rank or qualify them and proceed to generate new ones from the evaluated RBs.

The easiest procedure to provide the simulator and MATLAB integration was through MATLAB *system* function. The *system* function allows to execute command line orders/sentences as if it were executed through Windows command line. This allows to, for instance, call a runnable version of the simulator through a *runnable java jar* file, providing the RB to be evaluated as an argument. Additionally, this function allows to obtain the information given by the executed command/program that would have given as a text at the command line. This allows to program the simulator to, once called through argument, provide just the necessary information to be obtained at MATLAB as feedback.

The RB is encoded in MATLAB in JSON and given it through argument to the simulator at execution. The simulator is programmed to detect when it is executed through command line and given information through argument and so, it calls the *decodeJSONRules* described at *section 5.2.2* to decode the JSON RB so that the simulator can understand and use it at the meta-scheduling algorithm. Finally, the simulator is also modified to provide an adapted output if it has been called through argument to provide an RB. It is going to provide the simulation time, energy and the proportion of energy that corresponds to renewable one. This output is provided in the way of *time:totalEnergy:renewableEnergy* so that when received at MATLAB it can be easily separated by the *":"* character and identify the three feedback parameters.

The implementation in MATLAB is done in a separate file called *rbEvaluation* (Rule Block Evaluation). The purpose of separating this code into an additional external file rather than implementing it directly within the knowledge acquisition systems is to allow an easier use of it. This code simply develops the behavior previously defined above so that it can be implemented in any kind of knowledge acquisition system by calling it.

5.2.5.2. Pittsburgh implementation

Described at *section 4.4.1*, the Pittsburgh algorithm is implemented at the simulator to obtain optimal RBs. Its implementation is mainly divided in four files, as specified below:

- pittsburghMeta: this file is where the Pittsburgh behavior is implemented. It uses the *rulecMeta, encodeMeta* and *rbEvaluation* to adapt the algorithm behavior to the Workflow simulator.
- rulecMeta: only used at the beginning of the Pittsburgh algorithm, this file provides a basic random rule creation algorithm. It is possible to specify if the number of rules that must made the generated RB is fixed or if it can vary from a minimum of 10 rules to a maximum specified number of rules. Additionally, it is adapted to allow the possibility of creating rules that lacks some of its terms and, at the same time, to avoid creating invalid rules, it checks whether at least one terms is not set to 0 (empty term) and if true, it will repeat the creation of this rule.
- encodeMeta: once rules are created or new rules have been created from the previous ones, this function encode in JSON each of the rules to make them compatible with the simulator so that can be passed to the *rbEvaluation* function for its evaluation.
- rbEvaluation: as specified on the previous section, it utilizes an encoded RB into JSON and use it to call through command line the simulator to evaluate the generated RB.

This Pittsburgh implementation is built with flexibility and versatility capabilities. It allows to specify whether it is desired to generate RB with or without renewable battery and renewable proportion supply, individually or simultaneously. If one of the two or the two are selected to be avoided, all the rules generated will lack these parameters and so, their values will not be considered while creating the optimal RB to be executed. At the same time if either of the two or both are enabled, the generated rules will include these parameters for its consideration at the evaluation. Additional parameters can be tested, as the population size, the number or rules for each RB, the mutation probability and others such as the probability of selection, which, together with the previously specified parameters to be modified, provides high research capabilities to study different scenarios.

Thanks to its implementation and use it has been possible to obtain really good results as shown at *section 6*, improving considerably the previously results obtained on previous work.

5.2.5.3. Kasia implementation

Similar to *Pittsburgh*, the Kasia algorithm described at *section 4.4.2* provides an easy way to obtain optimal RB for a specific problem. It has been implemented as well within the simulator and its implementation is based in four different files:

- nkasiaMeta: this file contains the whole Kasia algorithm implementation as described at *section 4.4.2*. It relies on external *rulecMeta, encodeMeta and rbEvaluation* functions to implement the whole necessary functionalities for its correct behavior.
- rulecMeta: same as explained in the previous section
- encodeMeta: same as explained in the previous section
- rbEvaluation: same as explained in the previous section

The Kasia implementation is done to provide the same versatility as the one provided by the Pittsburgh algorithm and explained in the previous section. Most importantly, as shown at [10][28][29][30], the authors provide results where Kasia is shown to perform better than *Pittsburgh* algorithm, making it the main reason for its implementation. However, Kasia contains three weight parameters as specified at *section 4.4.2*. The performance and correct behavior of Kasia completely rely on the correct optimization/configuration of the weight parameters. Configuring these parameters for a specific scenario to achieve optimal results from Kasia can be a long, hard work that may require the use of optimization algorithms such as *Genetics and PSO* for the optimization of these two algorithms. Due to this, although Kasia should clearly provide better results than Pittsburgh, as it will be shown later at *section 6* the results obtained through Pittsburgh are better than the obtained with Kasia, although Kasia is able to provide better results than those obtained with the classic meta-scheduling algorithms in some scenarios.

5.2.6. Multiobjective scheduling

The initial and basic multiobjective optimization implemented at both Pittsburgh and Kasia used a *weighted sum* as specified at *section 4.5*, where each parameter to be optimized, time, energy and renewable energy proportion, are given a simple weight to make the maximum value for the sum of 10. In this way, a complex problem is reduced to a single mark and so, the higher it is, the better the generated RB will be, which can be used to find better RBs. Although functional, this simple approach is not the most optimal one, as it allows the worsening of one parameter to enhance the whole final mark.

To achieve a more advanced, proper multiobjective optimization problem solution, the theory of *Pareto efficiency* is used. Explained at *section 4.5.1*, Pareto seeks for better solutions not by marks, but for the improvements of the parameters to be optimized, considering a solution to be dominant over another if these solution improve at least one parameter without worsening the rest. The first problem found while using the Pareto efficiency is the lack of a correct marking or ranking method. At Pittsburgh and Kasia, solutions must be classified or ranked so that some can be considered better than others. To solve this, the *Pareto ranking* is used, explained at *section 4.5.1.1*. By using the *AM2* ranking method, it has been possible to provide the knowledge acquisition systems a way to rank solutions for its easier use to classify RBs better than others.

The Pareto efficiency has been implemented over the *Pittsburgh* algorithm with the objective of providing a proper *MOP* solution. Its implementation is made of 4 different functions and is implemented in MATLAB as the previous one:

- pittsburghPareto: the main file that contains the main function. This one implements the Pittsburgh algorithm implementation and make use of the newer *pareto_front* and *paretoRanking* functions to implement the whole Pittsburgh with pareto efficiency implementation.
- pareto_front: a simple function that for a given group of solutions will specify the solutions that are dominant over others. Dominated solutions will be marked with a 0, while solutions that are non-dominated are marked with a 1.
- paretoRanking: function that make use of the *pareto_front* function to implement the *AM2* pareto ranking specified at *section 4.5.1.1*.
- rulecMeta: function used for the creation of random RBs.

- encodeMeta: function aimed to encode into JSON the created RBs.
- rbEvaluation: function that connect MATLAB with the WorkflowSim simulator to evaluate the generated RBs.

Although the Pareto ranking is used, it must be pointed that pareto efficiency cannot properly tells if a solution is better than other and will only specify the solutions that are non-dominated by others. After the whole execution of the Pareto Pittsburgh approach, the algorithm will not provide a single optimal solution, but a group of solutions that for the theory of Pareto efficiency are optimal due to being non-dominated by any other solution. Due to this, the algorithm is programmed to return the set of solutions that are considered as optimal and so it must be the user the one who decide the best solution/RB to be used for the problem to be solved.

6. RESULTS AND DISCUSSION

The following section is aimed to show the benefits of the additional functionalities implemented within the simulator. The achieved results obtained with the knowledge acquisition systems is provided to contrast the enhancement obtained with the FRBS meta-scheduling algorithm with the classic strategies.

6.1. Simulation scenario

To start with, a key aspect to provide results is to build a simulation scenario. The simulation scenario is necessary to show the benefits of the latest simulator implementation and so it is required to create one that shows its potential and helps to analyze the results.

The simulation scenario is made of 3 datacenters, with 20 hosts for each one and one *Pe* for each host. The scenario is built to make each host to run one single VM that fully utilize all the available resources at the hosts which, on the one hand, limits the execution of tasks to one per host, but, on the other hand, provides the maximum available performance for a task assigned to a VM for its execution. Additionally, the simulation scenario is built to be completely heterogeneous, which implies that all the datacenters are different in characteristics among them, and not only that, but the heterogeneity is also implemented among the hosts at each datacenter, so that there is not two hosts with the same resources and characteristics.

This high heterogeneity among the different resources is built to properly test the effectiveness of the new meta-scheduling algorithm. A scenario where all the datacenters and their hosts present a complete homogeneity will make the advanced algorithm implementation completely worthless as a simple Round Robin algorithm will provide the same result with no complexity on its implementation. The main benefit of the advanced scheduling algorithm is the capability to make a proper and optimal scheduling to heterogeneous scenarios and optimize the scheduling process to achieve the desired result which in this specific case is to optimize the time, total energy and renewable energy. Below is provided three tables that show each datacenter hosts characteristics, providing an explanation for the particularity of each one subsequently.

First datacenter:

Host	Maximum frequency (GHz)	Maximum voltage (V)	Capacitance (nF)	IPC	MIPS
1	3	3	5	0.5	1500
2	2.9	2.95	6.25	0.575	1667.45
3	2.8	2.9	7.5	0.65	1820
4	2.7	2.85	8.75	0.725	1957.5
5	2.6	2.8	10	0.8	2080
6	2.5	2.75	11.25	0.875	2187.5
7	2.4	2.7	12.5	0.95	2280
8	2.3	2.65	13.75	1.025	2357.5
9	2.2	2.6	15	1.1	2420
10	2.1	2.55	16.25	1.175	2467.45
11	2	2.5	17.5	1.25	2500
12	1.9	2.45	18.75	1.325	2517.5
13	1.8	2.4	20	1.4	2520
14	1.7	2.35	21.25	1.475	2507.5
15	1.6	2.3	22.5	1.55	2480
16	1.5	2.25	23.75	1.625	2437.5
17	1.4	2.2	25	1.7	2380
18	1.3	2.15	26.25	1.775	2307.5
19	1.2	2.1	27.5	1.85	2220
20	1.1	2.05	28.75	1.925	2117.5

Table 7: first datacenter hosts characteristics

In *table 7* the first datacenter hosts characteristics are shown. This datacenter is built to simulate the characteristics of an old datacenter which does not provide high performance capabilities as shown on its MIPS and IPC and utilize a high amount of voltage to power the CPUs, which, at the end, leads to high power consumption. This combination provides a not so efficient tasks execution, requiring higher time and power to execute tasks.

Second datacenter:

Host	Maximum frequency (GHz)	Maximum voltage (V)	Capacitance (nF)	IPC	MIPS
1	3.5	2	5	2	7000
2	3.435	1.975	6.25	2.025	6955.875
3	3.37	1.95	7.5	2.05	6908.45
4	3.3	1.925	8.75	2.075	6857.875
5	3.24	1.9	10	2.1	6804
6	3.175	1.875	11.25	2.125	6746.875
7	3.11	1.85	12.5	2.15	6686.5
8	3.045	1.825	13.75	2.175	6622.875
9	2.98	1.8	15	2.2	6556
10	2.91	1.775	16.25	2.225	6485.875
11	2.85	1.75	17.5	2.225	6412.5
12	2.785	1.725	18.75	2.275	6335.875
13	2.72	1.7	20	2.3	6256
14	2.655	1.675	21.25	2.325	6172.875
15	2.59	1.65	22.5	2.35	6086.5
16	2.525	1.625	23.75	2.375	5996.875
17	2.46	1.6	25	2.4	5904
18	2.395	1.575	26.2	2.425	5807.875
19	2.33	1.55	27.5	2.45	5708.5
20	2.265	1.525	28.75	2.475	5605.875

Table 8: second datacenter hosts characteristics

In *table 8* the second datacenter hosts characteristics are provided. This datacenter, opposed to the first one, is built to provide a more modern implementation. On this one, the performance provided to execute tasks is higher, represented by a higher amount of MIPS and a much higher IPC. Additionally, it requires significantly less voltage to power the CPUs which together with the higher performance provide a more efficient tasks execution that will provide lower amount of time and power required to execute their tasks.

Third datacenter:

Host	Maximum frequency (GHz)	Maximum voltage (V)	Capacitance (nF)	IPC	MIPS
1	2.2	2	5	0.5	1100
2	2.155	1.975	6.25	0.565	1217.575
3	2.11	1.95	7.5	0.63	1329.3
4	2.065	1.925	8.75	0.695	1435.175
5	2.02	1.9	10	0.76	1535.2
6	1.975	1.875	11.25	0.825	1629.375
7	1.93	1.85	12.5	0.89	1717.7
8	1.885	1.825	13.75	0.955	1800.175
9	1.84	1.8	15	1.02	1876.5
10	1.795	1.775	16.2	1.085	1947.575
11	1.75	1.75	17.5	1.15	2012.45
12	1.705	1.725	18.75	1.215	2071.575
13	1.66	1.7	20	1.28	2124.8
14	1.615	1.675	21.25	1.345	2172.175
15	1.57	1.65	22.5	1.41	2213.7
16	1.525	1.625	23.75	1.475	2249.375
17	1.48	1.6	25	1.54	2279.2
18	1.435	1.575	26.25	1.605	2303.175
19	1.39	1.55	27.5	1.67	2321.3
20	1.345	1.525	28.75	1.735	2333.575

Table 9: third datacenter hosts characteristics

In *table 9* is shown the third datacenter hosts characteristics. This last datacenter is made to simulate a modern low power consumption datacenter. The performance capabilities are like the one provided by the first datacenter, although, on the other hand, the power consumption is much lower obtained thanks to its much lower voltage requirements and its lower frequency.

As it is desired to obtain the maximum throughput possible with the CPUs on each host while still allowing the reduction of the power consumption, the *OnDemand* governor has been used. As specified at *section 4.7*, the *OnDemand* governors dynamically change the CPU frequency and voltage according to the current CPU load, allowing to raise its frequency and voltage if more performance is required and automatically reducing it when a high performance is not required to reduce the power consumption.

Another important aspect when testing and studying the scheduling and meta-scheduling algorithms is the tasks to be executed. A proper task scenario must be selected if it is desired to correctly highlight the benefits of the created meta-scheduling algorithms. A high or low number of tasks will not allow to properly study the creation of newer meta-scheduling algorithms. If the number of tasks is low, all tasks can be easily scheduled to the most powerful datacenter and so it will not be possible to obtain and see different results among the different meta-scheduling algorithms. On the other hand, if a high or really high number of tasks are used, datacenters can reach a saturation state in where the scheduling of tasks must be continuously delayed until one resource is available for its use to execute tasks, and so, using different meta-scheduling algorithm will make no difference on the obtained execution results. Due to this, the DAX implemented in the simulator that has been used is the DAX Montage_100, which is made of 100 tasks and should avoid the previously specified problems.

Moreover, some additional features apart from the advanced meta-scheduling algorithms have been implemented such as the on-line planning. As specified, this planning method force the system to schedule tasks one by one, which for our particular scenario will not be of use and will harden the process of study of the meta-scheduling strategies, reason for which the on-batch planning is used, as it allows the scheduling of multiple tasks simultaneously into *BoT*.

As for the renewable energy, to remember, two different approaches are implemented, one with batteries and other with a renewable power supply proportion. At this stage, all test and researches are done ignoring batteries and using instead the renewable power supply proportion, which means that the renewable energy stored into the batteries has been set to 0. Additionally, all the study is done with a static renewable power proportion. To generate an optimal RB for a specific scenario it is required that the scenario presents certain level of predictable behavior. A scenario where the renewable proportion change can make the

knowledge acquisition system to not be able to reach a proper optimal RB, especially if the renewable proportion does not follow a complete predictable behavior and change in a different way at each execution.

More important, as specified at *section 5.2.2*, the implementation of the negated terms and the OR/AND terms connector allows to obtain optimal RBs with a significant lower amount of rules, fact that will be demonstrated at the *results section*. This assertion does not mean that without this last implementation into the FRBS meta-scheduling algorithms it is not possible to achieve optimal RBs/results, but it simply allows to obtain the same optimal results with fewer rules.

Finally, it must be pointed that the objective of this master thesis is to study and test different meta-scheduling strategies, for which a single *local scheduling* algorithm must be used and shared among all the different tests. Due to this, the local scheduling algorithm used in all the datacenters and tests is the *PowerAwareMaxMinSchedulingAlgorithm*, created at the *WorkflowSim with DVFS* simulator. As specified at *section 4.8.4.3*, this algorithm schedule tasks seeking to minimize the power consumption due to their processing rather than minimizing their execution time.

6.1.1. Knowledge acquisition algorithms configuration

Another important aspect at the simulation scenario is the configuration used at the knowledge acquisition systems. The configuration used plays a key role at the algorithms behavior and so the provided/achieved results by these algorithms. A bad configuration may lead to bad solutions/RBs, while a good configuration should ease obtaining optimal results. Moreover, depending on the values of some selected parameters, the execution time for these algorithms can significantly raise, which may or may not help to obtain better results, as these algorithms relies on several configuration parameters.

To remember, the developed knowledge acquisition algorithms are the following:

- Pittsburgh
- Kasia
- Pittsburgh + Pareto + AM2

All these algorithms share some common parameters as the population size/number of particles, minimum and maximum number of rules for each solution/RB, and the number of iterations. To try and place all these algorithms under the same conditions, these common parameters have been configured for the three algorithms with the same values, which are specified below:

- Population size/number of particles: 40
- Minimum number of rules for each RB: 10
- Maximum number of rules for each RB: 40
- Number of iterations: 20

It is important to point that the number of rules for each RB is randomly generated, guaranteeing a minimum number of 10 and maximum number of 40 as specified.

Additionally, both the Pittsburgh and the Pittsburgh with Pareto shares another common parameter, which is the number of parents to be selected for the creation of newer populations, where the number of parents has been set to 20.

Moreover, for the parent selection process the *roulette algorithm* has been used. This algorithm makes use of a Round Robin process where each particle is tested to see if it must be selected as a parent according to a selection probability that is assigned by means of the *particle quality*. This probability of selection is assigned in a way that provides a higher selection probability to the best solutions, providing lower selection probabilities to the worst particles/solutions. To achieve this, particles are ordered from best to worst and, once this is done, a *slope function* is used to assign the probability of selection from best to worst. Again, Pittsburgh and Pittsburgh + Pareto use these probabilities of selection, for which has been used a first and last probability of selection but, in this case, each algorithm has been given with a different probability of selection.

Pittsburgh probability of selection:

- Maximum probability of selection: 95%
- Minimum probability of selection: 5%

Pittsburgh + Pareto probability of selection:

- Maximum probability of selection: 60%
- Minimum probability of selection: 20%

Additionally, there is a last parameter that must be configured at these two algorithms, which is the *mutation probability*, which purpose is specified at *section 4.4.1*. Both algorithms have been set with the same mutation probability as specified below:

- Mutation probability: 25%

Finally, with respect to Kasia, there are 3 parameters that can be configured which are the algorithm *weights* specified at *section 4.4.2*. These weights are given the following values:

- w: 5
- φ_1: 2
- φ_2: 2

Note that the value set for the w parameter refers to the initial weight, which value reduce along time. The more iterations that have been executed, the lower that this value will be.

6.2. Results

The result section is aimed to provide information about the results achieved through all the development of this master thesis and all the improvements that have been done to the simulator. At this stage, the meta-scheduling algorithms used are:

- RoundRobinMetaSchedulingAlgorithm
- MinMinMetaSchedulingAlgorithm
- MaxMinMetaSchedulingAlgorithm
- RenewablePowerAwareMetaSchedulingAlgorithm

As a reference, all the results are going to be provided both numerically and in a percentage improvement way using the Round Robin algorithm as the basic one used to be contrasted to show the percentage improvement of every algorithm.

Moreover, at this section it will not only be contrasted the FRBS algorithm versus the classic algorithms, but it will also be tested the different knowledge acquisition system used for the optimization of the RBs. The knowledge acquisition system used so far are the following:

- Pittsburgh
- Kasia
- Pittsburgh + Pareto + AM2 ranking

The Pittsburgh and Kasia algorithms that does not use the Pareto approach use a weighted sum as previously specified. The three parameters that are optimized are the execution time, the energy, and the renewable energy proportion, using the following weights at the evaluation process to obtain a numerical maximum score of 10:

- Execution time: 45%
- Total execution energy: 40%
- Renewable energy proportion: 15%

6.2.1. Basic meta-scheduling algorithms vs FRBS algorithm.

At this subsection it is going to be compared the results obtained in the specified scenario with the basic algorithms and the FRBS algorithms. The RB used for the FRBS algorithm are obtained through the Pittsburgh and Kasia algorithms, both without Pareto and with the implementation of the negated terms and the OR connector. Additionally, this RB is obtained making the Pittsburgh and Kasia algorithms to optimize the time, energy and renewable energy at the same time.

In *table 10 and 11*, the different algorithms are compared, showing both, numerical and percentage results. In these tables is possible to see the improvements achieved by all the algorithms over the Round Robin one, highlighting the improvements achieved through the Pittsburgh algorithm. In the percentage table is easier to see the achieved improvements, showing the percentage save/improvement over the classic Round Robin algorithm, were the best results are achieved through the Pittsburgh algorithm by far.

On the other hand, although the RB provided by the Pittsburgh algorithm provides the best result, the Kasia algorithm not only falls far behind the Pittsburgh one, but also provides worse results than the one achieved through the basic meta-scheduling algorithms at this scenario. As specified at *section 5.2.5.3*,Kasia used 3 weights parameters that must be correctly configured for each specific scenario to obtain optimal results. Although great efforts have been put on optimizing these parameters, it has not been possible to fully

optimize them, reason for which the results provided by Kasia are not consistent with what is expected at this knowledge acquisition algorithm.

Note that in *table 11*, average power shown for the RB obtained through Kasia states a saving of -15,95% which indeed *is not* a saving in percentage, but the increase of 15,95% with respect to the Round Robin algorithm.

Note: for this simulation scenario, the following renewable energies proportion for each datacenter are used:

- First datacenter: 80%
- Second datacenter: 30%
- Third datacenter: 50%

As seen, the highest renewable energy proportion is given to the least efficient datacenter while the lowest proportion is given to the most efficient one. The objective is to see and test the capabilities of the Pittsburgh algorithm to try and obtain a RB that provides a good balance among all the parameters to be optimized while trying to provide the best result possible.

Algorithm	**Results**					
	Time (s)	**Total power (W)**	**Average power (W)**	**Total Energy (Wh)**	**Non-renewable Energy (Wh)**	**Renewable Energy (Wh)**
RoundRobin	134,88	274119,62	6,77	76,14	27,33	48,82
MinMin	76,47	141819,12	6,18	39,39	18,26	21,14
MaxMin	75,81	140935,51	6,19	39,15	18,12	21,03
Pittsburgh	65,86	105153,87	5,32	29,21	14,77	14,44
Kasia	81,37	191570,42	7,85	53,21	21,67	31,54

Table 10: algorithms comparison. Numerical results

Algorithm	Save (%)					
	Time	Total power	Average power	Total Energy	Non-renewable Energy	Renewable Energy
RoundRobin	-	-	-	-	-	-
MinMin	43,31	48,26	8,71	48,27	33,19	56,69
MaxMin	43,79	48,59	8,57	48,58	33,69	56,92
Pittsburgh	51,17	61,64	21,42	61,64	45,95	70,42
Kasia	39,67	30,11	-15,95	30,11	20,71	35,39

Table 11: algorithms comparison. Percentage results.

To ease the understanding of the results, different graphs are provided showing the numerical and percentage improvement of the obtained results. Note that due to the high difference in the range of values, the results are divided in multiple graphs to ease its visibility and understanding.

With respect to the time and energy consumption, as shown in *figure 17*, the Round Robin provides the worst results as expected, obtaining high improvements with the other algorithms. Moreover, the obtained RB through Pittsbutgh provides additional improvements with respects to the MaxMin and MinMin algorithms, demonstrating that the results obtained through the classic algorithms can be improved with the use of FRBS. However, as stated before, the RB obtained through Kasia provides worse results than Pittsburgh and the classic algorithms with the exeption of the Round Robin algorithm.

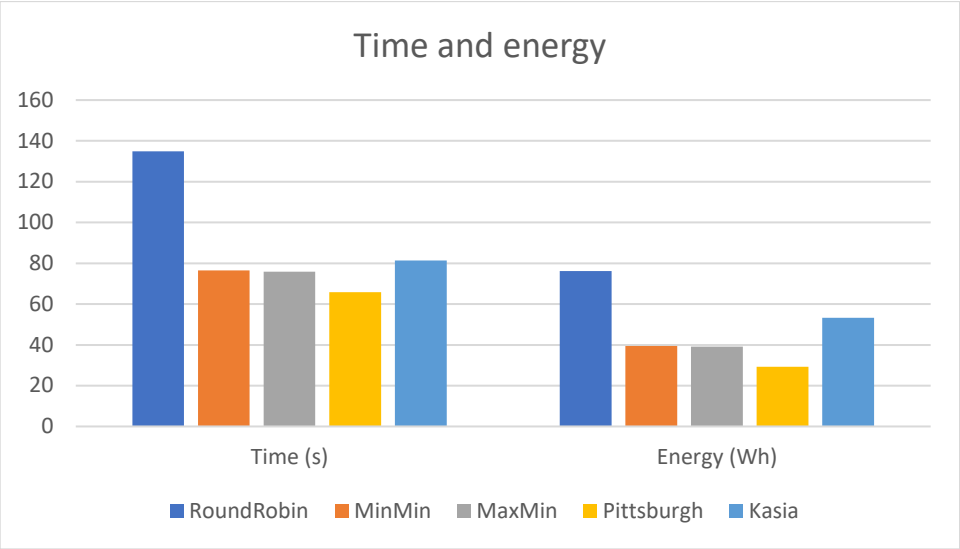

Figure 14: time and energy numerical results

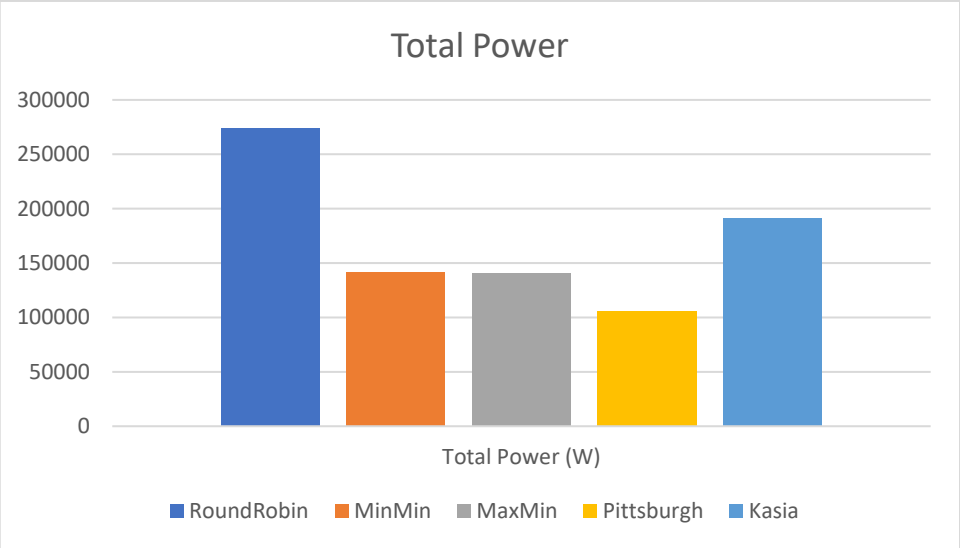

Figure 15: total power numerical results

As shown at *figure 18*, the same scenario is obtained with the total power consumption, where the worst result is provided through the Round Robin algorithm, obtaining significant improvements with the other algorithms, showing once more that the classic algorithms can be improved through FRBS, more specifically if Pittsburgh is used to obtain the RB to be used, which, as shown, provides the best results.

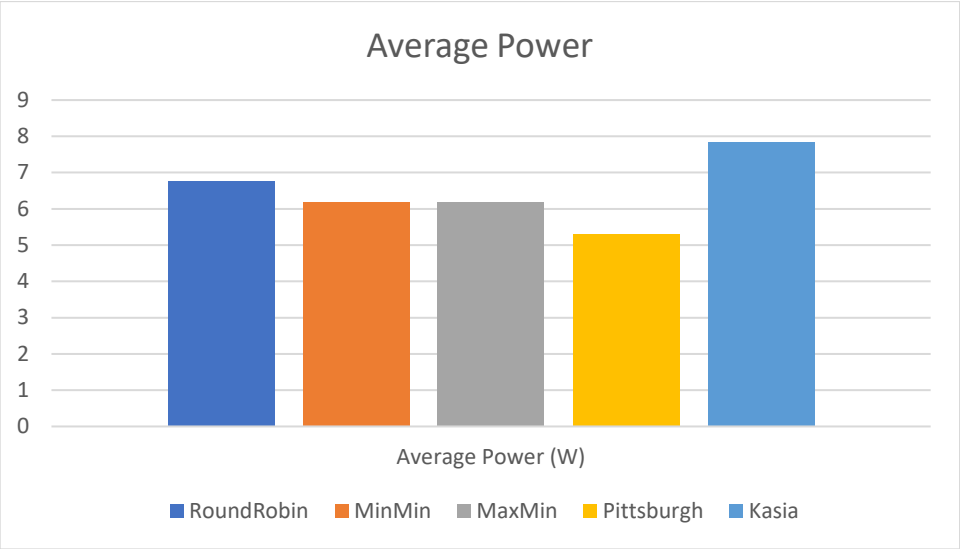

Figure 16: average power numerical results

A peculiar result is obtained and shown at *figure 19*, where surprisingly the worst result is not provided by the Round Robin algorithm but provided by the FRBS algorithm when using the RB obtained with Kasia. Note that excluding this peculiar result, the other obtained results reflects what is expected, with Round Robin providing the worst result, improved by MaxMin and MinMin and obtaining the best result using the RB provided by Pittsburgh.

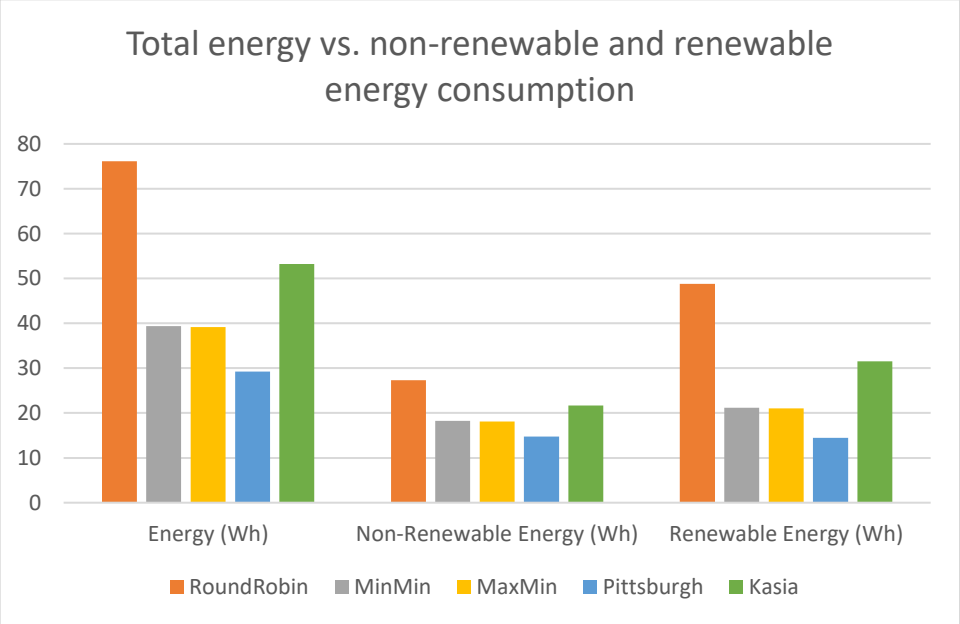

Figure 17: energy comparison numerical results

As expected, when comparing the total, non-renewable and renewable energy, the same results are obtained as shown at *figure 20*. It is important to point that the total energy is the sum of the non-renewable and the renewable energy, and so, a direct reduction of the total energy consumption will be reflected as a reduction at the non-renewable and renewable energy as well.

In *figure 21* is shown the general improvements results in percentage. This graph compares all the algorithms with the Round Robin one, showing the percentage improvement/save, making completely unnecessary to include the Round Robin into the graph as its percentage improvement with itself is 0%. Due to this, it is clearly seen that, with the exception of Kasia at the *average power*, all the algorithms provide a direct significant improvement on the reduction of all the parameters over the Round Robin one. There is a scenario where this graphic information can be tricky and is at the *renewable energy* consumption reduction/improvement. As this graph refers to direct reduction of the specified parameters, a reduction of the renewable energy use could be seen as bad, as one of the objectives is to prioritize the use of the renewable energy over the non-renewable one, or in other words, reduce the use of the non-renewable energy by using more renewable energy, although the most important part is the total energy consumption. As seen when contrasting the energy, non-renewable energy and the renewable energy improvement, all of them are significantly reduced, being the real important parameter to be considered the total energy consumption reduction. Additionally, at *table 12* where the three energy parameters are shown numerically, it is shown how the RB obtained through Pittsburgh provides the lowest energy consumption for the three parameters, total, renewable and non-renewable, which means that even though this algorithm makes the highest reduction into the renewable energy consumption, the overall provided result is the best of all the different algorithms.

To try to ease the understanding of the previous statement, at *table 12* additional information with regard the energy consumption is shown. As it can be seen, the lowest renewable energy proportion is provided by the RB obtained through the Pittsburgh algorithm, which means that, from the total amount of energy consumption done for the execution of tasks, the Pittsburgh algorithm is the one that use the lowest renewable energy in percentage way. On the other hand, as previously said, the most important parameter to be considered is, once more, the total energy consumption, clearly showing that the Pittsburgh algorithm is the one that provides the lowest amount of energy consumption by far when compared to the other algorithms even though it is the one that proportionally used

the lowest amount of renewable energy. At the same time, it is shown that the lowest non-renewable and renewable energy consumption is done through the Pittsburgh algorithm, showing that the slightly lower renewable energy proportion is not as important as it looked, but that the most important aspect is the whole energy consumption done through the use of every algorithm.

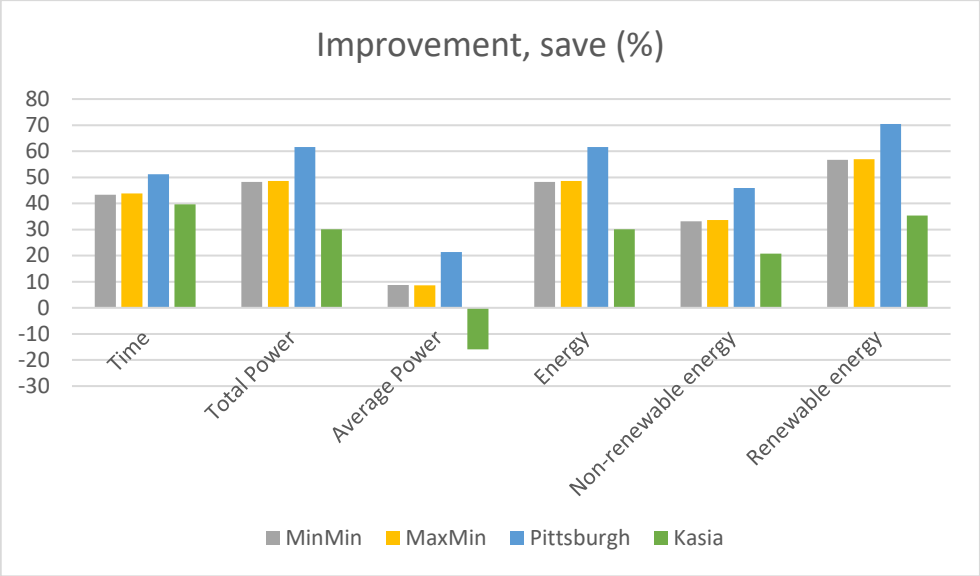

Figure 18: general improvement results in percentage

Algorithms	Results			
	Total energy (Wh)	**Non-renewable energy (Wh)**	**Renewable energy (Wh)**	**Renewable proportion (%)**
RoundRobin	76,14	27,33	48,82	64,12
MaxMin	39,39	18,26	21,14	53,67
MinMin	39,15	18,12	21,03	53,72
Pittsburgh	29,21	14,77	14,44	49,44
Kasia	53,21	21,67	31,54	53,64

Table 12: energy comparison and renewable proportion results

6.2.2. Negated terms and AND/OR implementation rule number impact

This subsection is aimed to show the virtues of the implementation of the negated terms and the AND/OR term connectors implementation. As stated at *section 5.2.2*, thanks to the implementation of these new FRBS characteristics it is possible to achieve more efficient RBs when compared to the previous implementation, where no negated terms were used and the AND connector was the only one used. The more efficient RBs refers to the possibility to obtain RBs made of a lesser number of rules while achieving the same results.

The Pittsburgh algorithm is used to obtain the RBs to compare results, optimizing only for time and total energy consumption.

Note: the renewable proportions used at each datacenter are the same as specified at *section 6.2.1*.

Algorithms	**Results**				
	Time (s)	**Total energy (Wh)**	**Non-renewable energy (Wh)**	**Renewable energy (Wh)**	**Number of rules**
Pittsburgh w/o negated terms, OR	65,53	29,17	14,77	14,40	32
Pittsburgh w/ negated terms, OR	65,59	28,94	14,69	14,26	19

Table 13: FRBS comparison

As shown in *table 13*, the numerical results obtained through the two different FRBS implementation are quite similar with very little variations but, as stated before, the number of rules for the implementation that use negated and non-negated terms together with the simultaneous use of AND/OR connectors is significantly lower.

One of the benefits of the lesser number of rules is a lighter FRBS, which translate into a faster response for the inputs/antecedents evaluation through the rules. As there is a lesser

number of rules, each antecedent will have to be evaluated a lower number of times, which provides a faster response. At the simulation scenario it does not provide better results, as the simulation is paused while the FRBS is evaluating all the inputs, but in a real scenario, antecedents require time for their evaluation, time that will grow fast as the number of rules is increased. Due to this, even if two RBs provide the exact same results on evaluation, in a real cloud scenario the one with a lower number or rules will provide a faster execution and so, a lower energy consumption for the tasks execution.

It is important to point that the simulator does not consider or simulate the time required for either the scheduling or the meta-scheduling process, and so, only consider the time required to execute tasks once those tasks are being executed into the assigned VM. Due to this, it is not possible to see a specific enhancement on the obtained results with the RB made of lesser rules, although in a real time scenario, where the scheduling and meta-scheduling process take part at the whole execution time of tasks, a slight difference should be achieved in favor of the RB with lesser rules.

On the other hand, the reduced number of rules provides a significant increase in terms of *interpretability*. The *interpretability* or *explainable artificial intelligence (XAI)* refers to the ability to provide easier to understand solutions that are created through any kind of artificial intelligence, or, in this case, FRBS. To clarify it, a RB made of a high number of rules will presents a low interpretability degree, or in other words, will present high difficulties to understand or translate the strategy followed to obtain the achieved results. With the reduction of the number of rules, it will be easier to understand by humans the logic and strategy defined at the RB that is used to achieve a certain solution, which is one of the objective to follow when creating solutions through any kind of artificial intelligence. It must be pointed that an optimal solutions, although functional, can be worthless if it is not possible to understand and interpret the intelligence defined to solve the solution, and so it is of high interest to provide solutions that can be understood while being optimal at the same time. At this particular case, interpretable solutions are achieved y reducing the number of rules of the RBs created to solve problems, which is what has been achieved through the implementation of the negated and non-negated simultaneous use together with the AND/OR connectors.

6.2.3. Pittsburgh vs Kasia vs Pittsburgh + Pareto + AM2

At this last subsection the 3 different knowledge acquisition algorithms implemented are compared. In order to show the benefits and flaws of these algorithms, it is necessary to properly test them. First, it is important to point that the knowledge acquisition algorithms will not provide a *fixed solution*. Each execution of these algorithms can and will provide different results, most of the times showing just minor differences among different executions. Second, depending on the scenario used the algorithms provide different results, being possible that, under certain scenarios some algorithms may provide bad results, while providing better results with other scenarios. Moreover, as specified at *section 5.2.5.3*, the Kasia algorithm implementation will not provide the results that it should provide according with what is specified at [10][28][29][30]. This algorithm should provide better results than Pittsburgh, but, as it is going to be shown on the provided results, this statement is not true for this particular scenario. Kasia relies on some *weights* parameters that, if not correctly adjusted/configured for the specific problem, will make it to not provide the desired/expected results. The optimization of the weights is an arduous work, requiring a significant high amount of time, both manually or through optimization algorithms such as *Genetic* or *Swarm*. As the correct optimization of these parameters may requires months, the weights have not been completely optimized to work under the different scenarios studied.

Additionally, as well as occurs with Kasia, some problems are found when optimizing solutions/RBs through the Pareto efficiency. First, as specified at *section 5.2.6*, Pareto does not provide a direct method or way to specify which solution is best but should provide a group of solutions that are considered as optimal, *non-dominated* or *dominants* under the theory of Pareto. Due to this, the final user must be the one in charge of manually selecting the solution that is considered as optimal, using the own expert criterion. As it will be seen, in this particular scenario Pareto efficiency does not provide a wide variety of results and so all the obtained results for each scenario are provided. Moreover, due to this a *pareto front* cannot be provided as there is an insufficient amount of data for its creation. Finally, a proper explanation about the reason behind the incorrect Pareto behavior/problems is going to be provided after the results achieved through the three studied scenarios.

The three created scenarios used exactly the same three datacenters as explained at *section 6.1* as well as using the same weights for the weighted sum for the Pittsburgh and Kasia algorithms without Pareto as stated at *section 6.2*. The three scenarios only differ on the established renewable proportion, being these proportion for each scenario as specified below:

- First scenario:
    - First datacenter: 80%
    - Second datacenter: 50%
    - Third datacenter: 50%
- Second scenario:
    - First datacenter: 50%
    - Second datacenter: 80%
    - Third datacenter: 50%
- Third scenario:
    - First datacenter: 50%
    - Second datacenter: 50%
    - Third datacenter: 80%

These scenarios are built in this way to try and study the behavior of the knowledge acquisition systems when the highest renewable proportion is found in a specific datacenter.

On the first scenario, the highest renewable proportion is set at the first datacenter that, to remember, together with the third datacenter has the lowest MIPS capabilities but, opposed to the third datacenter, the first one also make the highest power consumption. Due to this characteristic at the first datacenter, it is known that tasks scheduled to it will require a high amount of time for its execution and will make the highest energy consumption. As the renewable proportion is also a parameter to be optimized at the three knowledge acquisition systems, it is interesting to see how it affects to the creation of optimal RBs, which may end on the creation of suboptimal RBs due to the high renewable proportion at the first datacenter. It is important to point that this consideration is also going to affect the way the RBs are generated at the two other scenarios where the highest renewable proportion is set at different datacenters.

On the second scenario the highest renewable energy proportion is set at the second datacenter which is the one with the highest MIPS and together with the third datacenter, the one with the lowest power consumption. Due to the characteristics on this datacenter it is known that tasks executed at this datacenter must provide, on the one hand, the lowest execution time and, on the other hand, the lowest energy consumption (product of time executing tasks and the power consumption). As occurs with the first scenario, it is important

to study how providing the highest renewable proportion to this datacenter can affect to the creation of RBs.

Finally, on the third scenario the highest renewable energy proportion is given to the third datacenter which possess a MIPS capability like the first datacenter but with a power consumption like the second datacenter. Due to this, the third datacenter provides a more efficient execution of tasks than the one provided by the first datacenter but not as efficient as the second one, as although the power consumption is similar, its lower MIPS capability will prolong the time required to complete the execution of tasks and so will make this datacenter to make a higher energy consumption than the second datacenter.

As it is shown in *tables 14, 15, 16, 17, 18 and 19*, the Round Robin, MaxMin and MinMin algorithms which are provided as reference results provides the same execution timepower and energy consumption on the three tested scenarios while providing different renewable proportion results. These results are obtained as these basic algorithms implement neither any method to optimize the renewable energy consumption nor the total energy consumption and as the three scenarios only differ on the renewable proportions and these algorithms focus on the optimization of time, the results obtained through the three scenarios in time and total energy consumption are the same. However, these results are just provided as reference values to be used when comparing the results achieved with the three knowledge acquisition systems.

NOTE: take into account that at all the tables, the algorithms used at Pittsburgh, Kasia and Pareto is the *RenewablePowerAwareFuzzyMaxMaxSchedulingAlgorithm* and that the algorithms provides the RBs that provide the given results. Additionally, as previously specified, Pareto should and can provide multiple solutions after one single execution, all considered to be optimal and so all the solutions provided by it are added at each scenario entitled as *Sx*, where *S* refers to *solution* and *x* refers to the solution number.

First scenario:

In *tables14 and 15* all the results related to the simulation of the first scenario through the different algorithms are shown. As shown, the best results are obtained through Pittsburgh and Kasia, being the first the one that provides the best results. With respect to Kasia, although the improvement achieved on the execution time over the classics MinMin and MaxMin is incredibly small, the improvements achieved with respect to the power and the energy consumption is significantly improved over the classic algorithms. Finally, at this

simulation scenario, the Pareto algorithm only achieve to beat the results provided by the RoundRobin algorithm, falling far behind the solution provided by the other algorithms.

Due to the particular behavior of the Pareto algorithm and the use of the *AM2* ranking method, this algorithm can easily fall into a local best/optimal from where the algorithm will probably not be able to get out of, leading all the particles/solutions to this suboptimal local optimal solutions. Looking at the results provided with Pareto, the renewable proportion provided is outstanding with results superior to the 70%. This result tells that the RBs generated make most of the tasks to be scheduled to the first datacenter which have a renewable proportion of 80%. Due to the exceptional renewable energy proportion obtained, these two RBs are marked as optimal due to the *dominancy* theory in Pareto, identifying them as *dominant* over all the other solutions found, or, in other words, these two solution are *non-dominated*, which means that it cannot find any other solution that is able to dominate the two provided solutions. However, this does not mean that there is not any other solution that could be considered as *non-dominated* as the two provided solutions, but due to the behavior of the *AM2* ranking method over the Pareto it makes incredibly difficult to the algorithm to find other solutions. Comparing the results provided at the *Pittsburgh* and *Kasia* algorithms with the two solutions provided by the Pareto algorithm, it can been seen that the solutions provided by these two others algorithms do not show *dominancy* over the solutions provided by the Pareto algorithm, but, at the same time, the Pareto solutions are not *dominant* over the solutions provided by Pittsburgh and Kasia, which means that, in fact, there is additional optimal solutions to be found, although the Pittsburgh solution is *dominant* over the one provided by Kasia. What makes Pittsburgh and Kasia able to find this other optimal or non-dominated solutions is the evaluation method implementation, which base the classification of one solution as better than the other through an overall mark, which, in this particular case, have helped this two algorithms to find this solutions that tries to provide a better and more advanced solutions overall. To understand it, this evaluation method allows these algorithms to create RBs that schedule tasks among the available datacenter fairly or better, trying to make use of the strong points of each datacenter, which is a complex process. On the other hand, the Pareto algorithm seems to follow a simple and easy approach, which is to schedule all tasks or the majority of them to the datacenter that guarantee the best renewable energy proportion rather than trying to find a more advanced solution that could be considered as better for the case of study.

Algorithms	Results					
	Time (s)	Total energy (Wh)	Non-renewable energy (Wh)	Renewable energy (Wh)	Renewable proportion (%)	Number of rules
RoundRobin	134,88	76,14	24,03	52,12	68,45	-
MinMin	76,47	39,39	14,62	24,77	62,88	-
MaxMin	75,81	38,15	14,52	24,63	64,56	-
Pittsburgh	65,59	28,94	11,5687	17,57	60,71	12
Kasia	74,10	32,21	13,27	18,97	58,89	14
Pareto – S1	128,03	59,37	15,89	43,48	73,24	12
Pareto – S2	128,03	59,58	15,99	43,58	73,15	12

Table 14: first scenario knowledge acquisition time, energy and number of rules comparison

Algorithms	Results	
	Total power (W)	Average power (W)
RoundRobin	274119,62	6,77
MinMin	141819,12	6,18
MaxMin	140935,51	6,19
Pittsburgh	104191,82	5,29
Kasia	116064,68	5,22
Pareto – S1	213722,13	5,56
Pareto – S2	214492,06	5,58

Table 15: first scenario knowledge acquisition power comparison

- Scenario two:

The *tables 16 and 17* provide the results achieved at the second scenario. As seen, the results provided through Pittsburgh and Kasia are so close to each other, providing slightly better results than those provided by the classic algorithm but providing quite better results at the power and energy consumption. At the same time, it is seen that, except for the first Pareto solution, the other provided solutions are similar to the one provided by Pittsburgh and Pareto as well. As specified on the first scenario, due to the use of the AM2 ranking method over Pareto, Pareto tends to focus on scheduling tasks to one single datacenter rather

than scheduling tasks to multiple datacenters trying to avoid the datacenters weaknesses to reach the best results possible. In this scenario the highest renewable energy proportion is established to the datacenter with the highest MIPS capabilities and the lowest power consumption, which ease the process of finding an optimal and useful RB.

Algorithms	Results					
	Time (s)	Total energy (Wh)	Non-renewable energy (Wh)	Renewable energy (Wh)	Renewable proportion	Number of rules
RoundRobin	134,88	76,14	33,12	43,02	56,50	-
MinMin	76,47	39,39	14,24	25,15	63,85	-
MaxMin	75,81	38,15	14,17	24,98	65,48	-
Pittsburgh	73,16	31,82	10,69	21,13	66,40	14
Kasia	73,14	31,87	10,72	21,15	66,36	16
Pareto – S1	80,66	34,18	11,34	22,84	66,82	13
Pareto – S2	73,16	31,74	10,65	21,09	66,45	13
Pareto – S3	73,14	31,87	10,72	21,15	66,36	25
Pareto – S4	73,37	32,19	10,87	21,33	66,26	11
Pareto – S5	73,18	32,15	10,87	21,29	66,22	13

Table 16: second scenario knowledge acquisition time, energy and number of rules comparison

Algorithms	Results	
	Total power (W)	Average power (W)
RoundRobin	274119,62	6,77
MinMin	141819,12	6,18
MaxMin	140935,51	6,19
Pittsburgh	114548,70	5,22
Kasia	114738,74	5,23
Pareto – S1	123055,47	5,08
Pareto – S2	114265,00	5,21
Pareto – S3	114738,74	5,23
Pareto – S4	115890,79	5,27
Pareto – S5	115754,45	5,28

Table 17: second scenario knowledge acquisition power comparison

- Scenario three:

At tables 18 and 19 the results obtained at the third scenario are provided. As seen, at this one only the Pittsburgh algorithm is able to beat the results achieved through the classic MinMin and MaxMin algorithms with Kasia slightly falling behind in time and far behind in power and energy consumption with respect to the classic algorithms. Additionally, the results provided by the Pareto are significantly worse than the one obtained through the other algorithms, especially on the execution time, although beat the power and energy consumption results obtained with the Round Robin and Kasia algorithms. These results, when compared to the Pareto results provided at the *first scenario* looks quite similar, especially on time and the renewable energy proportion. At this scenario the highest amount of renewable proportion is given to the third datacenter that provides a power consumption similar to the one provided by the second one although even smaller and a MIPS capacity similar to the one provided by the first datacenter but slightly smaller. Considering this and the Pareto behavior shown on previous scenarios, the obtained results look logical. Again, Pareto seems to provide a behavior that prioritize the scheduling of tasks to one single datacenter avoiding the creation of complex RBs that explores the strong points of each datacenter to find optimal and useful results and.

Algorithms	**Results**					
	Time (s)	**Total energy (Wh)**	**Non-renewable energy (Wh)**	**Renewable energy (Wh)**	**Renewable proportion**	**Number of rules**
RoundRobin	134,88	76,14	34,23	41,92	55,06	-
MinMin	76,47	39,39	18,41	20,99	53,29	-
MaxMin	75,81	38,15	18,29	20,86	54,68	-
Pittsburgh	73,14	31,87	14,44	17,44	54,72	18
Kasia	81,37	53,21	25,11	28,10	52,81	14
Pareto – S1	146,30	42,29	14,74	27,55	65,15	14
Pareto – S2	146,10	42,73	14,97	27,76	64,97	14

Table 18: third scenario knowledge acquisition time, energy and number or rules comparison

Algorithms	Results	
	Total power (W)	**Average power (W)**
RoundRobin	274119,62	6,77
MinMin	141819,12	6,18
MaxMin	140935,51	6,19
Pittsburgh	114738,74	5,23
Kasia	191570,42	7,85
Pareto – S1	152257,32	3,47
Pareto – S2	153831,30	3,51

Table 19: third scenario knowledge acquisition power comparison

7. CONCLUSIONS

To conclude, this section is aimed to summarize all the work developed at this master thesis specifying the objectives and milestones achieved and the results obtained.

To start with, one of the main objectives was to properly extends the capabilities and make major code cleanups of the previously developed *WorkflowSim with DVFS and Metascheduling* at my bachelor's final project/work, objective that has been successfully achieved. First, as shown at *section 5.1*, several code cleanups have been done using the Java inheritance to make the *meta-scheduler* entities to directly extends from the *WorkflowSim with DVFS* entities achieving an important code reduction, as well as the partial removal of the static class used before for all the meta-scheduling algorithms. Second, plenty of improvements and additions have been included on the simulator as specified at *section 5.2* such as the classic on-line planning method, the extension of the FRBS capabilities through the negated terms and the OR connectors, the renewable extensions, a newer FRBS meta-scheduling algorithm that make use of the new FRBS capabilities as well as the renewable extensions, the knowledge acquisition system and the multiobjective scheduling.

At the *results section* is shown several important milestones have been achieved. First, it is shown the effectiveness of the knowledge acquisition systems such as *Pittsburgh* and *Kasia* to properly provide an easy method to obtain optimal RBs that easily beat the results provided through the classic meta-scheduling algorithms, as shown at *section 6.2.1*.

Second, thanks to the inclusion of the extended FRBS capabilities with the negated terms and OR connectors it has been possible to achieve a significant reduction of the number of rules within the generated RBs through the knowledge acquisition systems. As specified at *section 6.2.2*, it will provide a faster scheduling and meta-scheduling process at the meta-scheduler entity due to the reduction of the number of rules, that will reduce the computational requirements for the evaluation of the inputs/antecedents through the whole RB. This achievement will additionally help to reduce the energy consumption not for the execution of tasks, but for the scheduling of tasks at the meta-scheduler entity as lesser time will be required for the scheduling process. Additionally, the reduction of the number of rules helps to achieve a higher interpretability, easing the understanding of the proposed strategy to follow for the meta-scheduling process.

Third, the creation of the renewable scenario provides additional simulation capabilities, including the simulation of renewable energy through two different approaches, the inclusion of batteries with renewable energy and a direct renewable energy proportion from the unlimited power source. Although this is just a simple first implementation, it allows the study of the renewable parameters at the advanced scheduling provided by Pittsburgh and Kasia, which helps to prepare the cloud systems for future datacenters that make use of renewable energy to properly manage it and prioritize the execution of tasks to datacenters with renewable energy availability. Moreover, it has been included the renewable power models, which allows to implement different behavior at the direct renewable proportion power source for further study, including up to this point four different models. To ease the implementation of newer renewable power model, the whole model system is built over a base renewable power model from which all the models will inherit, as occur with the scheduling, meta-scheduling algorithms and the entities.

Finally, a correct multiobjective optimization is included through the implementation of the Pareto efficiency over the Pittsburgh algorithm. As shown at *section 6.2.3*, this algorithm shows its potential thanks to its capability of providing multiple optimal solutions to be used to solve a specific problem. However, it is shown how, at this point, these provided results shows inconsistency, providing outstanding solutions under some scenarios and terrible solutions over others scenarios, although at first sight the capabilities of this algorithm looks promising and further research must be done at this field.

8. FUTURE GOALS

After the development of this master thesis several future goals of research and study have been detected which must be considered for future improvements over the developed simulator that are specified below.

First, the inclusion of the Pareto efficiency as a *MOP* solution to properly solve this type problems shows its potential, although lacks to provide a consistent behavior, falling far behind the classic scheduling algorithms under certain situations. The main reason for this situation can be related to the *Pareto ranking* methods which tries to solve the lack of a proper ranking/classification system within the Pareto efficiency. It has been detected that a basic ranking method as the used *AM2* may provide suboptimal solutions, making the Pittsburgh algorithm to be stuck at local optimal. It is proposed to make further study at this field to try and achieve better results that helps Pareto to provide multiple optimal solutions to problems.

Second, as already specified, the Kasia algorithm should provide better results than Pittsburgh as specified at [10], but as shown at the *results section*, this behavior has not been achieved. The main reason is, as specified at *5.2.5.2*, this algorithm relies on some weights that if not correctly configured for the specific problem to be solved, the provided results may be far from optimal. It is proposed to conduct further study on this problematic to solve it and shows the true Kasia algorithm potential, for which may be necessary the use of *Genetic* and *Swarm* optimization algorithms to find the best weight possible for this specific scenario.

Third, as shown, it has been included at the simulator the so-called renewable power models to simulate different renewable energy availability at the simulated datacenters. Up to this point only a few renewable power models are included which a limited capability aimed to the study of different scenarios at the meta-scheduling process, but it is of interest to include real renewable models. These real renewable models should be based on real renewable energy plants such as solar and wind ones. For this, it should be considered the location of this real plants, the historical energy generation depending on the day of the year and time of the day which will significantly raise the capabilities of the simulator, providing a more realistic scenario aimed to provide a better tool for the research of the renewable energy based scheduling process.

To conclude, the interpretability of the rules within the RBs must be considerably improved. Currently, the RB generated through the knowledge acquisition systems have a low understanding level. This problematic must be solved, as the generated solutions should provide ways to allow their comprehension by humans so that rather than simply obtaining a solution that works for the given problem, this solutions should also be provided in a way that allows readers to properly understand how this optimal solution are achieved through the provided RBs.

9. BIBLIOGRAPHY

[1] Amazon Web Services (AWS) - Cloud Computing Services, 2020. Amazon Web Services, Inc. [online]

[2] Microsoft Azure Cloud Products, Services, Solutions | Microsoft Azure, 2020. *Azure.microsoft.com* [online],

[3] Stadia | Build a new generation of games, 2020. *Stadia.dev* [online]

[4] JUDGE, John, et al. Reducing data center energy consumption. *Ashrae Journal*, 2008, vol. 50, no 11, p. 14.

[5] BUYYA, Rajkumar; BELOGLAZOV, Anton; ABAWAJY, Jemal. Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges. *arXiv preprint arXiv:1006.0308*, 2010.

[6] CORDÓN, Oscar, et al. Evolutionary tuning and learning of fuzzy knowledge bases. *Genetic fuzzy systems*, 2001, vol. 19.

[7] Cingolani, Pablo, and Jesús Alcalá-Fdez. *"jFuzzyLogic: a Java Library to Design Fuzzy Logic Controllers According to the Standard for Fuzzy Control Programming"*

[8] Cingolani, Pablo, and Jesus Alcala-Fdez. *"jFuzzyLogic: a robust and flexible Fuzzy-Logic inference system language implementation."* Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on. IEEE, 2012.

[9] SALOT, Pinal. A survey of various scheduling algorithm in cloud computing environment. *International Journal of Research in Engineering and Technology*, 2013, vol. 2, no 2, p. 131-135.

[10] GARCÍA-GALÁN, S.; PRADO, R. P.; EXPÓSITO, JE Muñoz. Fuzzy scheduling with swarm intelligence-based knowledge acquisition for grid computing. *Engineering Applications of Artificial Intelligence*, 2012, vol. 25, no 2, p. 359-375.

[11] MAULIK, Ujjwal; BANDYOPADHYAY, Sanghamitra. Genetic algorithm-based clustering technique. *Pattern recognition*, 2000, vol. 33, no 9, p. 1455-1465.

[12] SMITH, Stephen Frederick. A learning system based on genetic adaptive algorithms. 1980.

[13] EBERHART, Russell C.; SHI, Yuhui; KENNEDY, James. *Swarm intelligence*. Elsevier, 2001.

[14] VAN VELDHUIZEN, David A.; LAMONT, Gary B. Evolutionary computation and convergence to a pareto front. En *Late breaking papers at the genetic programming 1998 conference*. 1998. p. 221-228.

[15] PRADO, R. P., et al. On providing quality of service in grid computing through multi-objective swarm-based knowledge acquisition in fuzzy schedulers. International journal of approximate reasoning, 2012, vol. 53, no 2, p. 228-247.

[16] ALBERTO, I., et al. Multiobjective evolutionary algorithms. Pareto rankings. *Monografias del Seminario Matematico Garcia de Galdeano*, 2003, vol. 27, p. 27-35.

[17] GOLDBERG, D. E. Genetic Algorithms in Search. Optimization and Machine Learning. Addism1—Wesley. *Reading, MA*, 1989.

[18] KIM, Kyong Hoon; BUYYA, Rajkumar; KIM, Jong. Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. En *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*. IEEE, 2007. p. 541-548.

[19] PIGUET, Christian; SCHUSTER, Christian; NAGEL, J.-L. Optimizing architecture activity and logic depth for static and dynamic power reduction. En *The 2nd Annual IEEE Northeast Workshop on Circuits and Systems, 2004. NEWCAS 2004.* IEEE, 2004. p. 41-44.

[20] PALLIPADI, Venkatesh; STARIKOVSKIY, Alexey. The ondemand governor. En Proceedings of the Linux Symposium. 2006. p. 215-230.

[21] CALHEIROS, Rodrigo N., et al. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 2011, vol. 41, no 1, p. 23-50.

[22] CALHEIROS, Rodrigo N., et al. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *arXiv preprint arXiv:0903.2525*, 2009.

[23] GUÉROUT, Tom, et al. Energy-aware simulation with DVFS. *Simulation Modelling Practice and Theory*, 2013, vol. 39, p. 76-91.

[24] CHEN, Weiwei; DEELMAN, Ewa. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. En *2012 IEEE 8th International Conference on E-Science*. IEEE, 2012. p. 1-8.

[25] COTES-RUIZ, Iván Tomás, et al. Dynamic voltage frequency scaling simulator for real workflows energy-aware management in green cloud computing. *PloS one*, 2017, vol. 12, no 1.

[26] COTES-RUÍZ, ADRIÁN, 2020, Metaplanificador para cloud computing. *Tauja.ujaen.es* [online]. 2020. [Accessed 5 February 2020]. Available from: http://tauja.ujaen.es/handle/10953.1/5130

[27] jFuzzyLogic, 2020. *Jfuzzylogic.sourceforge.net* [online]

[28] PRADO, Rocío Pérez; MUNOZ EXPOSITO, Jose Enrique; GARCIA-GALAN, Sebastian. Flexible fuzzy rule bases evolution with swarm intelligence for meta-scheduling in grid computing. *Computing and Informatics*, 2015, vol. 33, no 4, p. 810-830.

[29] GARCÍA-GALÁN, Sebastián; PRADO, Rocío P.; EXPÓSITO, José Enrique Muñoz. Swarm fuzzy systems: knowledge acquisition in fuzzy systems and its applications in grid computing. *IEEE Transactions on Knowledge and Data Engineering*, 2013, vol. 26, no 7, p. 1791-1804.

[30] PRADO, R. P., et al. Knowledge acquisition in fuzzy-rule-based systems with particle-swarm optimization. *IEEE Transactions on Fuzzy Systems*, 2010, vol. 18, no 6, p. 1083-1097.