Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

METAPLANIFICADOR PARA CLOUD COMPUTING

Alumno: Adrián Cotes Ruiz

Tutor: Sebastián García Galán
Depto.: Ingeniería de Telecomunicación

Tutor: Luis Ramón López López
Depto.: Ingeniería de Telecomunicación

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

Curso 2016-2017

METAPLANIFICADOR PARA CLOUD COMPUTING

Alumno: Adrián Cotes Ruiz

Tutor: Sebastián García Galán

Departamento: Ingeniería de Telecomunicación

Tutor: Luis Ramón López López

Departamento: Ingeniería de Telecomunicación

Firma del autor

Firma del tutor

LOPEZ LOPEZ LUIS RAMON - 75070269V

Firmado digitalmente por LOPEZ LOPEZ LUIS RAMON - 75070269V
Nombre de reconocimiento (DN): c=ES, serialNumber=75070269V, sn=LOPEZ LOPEZ, givenName=LUIS RAMON, cn=LOPEZ LOPEZ LUIS RAMON - 75070269V
Fecha: 2017.06.23 09:23:32 +02'00'

Contenido

INDICE DE FIGURAS

INDICE DE TABLAS

1. RESUMEN

En este trabajo fin de grado se ha desarrollado un simulador de Meta-Planificación mediante el cual se pueda realizar la unificación y centralización de múltiples *datacenters* con sus respectivos planificadores locales.

Así mismo, se han implementado cuatro algoritmos de Meta-Planificación, que establecen la estrategia de reparto de tareas a los distintos *datacenters*. Tres de ellos se basan en estrategias clásicas de planificación: MaxMin, MinMin y Round Robin. El cuarto hace uso de en un sistema basado en reglas borrosas (*fuzzy rule based system* o FRBS). El uso de este último algoritmo persigue la reducción y optimización de los consumos eléctricos producidos por el procesamiento de tareas.

Además, se procede a mostrar los márgenes de mejora en tiempo y consumo energético, respecto a las estrategias clásicas de planificación.

2. INTRODUCCIÓN

En este trabajo fin de grado se ha construido un planificador jerárquico (meta-planificador) que será utilizado para planificar tareas a otros planificadores locales. Como no se dispone de cientos de ordenadores, se ha usado una herramienta de simulación llamada WorkFlowSim, basada en el simulador CloudSim y programada en Java. Físicamente, el planificador jerárquico propuesto está constituido por una colección de clases en Java que se han integrado en la plataforma de simulación.

Usando WorkFlowSim, se han definido distintos centros de procesamiento de datos (CPD), también llamados *datacenters*. Están formados por un conjunto de nodos que ejecutan máquinas virtuales y es en ellas donde se realiza el procesamiento de tareas. Las tareas asignadas a un *datacenter* serán planificadas por medio de un planificador local, con el fin de conseguir una correcta asignación siguiendo criterios de mínimo tiempo de ejecución o mínima potencia consumida. El meta-planificador es el encargado de realizar la asignación de todas las tareas a los distintos *datacenters* a través de sus planificadores locales, en función del estado de los mismos.

2.1. Motivación del TFG

La tendencia de los últimos años [1] muestra como la demanda de servicios proporcionados por los CPD aumenta. Cada vez se realiza un mayor gasto e inversión en el desarrollo e implementación de nuevos centros de proceso, requiriendo en algunos casos, la realización de una continua y gran inversión en este campo para disponer de mayores prestaciones. Esto provoca una obsolescencia prematura de los equipos.

Para paliar esta situación, se propone el empleo de un Meta-Planificador y distintos centros de proceso. De esta forma se consigue la capacidad requerida por algunas tareas que necesiten mayores recursos computacionales: distintos centros pueden trabajar en equipo y son gobernados por un Meta-Planificador. Éste utiliza el estado de los *datacenter* para decidir donde enviará la tarea. De esta manera no solo se consigue reutilizar equipos que de otra manera sería descartados, además se obtiene mayor potencia de cómputo al disponer de un número mayor de equipos que ejecuten tareas.

Al utilizar un meta-planificador se consigue que empresas e instituciones que no disponen de las instalaciones suficientes puedan utilizar distintos *datacenters* (privados o públicos), permitiéndoles usar una infraestructura para cubrir sus necesidades

computacionales. Esto también supone un ahorro económico: no es necesario adquirir nuevos equipos para una necesidad puntual.

2.2. Estructura del documento

Este documento consta de una serie de capítulos en los que se detallan el proceso de diseño e implementación del Meta-Planificador. En concreto, la distribución del contenido del mismo es la que se detalla a continuación:

- **Capítulo 1. Resumen**. Se relacionarán de manera muy sintética los objetivos de este trabajo fin de grado.
- **Capítulo 2. Introducción**. En este capítulo se establece el contexto en el que se encuadra la necesidad que se desea satisfacer y las soluciones adoptadas para solventar el problema de partida.
- **Capítulo 3. Objetivos**. Se detallarán los fines que se desean conseguir con este TFG.
- **Capítulo 4**. **Materiales y métodos**. Recoge toda la información relativa al proceso de diseño del trabajo: las tecnologías y técnicas utilizadas, modelos de arquitectura adoptados y fragmentos de código relevantes de la implementación.
- **Capítulo 5. Resultados y Discusión**. Se especificarán los distintos resultados obtenidos tras la finalización de este trabajo, así como tiempos de ejecución y energía consumida con las distintas estrategias de planificación.
- **Capítulo 6. Conclusiones.** En este capítulo se exponen las conclusiones extraídas a lo largo del desarrollo de este trabajo. También se describen los conocimientos y destrezas adquiridas durante el proceso.
- **Capítulo 7. Líneas futuras.** Se mencionarán algunas líneas de trabajo futuras.
- **Capítulo 8. Bibliografía**. Reseñas a los libros, documentos online, revistas y sitios web que se han consultado durante la realización del TFG.

3. OBJETIVOS

El principal objetivo de este proyecto es la realización de un montaje e implantación de un meta-planificador con una estrategia de planificación alternativa a los sistemas de planificación clásicos. Junto a él, operan un conjunto de planificadores locales existentes en cada uno de los centros de procesamiento de datos. Permite la planificación de varios CPD de manera centralizada.

En este proyecto se han estudiado e implementado diferentes algoritmos de meta-planificación basados en los algoritmos clásicos de planificación. Además se ha procedido a implementar un sistema basado en reglas borrosas (FRBS, *Fuzzy Rules Based System*), el cual permite una mejora en la planificación de tareas.

Es un objetivo presentar los resultados de estas estrategias de planificación para demostrar la bondad de los sistemas de planificación de tareas basados en reglas borrosas y su aplicación a la meta-planificación.

4. MATERIALES Y MÉTODOS

Este capítulo está dividido en dos partes: en la primera (capítulo 4.1) se presenta un estado del arte con algunos conceptos relativos a controladores borrosos, *cloud computing*, modelo de potencia y virtualización. En la segunda parte se recoge el trabajo realizado, el cual ha sido dividido en dos fases, una primera de estudio (capítulo 4.2) y aprendizaje de los simuladores empleados, y una segunda fase (capítulo 4.3) donde se recoge el desarrollo llevado a cabo.

A modo resumen, este capítulo está dividido en los siguientes apartados:

- 4.1 Estado del arte: se detallan un conjunto de tecnologías empleadas para proporcionar unos conocimientos necesarios para facilitar la comprensión de la segunda parte de este capítulo.
- 4.2 Primera fase: Aprendizaje de los simuladores empleados: se introducen y explican los simuladores estudiados y el simulador empleado como base para la implementación del Meta-Planificador.
- 4.3 Implementación del Meta-Planificador: se explica detalladamente el proceso de desarrollo llevado a cabo en este proyecto.

EXPLICA ESTO

4.1. Estado del arte

4.1.1. *Lógica borrosa*

Antes de la aparición de la lógica borrosa (*Fuzzy Logic* - FL) [2], sistemas básicos de bases de reglas eran empleados para la solución de problemas, los cuales se basaban en sentencias condicionales con valores precisos y concretos. Estos primeros sistemas, poseían la incapacidad de poder resolver problemas que incorporaran la lógica humana, ya que podía incluir imprecisiones. Un ser humano puede expresar valores imprecisos para, por ejemplo, la temperatura ambiente o el nivel de ruido, tales como poco, moderado, bastante, mucho, etc. Un sistema de reglas clásico no puede trabajar con estos valores de imprecisión. De esta manera, mientras en los sistemas clásicos se trabajaba con pertenencias que otorgaban un resultado de verdadero o falso, estos nuevos sistemas borrosos permiten diferenciar entre un rango de pertenencia variable.

Con objeto de tratar de explicar la FL de manera más clara, se expone el siguiente ejemplo sobre rango de edades. Para diferenciar a una persona en función de su edad, podemos dividirla en tres rangos: joven, adulta, o mayor. Esta será la entrada de información del sistema, de manera que podamos calificar los rangos de edades de la siguiente forma:

Clasificación	**Edad**
Joven	$< 30\ a\tilde{n}os$
Adulto	$\geq 30\ a\tilde{n}os\ y\ < 60\ a\tilde{n}os$
Persona mayor	$\geq 60\ a\tilde{n}os$

Tabla 1: Calificación de edades

Tal y como se muestra en la tabla anterior, basándonos en la lógica clásica, se tendría que si una persona se encuentra en el rango de edad menor a 30 años, será estrictamente joven, además de que se calificará a una persona de tanto 20 años como de 29 años como jóvenes. Por otra parte, con este sistema que funciona mediante condiciones de verdadero y falso, se tendrá que una persona con 29 años es joven, y si esta, al día siguiente alcanzase los 30 años de edad, pasaría repentinamente a ser estrictamente adulto. Tal y como se puede observar, este sistema clásico cuenta con una gran imprecisión, debido a que se está indicando que tanto una persona de 20 como de 29 años son jóvenes, no diferenciando que la persona de 20 años es más joven que la persona de 29 años, así como que la persona que alcance los 30 años pasará a ser repentinamente adulta, cuando en la realidad, cabe esperar que, de manera lógica, dicha persona seguirá aun, en cierto grado, perteneciendo a la calificación de persona joven, pero teniendo una mayor y fuerte pertenencia a la calificación de persona adulta, lo que primará a esta de ser calificada como persona adulta, pero no privando a esta de ser reconocida como aún una persona joven en cierta medida.

Con objeto de solventar el problema citado, se creó la lógica borrosa, la cual es capaz de definir diferentes niveles o grados de pertenencia referentes a dichas calificaciones. De esta manera, una persona de 20 años tendrá un mayor nivel de pertenencia a la calificación joven que una persona de 29 años de edad, además de que, para el caso de esta, en parte tendrá un cierto nivel de pertenencia a la calificación de adulto, de manera que se evitan cambios bruscos en la pertenencia a diferentes calificaciones de edades,

de manera que una persona, a medida que vaya avanzando en edad, por ejemplo, en este caso de la persona de 29 años de edad, irá gradualmente perteneciendo menos a la calificación de joven, y perteneciendo gradualmente en mayor medida a la calificación de adulto, llegando el punto en el que, una vez dicha persona alcance los 30 años de edad, su pertenencia a la calificación de adulto será mayor a la calificación de joven, y, a medida que dicha persona vaya alcanzando una mayor edad, su nivel de pertenencia a joven se verá reducida hasta no poseer pertenencia alguna a dicho rango de edades, mientras que progresivamente verá su pertenencia a adulto aumentada. Se observa que, gracias a la FL, se permite un tipo de lógica que por otra parte no sería posible con los sistemas de reglas clásicos.

En los sistemas de FL, las clasificaciones tales como joven, adulto y persona mayor son llamadas funciones de pertenencia (MF – *membership function*). Los MF pueden ser presentados de una gran variedad de formas, permitiendo la representación de estas como gaussianas, formas triangulares, piramidales, trapezoidales, o incluso estableciendo su variación por puntos a lo largo del espacio de muestras. No obstante, se debe tener en cuenta que esta capacidad de representar las MF de diferentes formas dependerá principalmente del sistema de fuzzyficación empleado, algunos sistemas tales como Matlab cuentan con una gran variedad de formas de representar las MF, al igual que jFuzzyLogic [4], que también cuenta con una amplia capacidad de definición de diferentes tipos de MF, el cual ha sido utilizado en este proyecto.

A modo de ejemplo, se expone como sería gráficamente las funciones de pertenencia generadas por medio de jFuzzyLogic. Además, se adjunta el código necesario para definir este dato de entrada (antecedente) para este sistema borroso de la edad:

```
FUZZIFY edad
    TERM joven := (0, 1) (20, 1) (35, 0) ;
    TERM adulto := (20, 0) (35,1) (50,1) (65,0);
    TERM persona_mayor := (50, 0) (65, 1) (100, 1);
END_FUZZIFY
```

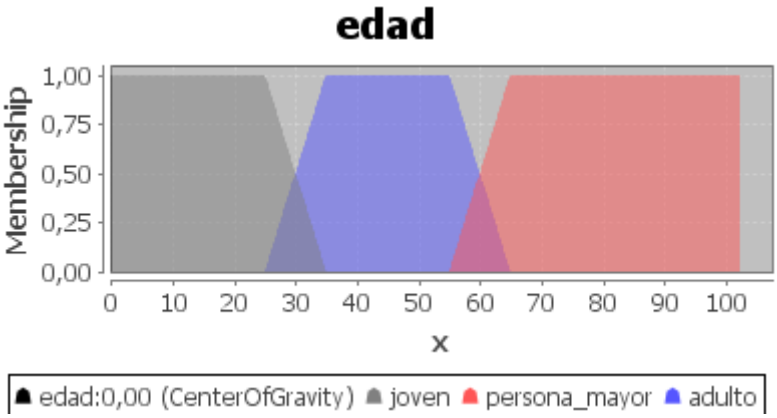

Figura 1: Antecedente de edad

Tal y como se muestra tanto en el fragmento de código como en la Figura 1, se observa como, por ejemplo, cuando se alcanza el rango de edad de 25 años, progresivamente comienza a decrecer el grado de pertenencia a la función de pertenencia joven, mientras empieza a aumentar el nivel de pertenencia a la MF adulto. De igual manera, se observa que esto también sucede para la transacción de pertenencia desde la MF adulto a persona_mayor.

Por otra parte, se debe tener en cuenta que el ejemplo anteriormente expuesto es del tipo aclaratorio, debido a que en los sistemas borrosos reales las funciones de pertenencia no son implementadas como se ha expuesto anteriormente, si no que el espacio de valores para tanto los datos de entrada (antecedentes) como para las salidas (consecuentes), se expresan en el rango de valores de 0 a 1, o bien, de -1 a 1, dependiendo del problema a solventar. Esto es debido a que tal y como se puede observar, si para la solución de un problema particular, por ejemplo, en el caso de la edad, esta se implementa en un rango real de edades, por ejemplo, de 0 a 100, implicaría que se tendría implementado un sistema el cual podría dar la solución a la calificación de una persona en los grupos de joven, adulto, y persona mayor en función de su edad siempre y cuando la edad de dicha persona no sobrepase el límite de edad impuesto de 100 años, ya que de lo contrario, si este rango de edad fuera superado, el valor se consideraría fuera de rango, no otorgando ningún valor en la salida. Debido a que, por ejemplo, en el caso de la edad (esperanza de vida), se espera que con el paso de los años, se puedan alcanzar edades mayores a los 100 años, implicaría que este sistema diseñado con dicho límite de 100 años no sería válido, pues una persona con una edad mayor a 100 años no sería calificada perteneciente a ningún grupo, de manera que, para solucionar esto, se trabajan con valores normalizados. Mediante la estrategia de los valores normalizados, se consigue solventar el problema anteriormente expuesto, de manera que el sistema trabajará con valores de entrada comprendidos entre 0 y 1, de manera que si los datos de

entrada con los que el sistema debe de trabajar son comprendidos entre 0 a 80 años, bastará con identificar el valor máximo de edad y, una vez identificado, dividir toda edad que se vaya a introducir como dato de entrada en el sistema por dicha edad máxima, para así conseguir el valor normalizado comprendido entre 0 y 1. Tal y como se observa, la bondad de trabajar con datos normalizados permite adaptar el sistema cómodamente ante cualquier rango de valores de entrada y de salida, bastando simplemente con realizar el proceso de normalización anteriormente expuesto.

No obstante, una vez explicado lo anterior, lo cual es una primera y básica aproximación de los sistemas de lógica borrosa, cabe destacar que estos FRBS están constituidos por más componentes que los expuestos anteriormente, siendo los antecedentes, consecuentes, y las MF, sólo una parte de estos sistemas. Entre sus componentes, podemos citar la existencia del Fuzzyficador, Defuzzyficador, motor de inferencia y la base de conocimiento (KB – *Knowledge Base*), la cual está formada a su vez por la base de datos y la base de reglas. Cabe destacar que todos y cada uno de estos componentes están relacionados entre sí, de manera que el correcto funcionamiento del sistema completo está condicionado a un correcto funcionamiento de todos y cada uno dichos elementos.

La estructura de un FRBS y la relación entre cada uno de sus componentes es tal como la mostrada en la Figura 2.

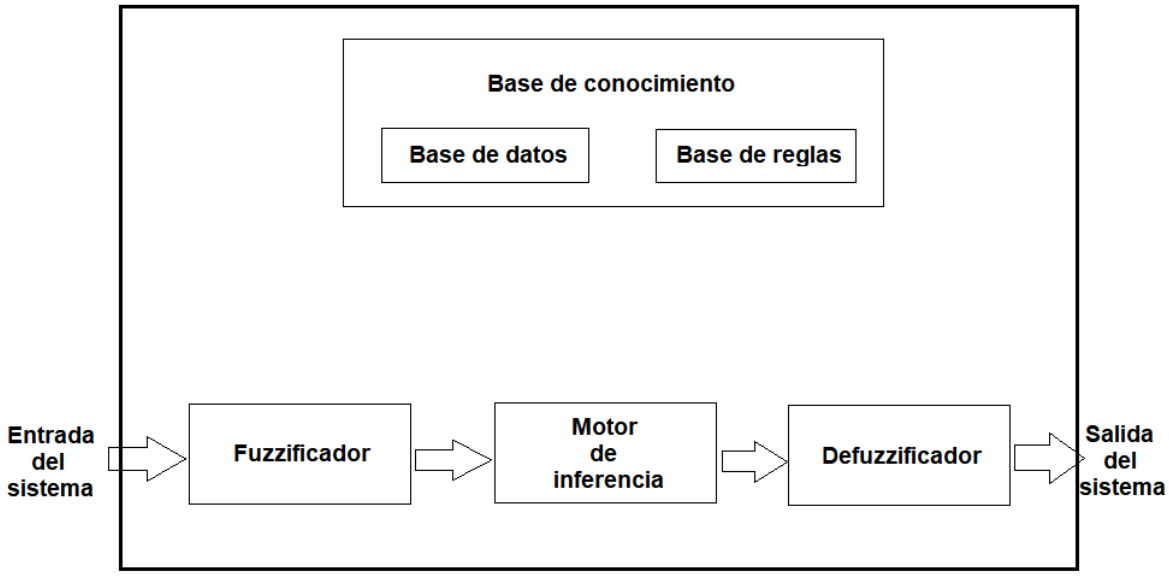

Figura 2: Estructura de un FRBS

A continuación, se definirán todos y cada uno de los componentes, y se tratará de explicar cómo interaccionan cada uno de los componentes del FRBS entre sí.

- Fuzzyficador:

El fuzzyficador constituye la primera etapa de un FRBS. El objetivo de este elemento del sistema es el de trabajar con las variables de entrada del sistema, las cuales, como ya ha sido especificado anteriormente son llamadas antecedentes. Dichos antecedentes estarán definidos mediante variables de entrada en un rango de valores reales, de manera que, como el motor de inferencia trabajará con valores normalizados (comprendidos entre 0 a 1), será necesario que estos sean normalizados previamente, proceso el cual es llevado a cabo en el fuzzyficador.

- Base de conocimiento:

Este componente está formado por la base de datos y la base de reglas. Mediante esta base de conocimiento (KB – *Knowledge Base*), se establecerá una relación entre las variables de entrada (antecedentes) y la o las variables de salida (consecuentes). La KB será utilizada por el motor de inferencia para obtener dicho consecuente en función de los antecedentes y el conocimiento almacenado en esta para la resolución de un tipo de problema específico.

Por lo general, la mayoría de estos sistemas utilizan una estructura basada en múltiples entradas y una única salida (MISO), aunque esto no implica que no puedan ser utilizados otros sistemas los cuales cuenten con múltiples salidas (MIMO).

- Base de datos:

La base de datos constituirá la parte de la KB en la cual se definirán las diferentes funciones de pertenencia para todos y cada uno de los diferentes antecedentes. Mediante estas funciones de pertenencia, el rango de valores de cada uno de los antecedentes (de 0 a 1 al tratarse de valores normalizados) será dividido en varias clasificaciones de manera que, en función del valor de dicho antecedente y de la base de datos, el motor de inferencia será capaz de obtener el nivel de pertenencia de dicha variable a cada función de pertenencia. Además, en este elemento no sólo se define el rango que cubre cada una de las funciones de pertenencia, sino también la forma de estas.

- Base de reglas:

La base de reglas está constituido por todas y cada una de las reglas las cuales serán utilizadas por el motor de inferencia en el proceso de inferencia para hallar, en función de los antecedentes, el valor del o de los consecuentes. Las reglas estarán definidas por medio de sentencias condicionales en las cuales obtendrá y observará el nivel de pertenencia de los antecedentes para obtener así el valor del consecuente. Además, se debe tener en cuenta que debido a la capacidad de los antecedentes de pertenecer en

mayor o menor medida a más de una función de pertenencia, cabe la posibilidad de que un conjunto de antecedentes cumplan las condiciones de más de una regla para obtener el resultado del consecuente. Por otra parte, pueden existir múltiples reglas en el que la condición de sus antecedentes sea la misma, pero variando sólo y exclusivamente el valor del consecuente.

Por otra parte, se debe tener en cuenta que debido a la naturaleza de estos sistemas, si se quiere obtener un buen resultado por medio del FRBS, será necesario definir una cantidad de reglas lo suficientemente amplia para que cubra todos o la mayoría de los casos de interés o que se puedan dar en nuestro sistema borroso, para obtener un buen resultado. Por otro lado, si esto no es llevado a cabo, propiciará que, bajo ciertas circunstancias, la solución expuesta sea mala o nula.

Por último, las reglas pueden ser definidas de dos formas diferentes, mediante condiciones AND o condiciones OR. En el caso de los definidos con la condición AND, el funcionamiento es tal que, de todos los antecedentes, se tendrá en cuenta el valor del mínimo de todos ellos para obtener el consecuentes, mientras que en el caso de aquellos definidos con OR, se tendrá en cuenta el mayor de los valores de los antecedentes para obtener el valor del consecuente.

- Motor de inferencia:

Este componente tendrá la función de, por medio de la base de conocimiento y los antecedentes, obtener el valor del consecuente. Este sistema funcionará de tal manera que, para cada uno de los antecedentes, comprobará su nivel de pertenencia a cada una de las funciones de pertenencia, de manera que, posteriormente, evaluando todas y cada una de las reglas que constituyen la base de reglas, será capaz de obtener un valor a la salida del sistema (consecuente)

- Defuzzyficador:

La función de este elemento es la de, una vez el motor de inferencia mediante el proceso de inferencia ha obtenido un valor para la salida del sistema, el cual será un valor normalizado, ajustar este valor borroso a un valor real tal que se adecue a las características del problema a resolver.

Con objeto de facilitar la comprensión de los FRBS, a continuación se expone un ejemplo el cual es resuelto mediante estos sistemas. El problema consiste en un sistema el cual debe de mover un vehículo de manera que este alcance un punto concreto, siendo la solución perfecta aquella en la que el coche alcanza dicha localización sin sobrepasar este. En caso de que no se alcance, la solución no será perfecta. En caso de que se

supere dicho punto, se considerará que el vehículo se ha siniestrado. Además, el vehículo no podrá oscilar yendo hacia delante y hacia atrás hasta alcanzar el objetivo óptimo de parada.

El primer paso para solventar dicho problema, pasa por identificar los antecedentes y consecuentes del sistema, para así, de esta manera, definir con qué variables de entrada y de salida se debe trabajar. En primer lugar, el vehículo simplemente debe de recorrer una distancia en línea recta, es decir, este no deberá realizar ningún giro, por lo tanto, se identifica que el sistema sólo deberá considerar como variables de entrada, la distancia del vehículo en cada momento, y la velocidad de este. Por otra parte, el vehículo para poder desplazarse, deberá contar con una determinada velocidad, de manera que, en función de los antecedentes anteriormente encontrados, el vehículo deberá aumentar o disminuir su velocidad, de manera que se identifica que el consecuente consistirá en la aceleración de este.

Una vez identificados los antecedentes y los consecuentes, se debe identificar si estos poseerán sólo valores positivos, o si también poseerán valores negativos. Para el caso de los antecedentes, se tiene que estos poseerán un valor positivo, debido a que, en el caso de la distancia, esta será positiva, indicando que está en una posición lejana al punto de parada, o 0, indicando que se ha alcanzado el punto de parada, mientras que la velocidad será, o bien 0, indicando que el vehículo permanecerá detenido, o positivo, indicando que el vehículo permanecerá en movimiento. Por otra parte, se tiene que el vehículo deberá ser capaz de modificar su velocidad, el cual estará representado por el consecuente aceleración, pero además, se debe tener en cuenta que el vehículo deberá poseer la capacidad de desacelerar, de manera que, se identifica que el sistema, a priori, necesitaría originalmente de dos consecuentes, aceleración y frenada. Debido a que trabajar con más de un consecuente puede incrementar el grado de dificultad del sistema, se puede observar que estos pueden ser compactados a uno único llamado aceleración, que podrá contar tanto con valores positivos en caso de acelerar, o valores de aceleración negativa, para permitir el proceso de frenado.

Una vez identificado el número de antecedentes y de consecuentes, se debe tomar la decisión de cuantas funciones de pertenencia deberán poseer estos.

Para los antecedentes, se pueden definir tres funciones de pertenencia para cada uno de estos, siendo estos cerca, media y lejana para la distancia, y lenta, normal y rápida para la velocidad.

Por otra parte, se cuenta con un único consecuente, siendo este la aceleración del vehículo. Se debe tener en cuenta que con motivo de obtener una mayor resolución o

precisión en las decisiones a tomar, es recomendable que este posea un mayor número de funciones de pertenencia que los antecedentes, de manera que, por ejemplo, en este caso, dado que han sido definidas 3 funciones de pertenencia para los antecedentes, para el caso del consecuente será recomendable utilizar 5 funciones de pertenencia. Estas funciones de pertenencia, para el caso del consecuente de aceleración, recordando que este debe poseer la capacidad de indicar si el vehículo debe frenar o acelerar, podrán ser definidos tal que se tengan los antecedentes frena mucho, frena, mantener, acelera y acelera mucho, contando así con dos funciones de pertenencia para el proceso de frenado, dos para el proceso de aceleración, y una restante para el proceso de mantener una velocidad constante.

A modo resumen, a continuación se muestra la Tabla 2 sobre los antecedentes y consecuentes.

Antecedentes		**Consecuentes**
Distancia	Velocidad	Aceleración
Cerca	Lenta	Frena mucho
Media	Normal	Frena
Lejana	Rápida	Mantener
		Acelera
		Acelera mucho

Tabla 2: Antecedentes y consecuentes

A continuación, se muestra en las siguientes figuras adjuntas como quedarían definidas las funciones de pertenencia en cada uno de los antecedentes y consecuentes, tal y como se muestra en las figuras Figura 3, Figura 4 y Figura 5.

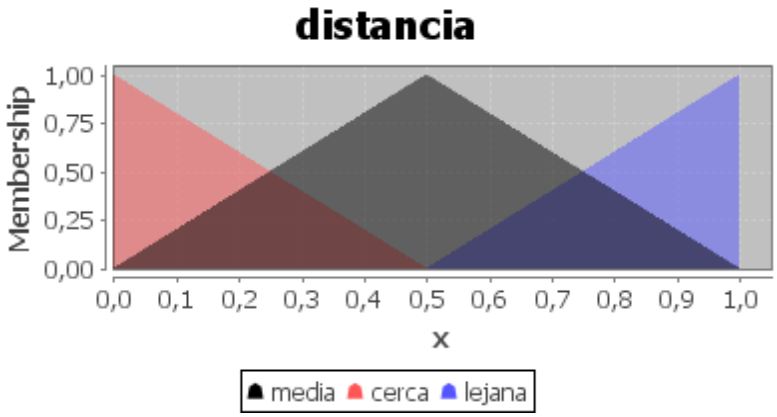

Figura 3: Antecedente distancia

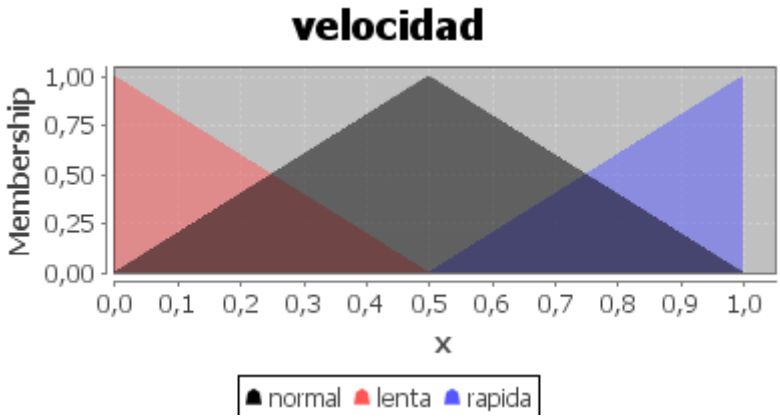

Figura 4: Antecedente velocidad

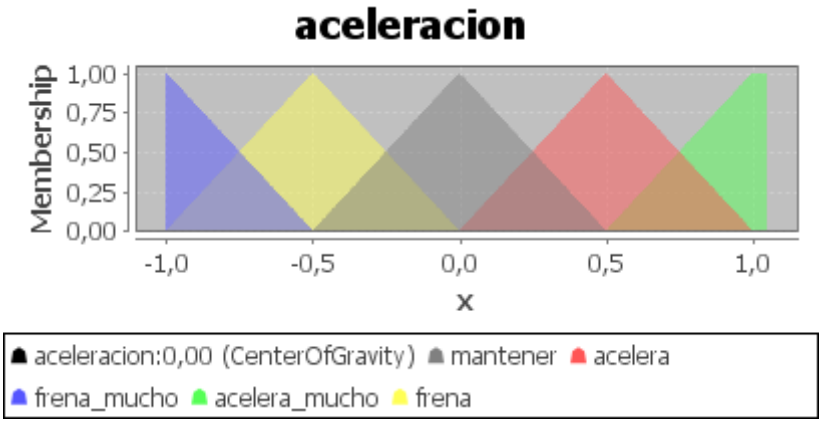

Figura 5: Consecuente aceleración

Por último, una vez definidos los antecedentes, consecuentes y sus respectivas funciones de pertenencia, será necesario definir el conjunto de reglas que constituyan la base de reglas del sistema FRBS que sea capaz de solventar el problema acogiéndonos a las características de este. Teniendo en cuenta que este sistema debe ser capaz de desplazar un vehículo el cual está detenido en un determinado punto, y hacer que este alcance el punto de destino sin sobre pasar este, se debe definir un conjunto de reglas tal

que cubra la mayor parte de los casos los cuales pueden suceder, para evitar que pueda ocurrir el caso en el que no haya un consecuente para una determinada combinación de antecedentes, ya que esto producirá que para dichas entradas no haya una acción de respuesta, lo que propiciará que se obtengan malos resultados de este sistema borroso. A continuación, se adjunta una base de reglas como primera aproximación a buscar una solución a este problema:

1. IF distancia IS cerca AND velocidad IS lenta THEN aceleración IS mantener
2. IF distancia IS cerca AND velocidad IS media THEN aceleración IS frena
3. IF distancia IS cerca AND velocidad IS rápida THEN aceleración IS frena_mucho
4. IF distancia IS media AND velocidad IS lenta THEN aceleración IS acelerar
5. IF distancia IS media AND velocidad IS media THEN aceleración IS mantener
6. IF distancia IS media AND velocidad IS rápida THEN aceleración IS frena
7. IF distancia IS lejana AND velocidad IS lenta THEN aceleración IS acelera_mucho
8. IF distancia IS lejana AND velocidad IS media THEN aceleración IS acelera
9. IF distancia IS lejana AND velocidad IS alta THEN aceleración IS mantener

Una vez definida la base de reglas, se debe comprobar de manera exhaustiva que esta otorga un buen resultado, es decir, que proporciona una solución óptima o cerca a óptima para el problema en cuestión. Además se debe tener en cuenta que el sistema trabajará tal y como se muestra en las figuras Figura 3, Figura 4 y Figura 5, con valores normalizados, de manera que este FRBS se podrá adaptar para cualquier entorno mediante la modificación de los valores máximos de los antecedentes y del consecuente, es decir, estableciendo la distancia de inicio a la que se encuentra el vehículo del punto en el que este se debe detener, la velocidad máxima que este puede alcanzar, así como la aceleración máxima que este puede alcanzar. No obstante, se debe tener en cuenta que una vez encontrada una base de reglas la cual proporcione una buena solución al problema en cuestión bajo unas determinadas condiciones de distancia, velocidad y aceleración máxima, no implica que esta deba proporcionar dicho mismo óptimo resultado si las condiciones del sistema son modificadas, por lo que es recomendable, una vez modificadas las condiciones del sistema, volver a evaluar esta para observar su comportamiento, y si este no es óptimo, será necesario volver a crear un conjunto de reglas tal que el resultado para dichas nuevas condiciones del problema sean óptimas.

Tal y como se puede observar, el proceso de búsqueda y evaluación exhaustiva de una base de reglas que proporcione un resultado óptimo para un determinado constituye la parte más compleja de un FRBS, especialmente si las condiciones del sistema son modificadas, lo cual incrementa considerablemente, la ya de por sí compleja tarea de

búsqueda de una base de reglas óptima. Con objeto de solventar el problema anteriormente expuesto, surgieron los sistemas de aprendizaje para FRBS, de manera que estos, mediante una búsqueda exhaustiva y automática, son capaces de encontrar, para unas condiciones concretas del problema en cuestión, una base de reglas la cual proporcione una solución óptima o cercana a óptima. Entre los diferentes sistemas de aprendizaje, son de destacar los algoritmos *Pittsburgh Approach* [20] y KASIA (*Knowledge Acquisition with a Swarm Intelligence Approach*) [4].

Pittsburgh Approach, consiste en un sistema de aprendizaje el cual tratará de, buscar una base de reglas óptima mediante la cual solventar el problema. Como principal característica de este sistema de aprendizaje se puede destacar el uso de un sistema de optimización basado en algoritmos genéticos [5] constituidos por una población, donde cada uno de sus miembros será una base de reglas completa y, mediante procesos de cruce y mutación, se procederá a iterativamente realizar la mezcla o unión de reglas obtenidas de varias bases de reglas (proceso de cruce) y a generar nuevas mediante los procesos de mutación.

KASIA, al igual que Pittsburgh, se trata de un algoritmo de búsqueda de una base de reglas óptimas, con la diferencia de que este, en lugar de estar basado en algoritmos genéticos, está basado en el algoritmo de optimización PSO (*Particle Swarm Optimization*) [12], formado por un conjunto de partículas que constituyen la población del sistema, realizarán la búsqueda de una solución óptima a un problema particular mediante la compartición de información. Cada una de estas partículas poseerán una base de reglas completas, tal y como ocurre en Pittsburgh.

La principal ventaja de utilizar algoritmos de aprendizaje es la capacidad de adaptarse para las características concretas de cada tipo de problema, evitando así tener que realizar manualmente la exhaustiva tarea de búsqueda de una base de reglas óptima.

4.1.2. *Tipos de Cloud computing*

En lo referente a *cloud computing*, existen una serie de tipos los cuales deben ser diferenciados entre sí. Todos los tipos están definidos como servicios, de manera que, globalmente, son definidos como XaaS (*X as a Service* – cualquier cosa como servicio). Cada tipo se diferenciará principalmente por el nivel de permisos y complejidad que recaerá sobre el usuario final, de manera que esté necesitará de un mayor o menor nivel de conocimientos dependiendo del que precise para su situación particular.

Los diferentes tipos de *cloud computing* quedan diferenciados por un sistema de niveles, en el cual los de más alto nivel requerirán unos conocimientos y

responsabilidades mínimas por parte del usuario final, mientras que los de más bajo nivel requerirán un mayor nivel de conocimientos y de responsabilidad por parte del usuario final, debiendo estos contar con un mayor nivel de conocimientos debido a que dispondrán de un mayor nivel de permisos para para poder gestionar estos tal y como el usuario precise. A continuación, se exponen los diferentes niveles de *cloud*:

- Nivel de aplicación:

Los *cloud* de este nivel están basados en el tipo SaaS (*Software as a Service* – aplicación como un servicio), que como su nombre indica, se encarga de ofrecer una aplicación o conjunto de aplicaciones como un servicio, siendo estos ejecutados remotamente en lugar de en las máquinas físicas de la unidad corporativa o en un servidor interno. Como principal ventaja de SaaS, es la falta de necesidad de conocimientos sobre la instalación y mantenimiento de dicha aplicación por parte del usuario o empresa final, de manera que será el proveedor de *cloud* contratado el encargado de gestionar la instalación, configuración y mantenimiento de este tipo de aplicación o servicio, haciéndose carga del correcto funcionamiento de este a cambio de un pago o tasa por parte del cliente.

- Nivel de plataforma:

Los *cloud* de este nivel están basados en el tipo PaaS (*Platform as a Service* – plataforma como un servicio). Este nivel otorga un mayor control al usuario final, de manera que el usuario final tendrá el control total del sistema operativo. De esta manera, en lugar de poseer un sistema el cual provee una aplicación o serie de aplicaciones, el usuario final tendrá el control total del sistema operativo, permitiendo así realizar tareas más complejas, como añadir nuevos servicios software, y poseer la responsabilidad en cuanto a la configuración y correcto funcionamiento de dichos software. A pesar de requerir al usuario final un mayor nivel de conocimientos, el servidor sobre el que se ejecuta (hardware) sigue siendo responsabilidad del proveedor de este servicio, por lo que será este el que se encargue de su correcto funcionamiento, logrando así que el usuario final tenga una menor cantidad de responsabilidades, pero otorgando a este un mayor control sobre los servicios a ofertar dado que poseerá el control completo del sistema operativo. No obstante, en este sistema *cloud*, el usuario final no tendrá la capacidad de elegir sobre qué máquina será ejecutado dicho sistema, ni de cuántos recursos dispondrá su sistema para la realización de las tareas requeridas, de manera que dicho sistema operativo será ejecutado en una máquina virtual, en un equipo elegido por el proveedor de servicios y otorgando este las capacidades computacionales requeridas para el correcto funcionamiento de este sistema contratado por el usuario.

- Nivel de infraestructura:

Los *cloud* de este tipo están basados en IaaS (*Infraestructure as a Service* – Infraestructura como un servicio). Este nivel proporciona aún más privilegios al usuario final, otorgando a este la capacidad de elección de dónde se deberán ejecutar sus máquinas virtuales, y cuantos recursos serán cedidas a estas de los disponibles en los equipos físicos, aumentando así la responsabilidad del usuario final en su correcto funcionamiento. Este sistema cuenta con la principal ventaja de permitir que el usuario final decida la infraestructura de máquinas virtuales que más se ajuste a sus necesidades computacionales, permitiendo a este realizar las modificaciones oportunas para alcanzar el nivel de rendimiento deseado. Asimismo, el usuario final deberá tomar las medidas oportunas para garantizar el correcto funcionamiento de dichas máquinas virtuales, mientras que en referencia al hardware, el proveedor del servicio deberá asumir su mantenimiento, reparación en caso de avería, y actualización en caso de requerir más recursos.

- Nivel hardware:

Este tipo de *cloud* están basados en HaaS (*Hardware as a Service* – hardware como un servicio). En este nivel la responsabilidad por parte del usuario final es casi completa, debido a que el proveedor del *cloud* se limitará sólo y exclusivamente a ofrecer el hardware contratado el cual tendrá que ser instalado por parte del usuario final en su propia infraestructura, recayendo en el proveedor sólo la responsabilidad del mantenimiento y sustitución del producto en caso de avería. De esta manera, el usuario final tendrá total responsabilidad en lo referente a alimentación de los equipos, interconexión de estos, y sistemas de refrigeración.

Como principal ventaja de este tipo de *cloud*, el usuario final no tendrá que preocuparse del mantenimiento del hardware al recaer esto sobre el proveedor. Además, si este requiriera de nuevos y más potentes equipos, o en caso de disponer de espacio suficiente para instalar más equipos, requiriera de más potencia, el proveedor podría ofrecerle dichos nuevos o extras recursos para cubrir las necesidades computacionales de este por medio de un aumento en el pago del servicio contratado. De igual manera, si sucediera el caso en el que el usuario final necesitara menos servicios a los contratados inicialmente, podrá solicitar al proveedor la modificación del servicio contratado, suponiendo un ahorro en costes económicos referentes al servicio contratado y al consumo energético.

Por otra parte, además de los tipos previamente definidos, existen otros los cuales dependen del tipo de recursos participantes, es decir, de la fuente que otorga los recursos al *cloud*, pudiendo diferenciarse tres tipos, públicos, privados, e híbridos.

- Públicos:

Formados por múltiples recursos procedentes de múltiples fuentes, pudiendo ser estas empresas, particulares, o cualquier otro tipo de usuario que decida donar voluntariamente sus recursos computacionales a este tipo de *cloud*. Como principal ventaja, se cuenta con la gratuidad del uso de los recursos, no obstante, se debe de tener en cuenta que la seguridad y garantía de disponibilidad ofrecida por este tipo de *cloud* no está garantizada. Esto es debido a que los recursos, al ser donados por terceros, pueden ser desconectados en cualquier momento, además de que no existe ninguna garantía respecto a la seguridad de los datos, o bien debido a conexiones no seguras, o bien debido a que un usuario esté donando recursos de manera voluntaria con intención maliciosa, tales como recabar datos de usuarios que utilicen este servicio.

Debido a la falta de garantía de seguridad y disponibilidad en este tipo de *clouds*, su uso puede verse limitado, debido a que empresas las cuales trabajen en proyectos o gestionen información la cual sea de interés mantenerla de carácter privado podrán sufrir robo de información. Así mismo, en sistemas de carácter crítico donde la disponibilidad deba ser máxima, esta no podrá ser garantizada.

- Privados:

Este tipo representan a los CPD formados por una propia organización que necesita dichas altas características computacionales, para cubrir su propia alta demanda de recursos. Gracias a este carácter privado, este tipo de *cloud* puede llegar a ofrecer el mayor nivel de seguridad, fiabilidad, y rendimiento, ya que será diseñado de acuerdo a sus propias necesidades.

Por otra parte, la organización propietaria de este servicio deberá realizar una asunción total de la responsabilidad sobre este, lo que implica que las tareas de instalación, configuración, mantenimiento, expansión, modernización, respaldo de información y redundancia de alimentación, red y recursos físicos deberán ser llevadas a cabo por esta.

- Cloud híbridos:

Este tipo de solución *cloud* trata de combinar los dos tipos anteriores con el objetivo de proporcionar las virtudes de ambo. Por una parte, gracias a la parte privada, la cual será instalada en la propia organización, podrá ser empleada para aquellas tareas que requieran de una mayor seguridad, fiabilidad y/o rendimiento, tales como el almacenamiento de datos masivo de carácter privado, así como el uso de aplicaciones o servicios críticos los cuales requieran de unas altas características de disponibilidad y rendimiento. Por otra parte, la parte pública podrá ser empleada para aquellas tareas de carácter no crítico, o como un aporte extra de capacidad de computo cuando se requiera,

de manera que se podrá recurrir a estos recursos extra sólo cuando sea necesario, manteniendo la mayor parte del tiempo el uso exclusivo de la parte del *cloud* privada.

Por otra parte, se debe de tener en cuenta que también incluirá las desventajas de ambos tipos de *cloud*, así como otras características. Por un lado, la parte privada tendrá que ser íntegramente mantenida por la corporación la cual ha montado dicho sistema, incluyendo su mantenimiento, reparación frente averías, futuras expansiones, alimentación, red, refrigeración, redundancia, y gestión, mientras que la parte pública presentará los inconvenientes de una mayor carencia de seguridad, fiabilidad, disponibilidad y rendimiento, además de que estos recursos de carácter público no tendrán porqué ser necesariamente gratuitos, ya que podrán ser donados o cedidos por terceros a cambio de una determinada cuantía económica.

4.1.3. *Modelos de potencia*

Dado que en este proyecto, uno de sus principales objetivos consiste en el ahorro energético en CPD, es necesario disponer de un método fiable para realizar la medición del consumo realizado por estos sistemas. Para obtener la energía consumida por un sistema, se debe de tener en cuenta que esto dependerá, principalmente, de la potencia consumida por dicho sistema, y el tiempo requerido para ejecutar un trabajo determinado, el cual es obtenido mediante la longitud de este y las millones de operaciones por segundo (MIPS) que una máquina es capaz de ejecutar. De esta manera, se puede obtener la siguiente expresión mediante la cual es posible realizar el cálculo de la energía consumida (1).

$$E = P * t \tag{1}$$

Donde además, se obtiene que:

$$t = \frac{L}{MIPS}$$

Siendo MIPS los millones de operaciones por segundo que es capaz de ejecutar un determinado equipo y L, la longitud de la tarea.

La ecuación empleada para obtener la energía consumida por la realización de un trabajo o tarea ha sido empleada en el presente TFG como medio para determinar cómo de apropiado es un determinado equipo para ejecutar una determinada tarea. Se debe de tener en cuenta que un equipo el cual cuente con un consumo de potencia alto, no tiene porqué conducir necesariamente a un alto consumo de energía, debido a que para ello, interfieren otros factores tales como la potencia de computo de dicho equipo, y la longitud

de la tarea en cuestión. Partiendo de esto, se puede observar como un equipo con un alto consumo de potencia, el cual cuente con una muy alta capacidad de computo, será capaz de realizar una determinada tarea con un menor consumo energético que otro equipo el cual realice un bajo consumo de potencia pero cuente con unas características computacionales muy limitadas, debido a que puede precisar de una gran cantidad de tiempo para poder llevar a cabo dicha tarea, la cual en el caso del equipo más potente, sería realizada en un muy corto período de tiempo, dando lugar a que, finalmente, el equipo con menor consumo de potencia, el cual a priori puede parecer el equipo idóneo para realizar dicha tarea, termine siendo el equipo el cual realice el mayor consumo de energía.

No obstante, lo que a priori parece una sencilla operación para obtener el consumo energético realizado por un equipo, conlleva una serie de dificultades, debido a que el consumo realizado por un equipo no se reduce solo y exclusivamente al realizado por el procesador del equipo, sino que además, este cuenta con otra serie de componentes los cuales también realizan consumo eléctrico, y deben ser tenidos en cuenta, ya que de lo contrario, se estaría obteniendo un consumo energético erróneo. La ecuación mediante la cual se obtiene el consumo total del sistema queda expresada de la siguiente manera (2):

$$E = E_d + E_s \tag{2}$$

Tal y como se muestra en la ecuación anterior, existen dos componentes, el componente dinámico, y el componente estático. La componente dinámica [6] es utilizada para representar al procesador del sistema. Esto es debido a que los procesadores modernos, cuentan con la capacidad de alterar sus necesidades de alimentación eléctrica en función de la carga de trabajo computacional que requieran. La obtención de la energía dinámica queda expresada mediante la siguiente ecuación (3):

$$E_d = P_d + t \tag{3}$$

Por otra parte, se tiene que la potencia dinámica [6], la cual representa, como ya ha sido expresado con anterioridad, el consumo realizado por la CPU, dependerá de la capacitancia de este, su frecuencia, la tensión al cuadrado, y una constante dinámica. La potencia dinámica queda expresada con la siguiente ecuación (4):

$$P_d = k_d * C * V^2 * f \tag{4}$$

La componente estática de la energía, por otra parte, queda expresada en función de la componente dinámica de la energía, multiplicada por una constante estática, la cual se suele aproximar a un valor del 30 % [8][9]. La componente estática queda expresada con la siguiente ecuación (5):

$$E_s = k_s * E_d \quad (5)$$

Resolviendo la ecuación (2) mediante la ecuación obtenida en (5), se obtiene la siguiente ecuación (6):

$$E = E_d + k_s * E_d \quad (6)$$

De esto, se observa que la energía total consumida se expresa y queda condicionada a la energía consumida por la componente dinámica, al estar la componente estática condicionada al consumo realizado por la componente dinámica. Además, se debe tener en cuenta que la componente dinámica depende directamente de la tensión y la frecuencia a la que estén funcionando la CPU, de manera que una menor frecuencia y tensión conducirá a un menor consumo de energía. No obstante, se debe tener en cuenta que una menor frecuencia ocasionará que la CPU cuente con una capacidad computacional menor, de manera que necesitará más tiempo para realizar una tarea específica que, por otra parte, sería resuelta en menor tiempo si la frecuencia estuviera elevada, debiendo tener en cuenta que si a dicha menor frecuencia la CPU se encuentra en una condición de saturación (100 % de uso de CPU) y a una frecuencia mayor esto no sucede, es probable que a dicha menor frecuencia, el consumo energético sea mayor, al necesitar de demasiado tiempo para resolver dicho trabajo o tarea.

4.1.4. *Dynamic Voltage and Frecuency Scaling (DVFS)*

DVFS es un método mediante el cual los procesadores modernos permiten alterar sus características de funcionamiento (tensión y frecuencia), para adaptarse a la carga de trabajo y poder conseguir un cierto nivel de ahorro energético. Mediante este proceso, los procesadores modernos son capaces de disminuir o aumentar tanto la tensión como la frecuencia cuando el uso de este es bajo, o, por el contrario, aumentar estos bajo una alta carga de trabajo. Gracias a esto, se consigue no solo un mayor ahorro energético en condiciones de baja carga de trabajo al bajar la frecuencia y la tensión del procesador, sino que también se consiguen unas condiciones térmicas más favorables para las CPU, debido a que, en circunstancias en las que estas estén ociosas, podrán disminuir su frecuencia y tensión hasta el nivel oportuno el cual le permita seguir atendiendo correctamente la carga de trabajo la cual tenga que atender dicha CPU.

Para la implementación de DVFS, la CPU debe de soportar dicha técnica, no obstante, la mayoría de los procesadores modernos ya soportan esta tecnología. Además, esta tecnología requiere del uso de gobernadores, los cuales deben estar implementados a nivel de *kernel* [10] (núcleo) del SO. En el simulador empleado para realizar el presente proyecto, se ha trabajado y cuenta con 5 gobernadores, los cuales pueden ser divididos

en dos grupos principales, estáticos, los cuales cuentan con valores de funcionamientos fijos, y dinámicos, los cuales permiten adaptar el funcionamiento a la carga de trabajo de la CPU.

Los gobernadores estáticos trabajan con valores fijos de tensión y frecuencia, de manera que fijarán la pareja tensión-frecuencia de manera que la CPU siempre trabajará con dichos valores sin importar la carga de trabajo de esta. El simulador utilizado en el proyecto, cuenta con tres tipos de gobernadores estáticos.

- Tipos:
    - *Performance*: este tipo de gobernador estático tiene como objetivo ofrecer siempre la mayor capacidad de computo posible de la CPU, fijando ésta a su frecuencia y tensión de trabajo máximas. Como principal ventaja, se consigue que siempre estén disponible en la CPU su mayor capacidad de cómputo para ejecutar tareas, consiguiendo así que las tareas sean siempre ejecutadas en el menor tiempo posible. Por el contrario, poseen una gran desventaja, debido a que siempre estará trabajando a su mayor frecuencia y tensión de funcionamiento, en condiciones de un nivel de carga medio-bajo e incluso en condición de estado ocioso, se estará produciendo un alto nivel de consumo energético el cual no es necesario, ya que en el caso de que la CPU estuviera trabajando a una menor frecuencia y tensión, se podría ahorrar consumo energético.
    - *PowerSave*: tiene como objetivo conseguir el ahorro máximo de energía por medio de fijar tanto la tensión como la frecuencia de funcionamiento de la CPU a sus valores más bajos posibles. Como principal ventaja cuenta con un gran nivel de ahorro energético, al forzar a esta a trabajar a su nivel de rendimiento mínimo. Por el contrario, al trabajar a su nivel más bajo de frecuencia y tensión, se tiene que la potencia de computo real disponible para la ejecución de tareas será mínima, de manera que bajo una alta carga de trabajo, la ejecución de las tareas será demorada al no poder trabajar a mayores frecuencias de funcionamiento, pudiendo propiciar el efecto contrario al deseado de ahorro de energía, provocando un gran consumo de energía al demorar al ejecución de tareas en el tiempo de manera excesiva.
    - *UserSpace*: permite al usuario seleccionar la tensión y frecuencia a la que se desea que la CPU trabaje, teniendo en cuenta que estos valores deberán ser elegidos limitados a los valores los cuales esta pueda trabajar. Al igual que los dos gobernadores anteriores, cuenta con la desventaja de

que al fijarse una determinada frecuencia, se corre el riesgo de que dicha frecuencia no proporcione la capacidad de computo necesaria para ejecutar las tareas evitando que estas sean encoladas por falta de recursos (*conservative*), o bien, que se seleccione una frecuencia la cual sí proporcione dichos recursos computacionales, pero la CPU presente un estado de inactividad la mayor parte del tiempo, conduciendo así a un alto grado de desaprovechamiento de los recursos computacionales, y conduciendo a un sobre consumo innecesario (performance).

Por otro lado, para solventar los problemas expuestos anteriormente en los gobernadores estáticos, surgieron, como ya se ha especificado antes, los gobernadores dinámicos, los cuales, a diferencia de los gobernadores estáticos, permiten variar los valores de tensión y frecuencia de funcionamiento de la CPU dinámicamente en función de la carga de trabajo bajo la cual se encuentra esta. Esto permite que, bajo una alta carga de trabajo, la CPU automáticamente escale tanto su frecuencia como su tensión para alcanzar un valor tal que permita otorgar una potencia de computo tal que satisfaga los requerimientos de computo en ese momento, de igual manera que, en una situación de estado ocioso de procesamiento, permitirá que tanto la frecuencia como la tensión de funcionamiento bajen de manera dinámica, permitiendo y logrando así tanto unas menores temperaturas como un mayor ahorro energético.

No obstante, se debe tener en cuenta que las CPU cuentan con un listado de tensiones y frecuencias a las cuales pueden trabajar, de manera que esta no podrá aumentar su frecuencia de funcionamiento por encima de su frecuencia máxima, ni bajar su ésta por debajo de su frecuencia mínima. Este funcionamiento se basa en una frecuencia base de reloj impuesta por la placa base, y una serie de multiplicadores los cuales harán que el procesador pueda trabajar a mayores frecuencias, de manera que dicho valor del multiplicador, condicionado a la frecuencia base impuesta por la placa, es lo que permitirá dicho cambio de funcionamiento en la frecuencia de la CPU. Además, se debe de tener en cuenta que la capacidad de que una CPU funcione correctamente bajo una determinada frecuencia, queda especialmente sujeto a las necesidades de tensión de esta, lo que indica que, para funcionar a una determinada frecuencia, la CPU necesitará de un valor superior de tensión, en caso contrario, el funcionamiento podría no traducirse necesariamente en una mejora en la potencia de computo, es por ello que dichos valores de tensión y frecuencia están relacionados entre sí, de manera que el cambio de uno de estos valores implique la modificación del otro.

El presente proyecto cuenta con dos tipos de gobernadores dinámicos, los cuales quedan especificados a continuación.

- Tipos de gobernadores dinámicos:
    - *onDemand*: funciona por medio de un único y exclusivo umbral mediante el cual se condiciona el estado de incremento o decremento de la frecuencia de funcionamiento. En el simulador empleado, por defecto, este valor está fijado a 95 % del uso de la CPU, no obstante, dicho valor puede ser modificado. El funcionamiento de este umbral marca, junto a un contador de iteraciones, la condición de si la frecuencia debe ser aumentada o disminuida. Dicho contador posee un valor de 100 (realizará 100 iteraciones), y tras dichas iteraciones, dependiendo del valor en el que se encuentre el uso de la CPU, procederá a incrementar o disminuir su frecuencia, de manera que si el uso de esta se posiciona sobre el umbral, el valor de la frecuencia de funcionamiento será aumentado a su máximo valor. Por el contrario, si tras 100 iteraciones el valor de carga de trabajo se posiciona bajo el umbral, la frecuencia decrecerá sólo un nivel en su multiplicador, volviendo a reiniciar el contador para repetir el mismo proceso, para determinar si la frecuencia debe ser aumentada o disminuida. Se debe tener en cuenta que la frecuencia podrá ser disminuida hasta alcanzar el valor mínimo del multiplicador permitido por la CPU, no permitiendo reducir la frecuencia de funcionamiento de esta por debajo de su valor de frecuencia mínima.
    - *Conservative*: al igual que *onDemand*, permite la modificación de la frecuencia de la CPU bajo la condición del uso esta, no obstante, este gobernador cuenta con dos umbrales en lugar de un único y exclusivo umbral. Estos umbrales, determinarán si la frecuencia debe ser aumentada o disminuida, de manera que si la carga de trabajo se posiciona por encima del umbral de aumento de frecuencia (*setUpThreshold*), esta será incrementada, mientras que si el uso de CPU es inferior al establecido en el umbral de disminución de frecuencia (*setDownThreshold*), esta será disminuida. Los valores por defecto para *setUpThreshold* y *setDownThreshold* son de 80 % y 20 % respectivamente, permitiendo, no obstante, su modificación manual.

4.1.5. *Máquinas Virtuales (VM)*

El proceso de virtualización constituye una parte muy importante en el campo del *Cloud Computing*. La virtualización consiste en un proceso en el cual, varias máquinas o sistemas operativos, son ejecutados al mismo tiempo en una única y exclusiva máquina. Este proceso o técnica de virtualización otorga una serie de beneficios de gran importancia para el campo del *Cloud Computing*.

En primera instancia, la virtualización supone un gran ahorro en costes de infraestructuras, al permitir ejecutar varias máquinas o servidores en un único y exclusivo equipo, de manera que, en lugar de necesitar, a modo de ejemplo, 10 máquinas físicas diferentes para ofrecer una serie de servicios los cuales no podrían ser ejecutados en un único sistema operativo (SO) por sus características propias, podrán ser ejecutados en una única máquina física la cual cuenta con potencia suficiente para ejecutar dichas 10 máquinas en una sola, gracias al proceso de virtualización, debido a que, a pesar de que para poder ofrecerse un determinado servicio se precise de un entorno específico ejecutado en un SO concreto, estos, mediante el proceso de virtualización, se ejecutarán como sistemas independientes dentro de la misma y exclusiva máquina, permitiendo así, como ya ha sido citado, un ahorro considerable de espacio en el CPD.

En segunda instancia, se consigue un mejor aprovechamiento de los recursos de las máquinas físicas. Este mejor aprovechamiento de los recursos se debe principalmente a que, el estado normal de los equipos, en el cual estos se encontrarán la mayor parte del tiempo, es el de un consumo de recursos de procesamiento mínimo, debido a que, por lo general, la carga de procesamiento de estos servidores será medio, bajo, muy bajo, e incluso reposo u ocioso (IDLE), la mayor parte del tiempo. De esta forma, se observa que máquinas en las cuales no se utilice el proceso de virtualización y se encuentre ejecutando un único y exclusivo sistema, podrán estar ociosas o bajo una baja carga de recursos la mayor parte del tiempo, lo que conlleva a un alto nivel de desaprovechamiento de los recursos de dicho sistema, situación que puede ser evitada mediante el empleo de la virtualización. Para conseguir esto, previamente será necesario realizar en primera instancia un proceso de análisis y estudio de los diferentes sistemas en ejecución, para detectar aquellos los cuales posean un bajo nivel de carga de procesamiento la mayor parte del tiempo. Una vez detectado los sistemas los cuales están desaprovechando sus recursos computacionales, estos puedan ser virtualizados y ejecutados en una única máquina física, pudiendo así ejecutar, dependiendo de características como la potencia computacional de la máquina en la cual se ejecutarán dichas máquinas virtuales, y las características de demanda de procesamiento de la

máquina a virtualizar, poder ejecutar en una misma máquina hasta un gran número de sistemas gracias al proceso de virtualización.

En tercera instancia, gracias al proceso de concentración de múltiples sistemas en un único equipo físico, se conseguirán obtener grandes niveles de ahorro energético. Este ahorro energético es conseguido gracias a la disminución en el número de máquinas físicas a utilizar, debido a que, a modo de ejemplo, previo a utilizar el proceso de virtualización, 10 servicios los cuales por sus características requieran ser ejecutados en sistemas operativos independientes que requieren 10 máquinas físicas, a pesar de que estos sistemas puedan estar la mayor parte del tiempo en reposo, seguirán realizando un consumo energético, debido a que aun en estado de reposo, el sistema necesitará realizar cierto grado de consumo energético para mantener el equipo en funcionamiento, debido a que ciertos componentes tales como la placa base, memoria del sistema (RAM), almacenamiento masivo, y la CPU, realizarán cierto grado de consumo energético. Por medio de la virtualización, dichas diez máquinas, las cuales están en estado ocioso la mayor parte del tiempo, podrán ser ejecutadas en una única máquina, logrando así que una única máquina esté en estado ocioso la mayor parte del tiempo, en lugar de 10 máquinas.

Por último, otra gran característica de las máquinas virtuales es la capacidad de realizar la migración [11] de estas. Ciertos entornos de virtualización, tales como VMWare [8], permiten la capacidad de realizar la migración de VM entre sistemas anfitriones, lo cual permite que VM las cuales están siendo ejecutadas en un cierto sistema anfitrión, pueda ser pasado y ejecutado a otro. Esto otorga una serie de ventajas, como la capacidad de migrar una VM la cual realiza un alto consumo de recursos computacionales a otro sistema el cual cuente con más recursos computacionales libres para poder ejecutar dicha esta y viceversa.

4.2. Primera fase: aprendizaje de los simuladores empleados

Tal y como ha sido explicado en el capítulo de introducción, para la realización de este proyecto, en el cual se debe realizar la implementación de un Meta-Planificador para realizar un posterior estudio e implementación de algoritmos de Meta-Planificación, se ha procedido a utilizar un sistema de simulación bajo el lenguaje de programación Java. El simulador escogido para realizar esta implementación es WorkflowSim con DVFS, debido a las bondades que este presenta, las cuales serán detalladas más adelante. No obstante, a pesar de la gran versatilidad e idealidad de este simulador para realizar el trabajo objetivo de este TFG, cuenta con la desventaja de ser un simulador con un alto grado de complejidad, de manera que para poder realizar la implementación de dicho Meta-

Planificador, previamente ha sido necesario realizar un estudio minucioso de este entorno de simulación.

Por otra parte, parte de la gran complejidad de este entorno de simulación se debe a que este entorno ha surgido como la evolución de otros entornos de simulación de *cloud computing,* tal y como se muestra en la Figura 6, de manera que, para poder realizar un correcto estudio y aprendizaje de este, ha sido necesario realizar un estudio previo de los simuladores de los cuales ha surgido.

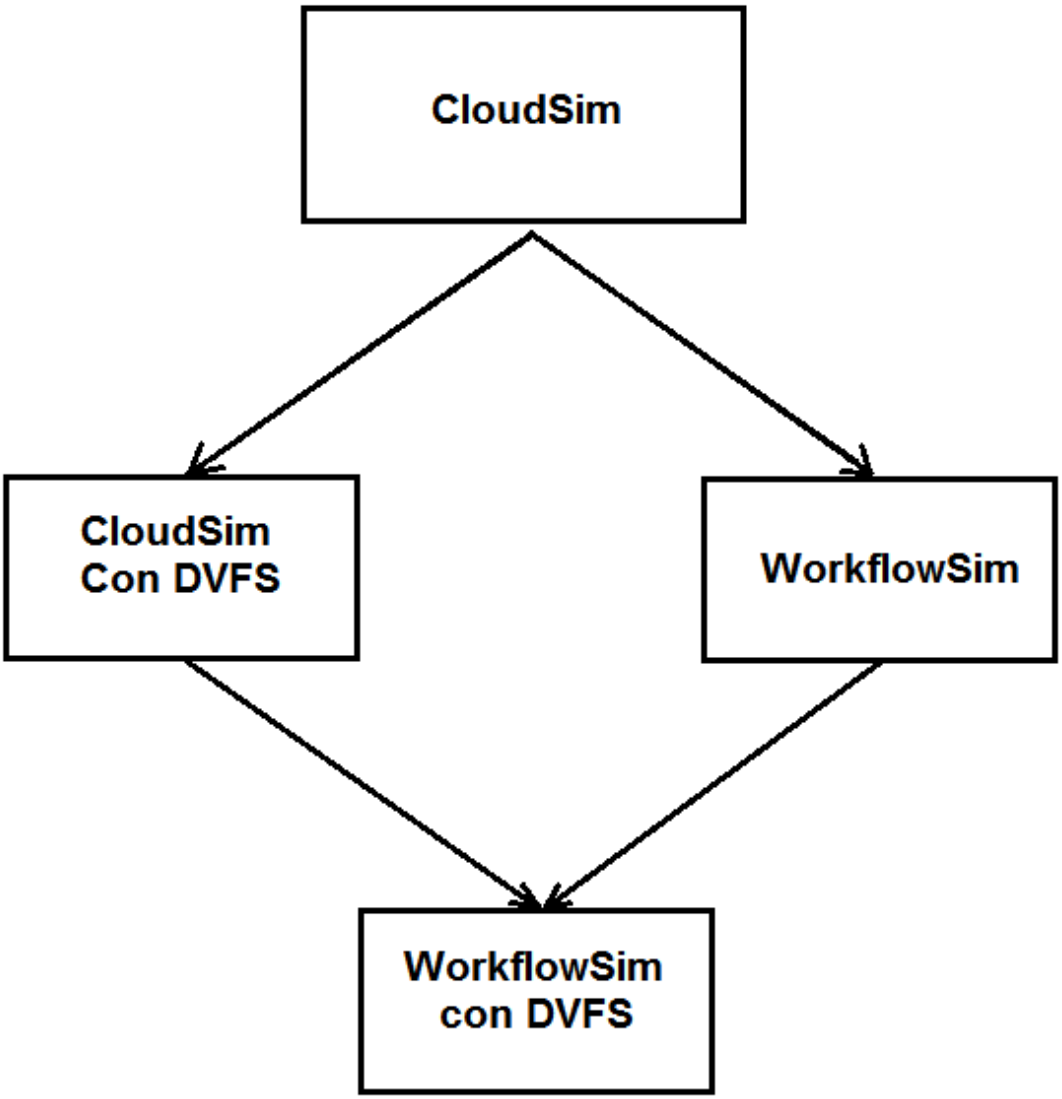

Figura 6: Evolución de CloudSim

4.2.1. *CloudSim*

Tal y como ha sido explicado anteriormente, CloudSim [13, 14], es un simulador de *cloud* escrito en el lenguaje de programación Java. Este simulador se encuentra

estructurado en una serie de elementos llamados entidades, de manera que, cada una de estas tendrá la capacidad de realizar una función específica. Además, cuenta con otra serie de elementos que contribuyen a realizar la simulación del entorno de *cloud*. Gracias a estar constituido por esta serie de elementos, los cuales serán descritos con mayor nivel de detalle a continuación, CloudSim cuenta con un alto nivel de flexibilidad, lo que permite realizar la simulación de gran cantidad de escenarios diferentes, en los que se podrán variar una gran cantidad de parámetros para adaptar el entorno de simulación al escenario de pruebas deseado.

Las entidades constituyen una parte muy importante del simulador, desarrollando cada una de estas una funcionalidad específica. Además, estas se comunican entre sí por medio de eventos que están clasificados y diferenciados por un sistema basado en etiquetas (Tags). Las entidades que constituyen CloudSim son CloudSimShutdown, CloudInformationService, Datacenter y DatacenterBroker, siendo Datacenter y DatacenterBroker aquellas que cuentan con un mayor nivel de complejidad y que constituyen un papel de gran importancia en el simulador.

- CloudSimShutdown:

Su funcionalidad consiste en, una vez el simulador no cuenta con ningún otro evento pendiente que atender y procesar, realizar el proceso de finalización de simulación, deteniendo aquellas entidades que aún se encuentren en funcionamiento.

- CloudInformationService:

Se encarga de atender las peticiones tanto del Datacenter como del DatacenterBroker, de manera que, en esta entidad, se realizará el registro de todos los *Datacenter* para que, posteriormente, los DatacenterBroker puedan realizar una petición de características de todas estas.

- Datacenter:

Esta entidad representa un CPD, de manera que será esta la que se encargará de realizar la funcionalidad de ejecución y procesado de todas y cada una de las tareas que sean enviadas a esta. Esta entidad a su vez estará constituida por una lista de host (equipos) en la cual se almacenarán todos los equipos destinados a realizar las tareas de procesamiento de tareas. Además, a su vez, cada uno de estos host estarán constituidos por una serie de Pe (*Processor element* - procesadores), que proporcionan a este la capacidad de realizar el procesamiento de tareas. Por otra parte, un host, de por sí, no es capaz de realizar la ejecución de tareas en este simulador, de manera que, para que este pueda realizar dicha ejecución de tareas, previamente debe ser creado una VM sobre

dicho host, de manera que sobre esta se realizará una asignación de una determinada cantidad de recursos.

- DatacenterBroker:

La funcionalidad del broker es la de trabajar en nombre del usuario final. Realiza la tarea básica de planificación, de manera que poseerá una bolsa de tareas (Cloudlets) las cuales serán enviadas a un determinado Datacenter para su realización. Además, también realiza otra serie de tareas tales como recabar información de los Datacenter disponibles en el sistema, y asignar a estos la creación de las VM para posteriormente realizar la asignación de tareas a estas, con objeto de que estas realicen su procesado. Por otra parte, el broker, una vez haya obtenido de regreso todas las tareas ejecutadas, y ya no cuente con más tareas que ejecutar en su lista de tareas, realizará la función de ordenar la destrucción de las VMs creadas por este, así como su propia auto-finalización una vez todas estas tareas hayan sido ejecutadas.

Tal y como ha sido citado anteriormente, CloudSim cuenta con otra serie de elementos de gran importancia, que no son entidades, pero que cuentan con un papel fundamental en el simulador. Dichos elementos, de los cuales algunos ya han sido mencionados anteriormente, son los elementos host, VM, Pe, y las Cloudlet, son los que otorgan dicha gran flexibilidad al entorno de simulación. A continuación, se explica detalladamente la funcionalidad de estos elementos.

- Host

Es el encargado de realizar la representación o función de simulación del comportamiento de un equipo, el cual está representado por una clase en Java. Está constituido por una serie de parámetros, siendo estos la cantidad de memoria RAM, espacio de almacenamiento, ancho de banda, y uno o varios objetos de la clase Pe (*Processor element*). Mediante estos parámetros, se podrá indicar una serie de características a estos host, de manera que, posteriormente, podrán ser utilizados a la hora de realizar la creación de una VM en un determinado host, que requerirá de unas determinadas características de procesamiento, entre las que se incluyen una determinada cantidad de RAM, almacenamiento, y un cierto número de Pe.

- VM

El elemento VM (*Virtual Machine*), representa a una máquina virtual la cual se ejecutará en un determinado host, siendo ésta representada por un objeto de la clase Vm Java. Este elemento constituye una parte importante en la simulación de equipos, ya que permite dividir los recursos de un determinado host mediante la reserva de estos con el

objetivo de permitir la ejecución de múltiples VM en un mismo equipo. Como resultado de la ejecución de múltiples VM en un mismo host, se tiene la capacidad de que dicho host pueda realizar la ejecución o procesamiento de más de una tarea de manera simultánea, ya que para que una determinada tarea pueda ser ejecutada deberá asignarse a una determinada VM, la cual debe estar en un estado de reposo, es decir, no debe estar ejecutando ninguna otra tarea. Por otra parte, para poder crear este en un determinado equipo, dicho host debe cumplir los requerimientos de esta, siendo estos una determinada cantidad de RAM, almacenamiento, ancho de banda, y un determinado número de Pe. Además, la creación de una VM en un determinado equipo no impedirá la creación de otras posteriormente, pero se debe tener en cuenta que tras la creación de este elemento en un host, las capacidades de cómputo libre de este se verán disminuidas, impidiendo que los recursos reservados para la creación de esta puedan ser empleados en la creación de otra VM.

- Pe

Este elemento o recurso, el cual está representado mediante un objeto de la clase pe en Java, realizará la función de simulación de un procesador. Un objeto de este tipo cuenta sólo con dos atributos, los cuales son el identificador de dicho procesador, y la cantidad de MIPS que este es capaz de ejecutar. La capacidad de MIPS de un Pe desempeña un papel fundamental en cuanto a la capacidad de cómputo que un host dispondrá para la ejecución de tareas, de manera que, a mayor capacidad de MIPS en un determinado host, este dispondrá de más capacidad de cómputo para la ejecución de tareas, lo que implica que una determinada tarea podrá ser ejecutada en una menor cantidad de tiempo.

- Cloudlet

Este elemento que está representado por una clase en Java de su mismo nombre, realiza la función de simular el comportamiento de una tarea. Este objeto cuenta con una serie de parámetros que sirven para caracterizar a las tareas que deben ser ejecutadas en el sistema. Entre estos parámetros, una Cloudlet cuenta con un identificador (id), el cual realiza un papel importante, ya que será, como su nombre indica, lo que identifique a esta, y lo que permitirá su asignación a un determinado broker, permitiendo a este a su vez, realizar su asignación a una determinada VM de un determinado host de un Datacenter. Otro parámetro es la longitud (length), que como su nombre indica, representa el número de instrucciones que dicha tarea consumirá para completar su ejecución, de manera que el tiempo de ejecución guardará una relación directa con su longitud y los MIPS disponibles por la VM que realizará la ejecución de esta. Por otra

parte, cuenta con otros dos parámetros, siendo estos el tamaño del fichero (fileSize) y el tamaño del fichero de salida (outputSize), que representan el tamaño de la tarea previamente y posteriormente a realizar su ejecución.

Tal y como ha sido mencionado previamente, los eventos otorgan la capacidad de comunicación entre entidades en el entorno de simulación de CloudSim. El núcleo de este, formado por la clase estática en Java llamada cloudsim, la cual se encarga de, mediante un proceso iterativo, atender o comprobar los diferentes eventos que se encuentran en cola, de manera que, para cada una de las entidades, se encargará de comprobar dichos eventos pendientes que deben ser ejecutados en sus respectivas entidades para realizar las tareas oportunas correspondientes a dichos eventos sobre dichas entidades.

Los eventos, principalmente están diferenciados por un identificador y una etiqueta. El identificador del evento, no representa una manera de identificar dicho evento, si no que realizará de función de identificar a qué va destinado, es decir, en dicha variable identificador (id), se debe establecer el identificador de la entidad hacia la que va destinada dicho evento, mientras que la etiqueta, representa el tipo de evento que será enviado a dicha entidad.

Las etiquetas (tags), como ya se ha citado, sirven para identificar el tipo de evento que es enviado a una determinada entidad, los cuales son identificados mediante un identificador numérico. En el entorno de simulación de CloudSim, existen una gran cantidad de diferentes tipos de eventos, no obstante, cada entidad será capaz de procesar un conjunto de eventos concretos. A continuación se adjunta en la Tabla 3 los tipos de eventos más comunes utilizados en la comunicación entre entidades de CloudSim.

TAG	**Origen**	**Destino**	**Nombre de evento**
-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

Tabla 3: Resumen de eventos de CloudSim

A continuación, se procede a explicar el funcionamiento de los eventos anteriormente expuestos.

- END_OF_SIMULATION

Este evento es originado por la entidad Broker una vez que no queden tareas pendientes por ser enviadas para su procesamiento y que todas las tareas enviadas han sido devueltas como tareas procesadas. Provoca que la entidad Broker finalice su ejecución.

- REGISTER_RESOURCE

Evento originado por el Datacenter para registrarse en la entidad CloudInformationService en el inicio de la simulación.

- RESOURCE_CHARACTERISTICS

Evento originado tanto por las entidades Broker como por las entidades Datacenter. Cuando este evento es originado por un Broker, este está solicitando a todas las entidades Datacenter del entorno de simulación que envíen sus características a este. Cuando una entidad Datacenter genera este evento, realiza la función de respuestas del evento generado por un Broker en respuesta a la solicitud de características. Mediante esta respuesta, cada Datacenter enviará sus características a la entidad Broker que lo haya solicitado, de manera que la entidad Broker procederá a comparar las características de cada Datacenter para posteriormente realizar los procesos de planificación y creación de VM en el Datacenter elegido.

- RESOURCE_CHARACTERISTICS_REQUEST

Evento interno de las entidades Broker el cual inicia el proceso para obtener la información de todas las entidades Datacenter del entorno de simulación.

- CLOUDLET_RETURN

Evento generado por una entidad Datacenter en el cual este devolverá a la entidad Broker la cual le asigno una tarea (Cloudlet) a una determinada VM para su procesamiento, una vez que esta haya sido completada.

- CLOUDLET_SUBMIT

Evento generado por un Broker en el cual asignará a una VM de un host de un determinado Datacenter una determinada tarea para su procesamiento, enviando ésta a dicha VM para que proceda a su ejecución.

- VM_CREATE_ACK

Evento que puede ser generado por un Broker con destino hacia un Datacenter o viceversa. Cuando este evento es generado por un Broker, solicita la creación de una determinada VM en el Datacenter destino, mientras que cuando este evento es generado por un Datacenter, realiza la función de respuesta a la solicitud de creación de una VM, respondiendo con asentimiento positivo o negativo en función de si esta ha sido creada con éxito.

- VM_DESTROY

Evento generado por una entidad Broker con destino hacia un determinado Datacenter. Este evento es generado una vez un determinado Broker no posee más tareas que ser enviadas para su procesamiento, y todas las tareas previamente enviadas han sido devueltas tras su procesado. La finalidad de este evento es el de solicitar la destrucción de las VM creadas por este con objeto de liberar recursos en dichos Datacenters en los cuales haya creado VMs para que así estos recursos puedan ser utilizados por otros Broker.

- VM_DATACENTER_EVENT

Evento interno de las entidades Datacenter. Este evento es generado cuando una entidad Datacenter tiene tareas que procesar, de manera que se auto-enviará este evento para ordenar el procesamiento de dichas tareas. En este mismo evento, una vez se comprueba que el procesamiento de una tarea ha finalizado, esta será enviada de regreso al Broker que asignó dicha tarea.

A continuación, se expone un ejemplo básico de simulación, en el cual se detallan todos los pasos que son llevados a cabo en la simulación por cada una de las entidades para llevar a cabo el proceso de ejecución de las tareas. En este ejemplo, sólo se tendrá en cuenta la ejecución de una única Cloudlet en un entorno constituido por un único

Datacenter, con un único host, y un único Broker el cual procederá a crear una VM en dicho único equipo de dicho Datacenter para realizar las tareas de simulación.

1. Inicialización de la simulación. En este primer paso, se procede a inicializar todas las entidades del entorno de simulación. El único Datacenter procederá a registrarse en el CloudInformationService por medio del evento REGISTER_RESOURCE (Tag 2) y la entidad Broker procederá a iniciar el proceso necesario para obtener las características de todos los Datacenters mediante el evento RESOURCE_CHARACTERISTICS_REQUEST
2. La entidad Broker procede a realizar la petición de las características de todos los Datacenters (en este caso sólo de uno) mediante el evento RESOURCE_CHARACTERISTICS
3. El único Datacenter procederá a enviar al Broker sus características como respuesta al evento enviado por este mediante el evento RESOURCE_CHARACTERISTICS. Una vez recibidas las características, el Broker procede a realizar el estudio de las características de estos para elegir al mejor de todos para la creación de las VM. Dado que solo hay uno, elegirá este.
4. La entidad Broker procede a solicitar la creación de las VM en el Datacenter elegido por medio del evento VM_CREATE_ACK. Además, este creará una lista con las VM solicitadas para su creación, ya que esperará respuesta por parte del Datacenter sobre si estas VM han podido ser creadas o no con éxito, y, posteriormente, utilizar dicha lista para proceder a eliminar las VM que este haya solicitado crear una vez haya finalizado la ejecución de todas las tareas y no disponga de más tareas que procesar.
5. El Datacenter procesará las peticiones de creación de VM recibidas y procederá a su creación si posee los recursos necesarios para ello. Una vez finalizado el procesamiento de dichas peticiones, procederá a informar al Broker del éxito o no de la creación de la o las VM (en este caso sólo una) mediante el evento VM_CREATE_ACK.
6. Una vez el Broker ha recibido el asentimiento de creación de las VM, procede a enviar las tareas (en este caso una única Cloudlet) al Datacenter para que se proceda a ejecutar dicha tarea. Las tareas son enviadas mediante el evento CLOUDLET_SUBMIT.
7. La entidad Datacenter procede a realizar el procesamiento de dicha tarea enviándose a sí mismo el evento interno VM_DATACENTER_EVENT.

8. Una vez el Datacenter ha finalizado el procesamiento de la tarea recibida, procederá a enviar el resultado al Broker mediante un evento CLOUDLET_RETURN.
9. El Broker una vez ha recibido dicha tarea, comprobará si existen más tareas por ejecutar. Como en este caso no existen más tareas pendientes por ejecutar, y todas las tareas enviadas han sido ejecutadas, procederá a solicitar al Datacenter la eliminación de las VM creadas mediante el evento VM_DESTROY. Ante este evento, el Datacenter procede a destruir las VM para así liberar recursos. Además, el Broker se enviará a sí mismo un evento END_OF_SIMULATION para proceder a finalizar su ejecución. Por otra parte, el núcleo de CloudSim observará que ya no existen más eventos pendientes por procesar, por lo que procederá a finalizar la ejecución de la simulación.

Se debe tener en cuenta que el ejemplo expuesto anteriormente es un ejemplo muy básico, de manera que no siempre el proceso para finalizar la simulación desde su comienzo será tan directo como en el ejemplo expuesto. En escenarios más complejos en los que el número de tareas sea mayor al número de recursos disponibles, se observará que las tareas tendrán que ser enviadas al o los Datacenter de manera progresiva, a medida que estos vayan procesando las tareas recibidas, de manera que cuando el o los Brokers reciban las tareas previamente enviadas ya completadas, estos podrán enviar nuevas tareas a los Datacenters para que se proceda con la ejecución de las nuevas tareas. Por otra parte, en entornos más complejos en los que un Broker proceda a crear más VM de las que puede crear el Datacenter, esta enviará información mediante eventos VM_CREATE_ACK informando de que no todas las VM han podido ser creadas, o bien informando que ha sido necesario modificar las características computacionales (MIPS) de las VM solicitadas para su creación con objeto de crear todas las VM solicitadas.

Como ya ha sido citado anteriormente, CloudSim es un entorno de simulación el cual cuenta con una gran flexibilidad. Esto otorga una gran ventaja a dicho entorno de simulación, ya que sus restricciones son mínimas a la hora de crear diferentes tipos de escenarios, pudiendo crear un número específico de Cloudlets de la longitud deseada, así como el número de Datacenters y número de host en estos, además de que para cada host se podrá especificar las características computacionales deseadas, pudiendo especificar la capacidad de memoria, número de procesadores por host y MIPS por procesador deseado.

Sin embargo, CloudSim cuenta con una serie de desventajas. Por una parte, solo permite realizar la simulación de procesamiento de tareas teniendo en cuenta tiempos, es decir, este no informará en ningún momento ni permitirá obtener información sobre el consumo energético consumido por los Datacenters para realizar el procesamiento de las tareas, lo cual impide realizar el estudio del consumo energético de un entorno Cloud específico, limitando así el estudio de dichos entornos simulados a los tiempos de ejecución. Por otra parte, tampoco permite realizar la ejecución de flujos de trabajo, es decir, sólo permite realizar la creación de tareas (Cloudlets) simples, las cuales podrán poseer una longitud especificada, pero no se podrá establecer dependencias entre tareas, impidiendo establecer que ciertas tareas realicen la función de tareas padre sobre otras hijas. Esto impide realizar una representación fiel de la realidad, ya que para realizar un trabajo específico, este estará compuesto de una serie de tareas las cuales poseerán dependencias entre sí, impidiendo que ciertas tareas puedan ser ejecutadas hasta que se ejecuten otras previamente y se obtenga el resultado de estas.

Debido a las limitaciones anteriormente citadas, se observó que el entorno de simulación CloudSim, aun siendo muy flexible, contaba con la imposibilidad de obtener información cada vez más de interés en los entornos Cloud, para el caso referente a los consumos, y de la imposibilidad de simular flujos de trabajo, lo que dio lugar a que, a partir de CloudSim, como evolución natural, surgieran dos nuevos simuladores, CloudSim con DVFS y WorkflowSim, los cuales tratan de solventar los problemas anteriormente expuestos.

4.2.2. *CloudSim con DVFS*

Como ya ha sido mencionado con anterioridad, CloudSim con DVFS [15] surgió para solventar uno de los dos grandes problemas previamente comentados de CloudSim, concretamente su incapacidad para calcular el consumo de energía realizado por el o los Datacenters. CloudSim con DVFS surgió como una extensión para añadir la citada funcionalidad de la medición de energía, utilizando como base el simulador de CloudSim, de manera que guarda una gran similitud en la estructura de este simulador, al contar con el mismo núcleo, y contando con las mismas entidades que este poseía. No obstante, para otorgar la capacidad de medición de consumo, en el cual se integran las tecnologías modelos de potencia (3.1.3) y de DVFS (3.1.4), se introdujeron nuevas clases Java. Además, para la inclusión de los modelos de potencia y DVFS, se introdujo una evolución de las entidades Datacenter y Broker, llamadas PowerDatacenter y PowerDatacenterBroker, las cuales poseen una serie de modificaciones las cuales

permiten realizar la medición de potencia en la simulación, así como otros clases java tales como las evoluciones de host y Vm, ahora llamadas PowerHost y PowerVm.

Un cambio importante que se introdujo tras la inclusión de los modelos de potencia es la caracterización de procesadores reales. Esto permite realizar la simulación de consumos y tiempos basado en sistemas o equipos reales. La simulación de dichos procesadores se realiza mediante mediciones reales de equipos con procesadores concretos de manera que para todas y cada una de las frecuencias de funcionamiento de este, sobre el cual se desea crear el modelo de potencia se realizarán medidas de consumo tanto en reposo como bajo carga máxima (100% de uso de CPU). La principal ventaja de esto es que permite realizar estimaciones reales de consumos para entornos reales que estén basados en un modelo concreto de CPU, de manera que si se necesita implementar un determinado *datacenter* real con un determinado procesador para los equipos físicos, realizando previamente el estudio de consumos y creación de dicho modelo, se podrá proceder a realizar la simulación de consumos de dicho procesador bajo este nuevo simulador.

Por otra parte, CloudSim con DVFS cuenta con grandes similitudes en el funcionamiento respecto a CloudSim, existiendo variaciones mínimas. La principal variación en el funcionamiento respecto a CloudSim es la modificación del comportamiento de la entidad Datacenter (PowerDatacenter para CloudSim con DVFS), en la cual, en este simulador, la simulación de las tareas es realizada de una nueva forma. En CloudSim, dicho procesamiento se realizaba mediante la estimación del tiempo de ejecución necesario para completar su ejecución, realizando solo una única vez la ejecución de las tareas en un único evento, en el cual se obtenía dicho tiempo estimado de ejecución. En CloudSim con DVFS, la simulación de la ejecución de las tareas se realizará en múltiples iteraciones en un tiempo llamado SchedulingInterval (intervalo de planificación), que por defecto posee un valor de 0.01. Cada SchedulingInterval, el entorno de simulación procederá a realizar la estimación de la potencia consumida en ese tiempo de ejecución y la porción de la tarea que ha sido ejecutada, de manera que, dependiendo de la frecuencia de funcionamiento de los procesadores de cada una de las máquinas físicas del Datacenter en el cual se está realizando el procesamiento de las tareas, se realizará una estimación del consumo realizado para su ejecución.

Debido a la implementación de DVFS, podrá suceder que durante la ejecución de una determinada tarea o conjunto de tareas, la frecuencia de funcionamiento del equipo se vea alterada debido a la carga de trabajo del Pe (procesador) de la máquina física sobre la que se está ejecutando la o las VM las cuales están ejecutando las tareas, lo que

provocará que los MIPS de dicho Pe se vean reducidos. Esto puede provocar que, al disponer el Pe de menos MIPS, la suma de los MIPS de las VM que están siendo ejecutadas en el host sea mayor a los MIPS Pe, lo que implicará que CloudSim deba realizar un reajuste en los MIPS asignados a cada una de esas VM, de manera que estos MIPS disponibles para cada VM se podrán ver reducidos, implicando así que al disponer las VMs de menos MIPS, el tiempo de ejecución para completar las tareas que están siendo ejecutadas se vea incrementado.

Para el estudio de los consumos realizados por la simulación, CloudSim con DVFS mostrará la información de consumos de dos maneras diferentes, de manera iterativa, y de manera resumida. La información de consumos mostrada de manera iterativa corresponde al consumo realizado tras cada iteración de procesamiento de tareas que realiza la entidad PowerDatacenter, de manera que mostrará información de las frecuencias de funcionamiento (índice de funcionamiento de la tabla de frecuencias), así como la potencia consumida en cada iteración. La información resumida se mostrará al finalizar la ejecución de todas las tareas del entorno de simulación, de manera que, al finalizar esta, mostrará una tabla de información resumida en la cual mostrará la siguiente información de consumos.

Información	**Unidad de medida**
Tiempo total de simulación	Tiempo, expresado en segundos
Sumatorio de potencia	W
Potencia media	W
Consumo energético	Wh

Tabla 4: Información resumida de consumos de DVFS

Tal y como se puede observar, gracias a la implementación de los modelos de potencia y de DVFS en CloudSim, se permite la simulación de una infraestructura real en el cual se podrá realizar un estudio de consumos energéticos de un entorno real. No obstante, aun permitiendo la simulación de entornos reales y la obtención de los consumos del Datacenter, CloudSim con DVFS sigue sin contar con la capacidad de simular flujos de trabajo, lo que implica que, aun pudiendo simular un *datacenter* basado en una infraestructura hardware real, no se podrá simular de manera correcta y fiel a la realidad cómo son ejecutadas las tareas, de manera que las simulaciones que se podrán realizar sólo servirán para realizar estudios muy básicos, al no poder simular situaciones reales de tareas.

4.2.3. *WorkflowSim*

Como ya ha sido citado anteriormente, tanto CloudSim como su variante con DVFS contaban, como principal inconveniente, la imposibilidad de realizar la simulación de ciclos de trabajo o DAG (*Directed Acyclic Graph*) [16], de manera que no se podían simular situaciones reales de ejecución de tareas en las cuales existen dependencias y restricciones en cuanto al orden de ejecución de estas. Debido a esto, se procedió a crear el nuevo simulador WorkflowSim [17] el cual utilizaba como base CloudSim en el cual se implementó la funcionalidad para trabajar con DAG los cuales están descritos en archivos DAX, que están escritos en formato XML.

Como principal, cuenta con la capacidad de simular un conjunto de procesos reales, permitiendo así establecer restricciones entre tareas, es decir, permite establecer qué tareas serán tareas padres de otras tareas, estableciendo un entorno jerárquico de tareas en el cual requerirá que ciertas tareas sean ejecutadas previamente a poder ejecutar otras tareas, estableciendo así límites en cuanto al orden de ejecución de las tareas, lo cual supone una gran mejora respecto a CloudSim, en el cual todas las tareas eran ejecutadas siempre y cuando hubieran VMs libres en el sistema, sin posibilidad para establecer restricciones en el orden de ejecución de estas. A continuación, se adjunta en la Figura 7 un ejemplo de flujo de trabajo.

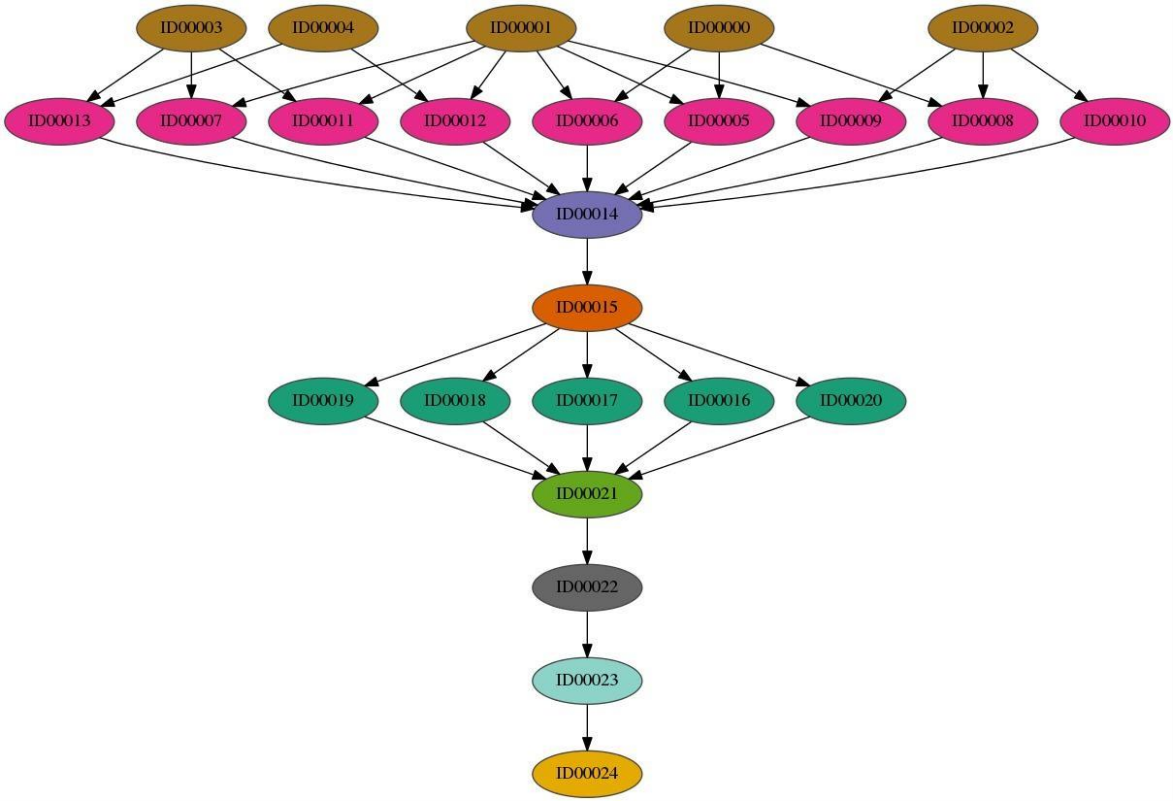

Figura 7: Ejemplo de DAG

Así mismo, para las diferentes tareas especificadas en un flujo de trabajo, se podrá indicar no solo su dependencia con otras tareas para establecer el orden en el que estas deben ser ejecutadas o las restricciones de ejecución entre estas, sino también la capacidad de establecer una longitud diferente a cada una de estas tareas. Esto permite que cada una de estas tareas requiera un tiempo diferente para alcanzar el estado de completado dependiendo de los MIPS del sistema en el cual estén siendo ejecutadas, logrando así que en una máquina con un determinado número de MIPS, cada una de estas tareas las cuales poseerán una longitud diferente requerirán un determinado tiempo diferente para ser ejecutadas. Gracias a esto, no solo se permite establecer un orden jerárquico de ejecución de tareas, sino que al también permitir establecer la longitud de estas de manera individualizada, permite representar con aun más fidelidad una situación real de ejecución de tareas. Además, mediante este sistema DAX, en la definición de las tareas también es posible especificar el tamaño de las tareas de entrada, previo a su ejecución, y el tamaño de estas tras su ejecución, es decir, el tamaño que ocupará el resultado del procesamiento de dichas tareas.

Como principal desventaja de WorkflowSim, este simulador no presenta la capacidad de realizar el cálculo de consumos de energía debido al procesado de las tareas en los host, de manera que sólo se tendrá en cuenta la simulación de tiempos a la hora de realizar esta. Esta incapacidad de realizar la simulación de consumos supone que a pesar de que permite realizar la simulación de tareas con fidelidad a la realidad, esto sólo servirá para realizar el estudio de tiempos y observar y/o planificar, en un entorno con un hardware específico, como deben de ser ejecutadas dichas tareas para obtener el mejor resultado posible en cuanto a tiempos, pero sin la capacidad de realizar un estudio de consumos el cual permita estudiar el entorno con el objetivo de tratar de reducir los costes económicos producidos por los equipos por el consumo de estos.

A diferencia de CloudSim con DVFS, en el cual la estructura original de entidades era respetada, realizando pequeñas modificaciones sobre las entidades Datacenter y Broker originales, WorkflowSim cuenta con una modificación más profunda en cuanto al sistema de entidades, añadiendo nuevas entidades WorkflowPlanner, Parser, ClusteringEngine, WorkflowEngine y WorkflowScheduler, que realizan tareas más complejas que las realizadas inicialmente en CloudSim o CloudSim con DVFS. A continuación, se expone de manera detallada el funcionamiento de cada una de estas nuevas entidades introducidas en WorkflowSim.

- Planner:

En su inicialización realiza la creación de las entidades Parser, ClusteringEngine, WorkflowEngine, y WorkflowSchedulers. El Parser, solo se ejecutará al inicio de la simulación para proceder a realizar la conversión del DAX a una lista de tareas, de manera que cuando esta conversión ha sido llevada a cabo, el conjunto de tareas será enviado la entidad WorkflowEngine.

- ClusteringEngine:

En la creación de esta entidad, será el encargado proceder a crear la entidad WorkflowEngine.

Las entidades WorkflowEngine y WorkflowScheduler están fuertemente relacionadas, constituyendo una parte importante del núcleo de funcionamiento de WorkflowSim, ya que una vez pasado la inicialización de la totalidad de las entidades del simulador, la comunicación y funcionamiento de las entidades principalmente se reducirá a las entidades WorkflowEngine, WorkflowScheduler y WorkflowDatacenter.

- WorkflowEngine:

En su creación, realizará la función de creación de todas las entidades WorkflowScheduler, almacenando un listado de Schedulers y sus identificadores, que utilizará posteriormente para realizar el proceso de envío de tareas a estas entidades. Además, establece a estos Schedulers el identificador de la propia entidad WorkflowEngine, para que cuando estas entidades requieran establecer comunicación con esta mediante un evento, conozcan su identificador y puedan realizar dicha comunicación. Así mismo, se encarga de almacenar y gestionar las tareas que la entidad Planner ha transformado a una lista de Cloudlets procedentes del fichero DAX mediante la entidad Parser, así como de atender a las peticiones de nuevas tareas por parte de los Schedulers. Estas tareas de gestión del listado de tareas y de atención a las entidades Schedulers consisten en, cuando un Scheduler no posee ninguna tarea que planificar, enviará un evento a la entidad Engine de manera que pedirá a esta que nuevas tareas le sean enviadas, de manera que esta estudiará el listado de tareas y observará cuales podrán ser enviadas para su procesamiento. Para que una tarea pueda ser enviada para su procesamiento tiene que cumplirse que, o bien no posea ninguna tarea padre, o bien las tareas padres de esa tarea ya hayan sido ejecutadas. Además, para realizar dicha función en la cual sabrá si una tarea o conjunto de estas, las cuales dependen de otras tareas padres para poder proceder a realizar su ejecución, poseerá un listado de aquellas que han sido enviadas para su procesado, de manera que cuando estas han sido procesadas, se enviarán de regreso a esta entidad, quedando ésta informada, guardando dicha información en dicho listado que le permitirá posteriormente conocer qué tareas

han sido ya procesadas para así poder proceder a enviar aquellas tareas que dependieran de las ya procesadas.

- WorkflowScheduler:

Supone una evolución de la entidad broker de CloudSim. Su funcionalidad consiste en realizar la planificación de tareas en un determinado Datacenter. Cada entidad Scheduler está vinculada a un determinado Datacenter, de manera que esta se encargará de gestionar los recursos de dicho Datacenter, de manera que, esta entidad, en caso de no poseer ninguna tarea, procederá previamente a solicitar mediante un evento a la entidad Engine que esta le envíe una tarea o conjunto de tareas, para, posteriormente, una vez estas han sido recibidas, proceder a realizar el estudio de las diferentes VMs del Datacenter al que está vinculado, observando para cada tarea cual es la VM idónea para realizar el procesado de esta. El proceso de planificación llevado a cabo dependerá principalmente del algoritmo de planificación utilizado por dicha entidad, los cuales serán explicados más adelante. Además, al igual que ocurría con la entidad broker, la entidad Scheduler será la encargada de proceder a crear las VMs en el Datacenter al que esté vinculado, así como a proceder a su destrucción una vez esta haya finalizado.

- WorkflowDatacenter:

Mínimas modificaciones respecto a CloudSim. Se incluye la capacidad de trabajar con un nuevo tipo de VM llamadas CondorVM, en las cuales se puede guardar el estado de la VM, para saber si esta está libre u ocupada, sabiendo así si a dicha VM se le podrá asignar una nueva tarea para realizar su procesado.

Tal y como ha sido citado anteriormente en la entidad WorkflowScheduler, los algoritmos de planificación son utilizados por esta entidad para realizar el proceso de planificación. Estos algoritmos, dependiendo del utilizado, modificarán la estrategia de planificación que utilizará el Scheduler para la toma de decisiones referente a la distribución y asignación de tareas a las diferentes VM de la entidad Datacenter. Se debe tener en cuenta que, en WorkflowSim, a pesar de realizar una aproximación con una alta fidelidad a los entornos reales, no realizará ningún tipo de planificación o control sobre el consumo, como ya ha sido anteriormente mencionado, de manera que la planificación se ceñirá exclusivamente a lo referente a tiempos de ejecución de las tareas, para optimizar este de acuerdo a la estrategia de planificación utilizada. A continuación, se exponen tres algoritmos de planificación, para los cuales se aporta una descripción de funcionamiento detallada, siendo estos los algoritmos MinMin, MaxMin, y RoundRobin.

- MinMin:

El funcionamiento de este algoritmo de planificación consiste en, para una bolsa de tareas (será explicado más adelante) que poseerá la entidad Scheduler, procederá a buscar, de todo el conjunto de tareas, la tarea más pequeña de todas estas, y, una vez haya encontrado dicha tarea, procederá a buscar entre todas las VM disponibles en el Datacenter, aquella que cuenta con el mayor número de MIPS, de manera que esta tarea sea ejecutada en el menor tiempo posible, teniendo en cuenta, además, que dicha VM tendrá que estar en un estado ocioso o desocupado (IDLE). Este algoritmo asegurará que las tareas más pequeñas siempre serán priorizadas en el orden de ejecución, relegando las tareas más grandes y pesadas al final de la cola de ejecución. Como principal desventaja, este algoritmo propiciará que las tareas más grandes y por lo tanto las que más recursos necesitarán para ser ejecutadas, sean ejecutadas por aquellas VM que posean la menor cantidad de recursos, de manera que, previsiblemente, podrá propiciar que la completa ejecución de las tareas se vea ralentizada o retrasada, al requerir más tiempo del necesario para completar su ejecución.

- MaxMin:

Al igual que MinMin, este algoritmo de planificación busca como objetivo optimizar el tiempo de ejecución de las tareas pendientes por ejecutar que posee la entidad Scheduler. Al contrario que MinMin, este algoritmo procederá a buscar aquellas tareas de mayor longitud para enviarlas a aquellas VM las cuales proporcionen el menor tiempo de ejecución y que estén en estado ocioso, es decir, persigue priorizar el procesamiento de aquellas tareas que necesiten de mayores recursos para ser procesados en aquellas VM que puedan otorgar la mayor cantidad posible de recursos para el procesamiento de estas tareas, con el objetivo de tratar de evitar que la ejecución del total de las tareas se vea ralentizada o retardada como consecuencia de asignar las tareas más pesadas a las VM con menores recursos.

- RoundRobin:

El algoritmo de planificación de RoundRobin consiste en un algoritmo muy básico el cual simplemente buscará el reparto equitativo de las tareas en referencia al número de tareas que cada VM procesará sin tener en cuenta consumos o tiempos. Para cada conjunto de tareas entrantes, irá repartiendo estas de manera circular, asignando una tarea a cada VM que se encuentre en un estado ocioso hasta que ya no se dispongan de más tareas que asignar, o bien hasta que a no se dispongan de más VM ociosas a las que asignar tareas, sin tener en cuenta, en ningún momento, parámetros como la longitud de las tareas, los MIPS de las VMs, ni la longitud total de las tareas que han sido ejecutadas por una VM. Debido a que, como se ha especificado, no se tienen en cuenta

tiempos ni los recursos de las VM, este algoritmo propiciará un entorno en el cual se realice una mala planificación de las tareas, ocasionando que tareas de un gran tamaño puedan ser asignadas a aquellas VM que cuenten con unos recursos muy limitados para realizar el procesado de tareas, pudiendo demorar considerablemente el procesamiento de aquellas tareas para las cuales se debería priorizar su asignación a aquellas VM con mayor cantidad de recursos.

Por otra parte, la planificación de tareas no solo se reduce al comportamiento de cómo estas son tratadas y estudiadas para ser enviadas a un determinado recurso con el objetivo de su procesamiento, sino que además, se tiene en cuenta el comportamiento del planificador en como tratará las tareas en función de su llegada a este. Dependiendo de cómo trate el planificador a dichas tareas recibidas, se pueden diferenciar dos métodos de planificación, siendo estos On-Line y On-Batch.

- On-Line:

Método de planificación básico en el cual, el planificador procederá a planificar las tareas en función del algoritmo de planificación programado en este, teniendo en cuenta las tareas como un único elemento, es decir, las tareas serán tratadas de manera individualizada, de manera que, tan pronto una tarea sea recibida, esta será planificada para ser enviada a una VM para ser procesada. Como principal ventaja, este tipo de planificación cuenta con unos tiempos de espera en tareas para ser procesadas prácticamente nulos. Por otra parte, este tipo de planificación cuenta con una gran desventaja. Debido a que las tareas son tratadas de manera individualizada, no se posee ninguna garantía de que se realice una correcta planificación de las tareas, ya que, por ejemplo, un planificador el cual esté utilizando el algoritmo MaxMin, cuando reciba un conjunto de tareas, procesará estas según el orden de llegada o tan pronto estas hayan sido recibidas, de manera que si una tarea ha sido recibida, esta será enviada a la VM más potente, sin tener en cuenta si la siguiente tarea requiere un costo computacional considerablemente mayor a esta ya asignada a la VM más potente para su ejecución, provocando así que tareas las cuales requieran una gran cantidad de recursos para completar su ejecución sean asignadas incorrectamente.

- On-batch:

Este método de planificación [18], a diferencia de On-Line, trabajará con una bolsa de tareas. Esto implica que, en lugar de planificar las tareas tan pronto estas hayan sido recibidas, procederá a, o bien esperar hasta alcanzar un determinado número de tareas en la bolsa de tareas para planificar, o bien esperar un determinado tiempo desde que recibe una tarea, para seguir recibiendo tareas previamente a proceder a realizar el

proceso de planificación mediante el algoritmo de planificación seleccionado. Como principal ventaja, propiciará una mejor planificación de las tareas, ya que estas no serán tratadas de manera individualizada, sino que serán tratadas como un grupo de tareas, permitiendo así a algoritmos como MinMin y MaxMin, trabajar de manera que puedan buscar las tareas más pequeñas o más grandes para ser procesadas por las tareas con mayor cantidad de recursos computacionales, garantizando así una correcta planificación de las tareas. Por otra parte, como desventaja, introducirá un determinado tiempo de retardo en la planificación de tareas, ya que estas no serán planificadas tan pronto como sean recibidas, si no que tendrán que esperar previamente a ser planificadas.

Al igual que sucedía con CloudSim y CloudSim con DVFS, WorkflowSim implementa una comunicación entre entidades basada en el envío de eventos. Dado que WorkflowSim está basado en CloudSim, incluye los mismos eventos como base, no obstante, debido a la inclusión de nuevas entidades y de nuevas funcionalidades las cuales no estaban presentes anteriormente en CloudSim, este cuenta con nuevos eventos, de manera que se suman a los ya presentes en CloudSim para otorgar dicha comunicación y funciones extras que permite debido a las nuevas funcionalidades. Las nuevas etiquetas utilizadas para los nuevos tipos de eventos introducidos en WorkflowSim pueden ser encontrados en la clase WorkflowSimTags dentro del paquete org.workflowsim, habiéndose realizado la inclusión de 7 nuevas etiquetas para representar los 7 nuevos tipos de eventos. Los eventos de CloudSim y de WorkflowSim pueden ser fácilmente diferenciados gracias a la nomenclatura numérica que estos utilizan. Las etiquetas utilizadas en los eventos de CloudSim no poseen ningún tipo de prefijo, siendo estas identificadas por seguir la nomenclatura xx, mientras que las etiquetas utilizadas por los nuevos eventos de WorkflowSim utilizan la nomenclatura 10xx, representando las “x” tanto para CloudSim como para WorkflowSim dígitos numéricos. A continuación se adjunta en la Tabla 5 los eventos más utilizados en WorkflowSim durante el proceso de simulación

TAGS	**Tipo de evento**
-1	END_OF_SIMULATION
2	REGISTER_RESOURCE
6	RESOURCE_CHARACTERISTICS
15	RESOURCE_CHARACTERISTICS_REQUEST
20	CLOUDLET_RETURN

21	CLOUDLET_SUBMIT
32	VM_CREATION_ACK
33	VM_DESTROY
41	VM_DATACENTER_EVENT
1000	START_SIMULATION
1001	JOB_SUBMIT
1005	CLOUDLET_UPDATE

Tabla 5: Eventos de WorkflowSim

Tal y como se observa en la tabla, solo tres nuevos eventos han sido introducidos respecto a CloudSim en una simulación normal, manteniendo el resto de eventos que ya estaban presentes en este simulador. La mayoría de estos eventos los cuales ya estaban presentes en CloudSim realizan una funcionalidad idéntica o muy parecida, no obstante, algunas de estas presentan pequeñas modificaciones en su funcionamiento, es por ello que se procede a explicar de nuevo estos eventos, principalmente por la inclusión de nuevas entidades, las cuales utilizan estos eventos para establecer comunicación en diferente forma a como lo realizaban las entidades de CloudSim.

- END_OF_SIMULATION:

Evento el cual se ejecuta una vez todas las tareas o Cloudlets han sido finalizadas, de procediendo a finalizar todas las entidades. Este evento es generado por la entidad Engine y es enviado a todos los Schedulers, para informar a estos de la no existencia de nuevas Cloudlets que procesar.

- REGISTER_RESOURCE:

Evento utilizado por las entidades Scheduler y Datacenter para registrarse en la entidad CloudInformationService en el inicio de la simulación

- RESOURCE_CHARACTERISTICS:

Evento utilizado por las entidades Schedulers para solicitar las características de las entidades Datacenter, para así, una vez la entidad Scheduler conoce las características de un Datacenter, poder proceder a realizar la planificación de tareas.

- RESOURCE_CHARACTERISTICS_REQUEST:

Evento utilizado por las entidades Engine y Scheduler para comunicarse con la entidad CloudInformationService, para así, de esta manera, poder la entidad Engine conocer las entidades Schedulers que están registradas en el sistema, para así poder comunicarse con estas, así como por la entidad Scheduler, para poder conocer qué entidades Datacenter hay en el entorno. Una vez la entidad Scheduler ha obtenido la información referente a los Datacenter existentes, podrá proceder a consultar sus características por medio del evento RESOURCE_CHARACTERISTICS.

- CLOUDLET_RETURN:

Evento utilizado por las entidades Datacenter para devolver una Cloudlet a la entidad Scheduler para al cual ha finalizado su procesado. Además, tal y como ha sido mencionado con anterioridad, la entidad Engine es la que se encarga de enviar a las entidades Schedulers las tareas que pueden ser procesadas, ya que esta se encarga de gestionar el conjunto de tareas y controlar qué tareas pueden ser o no ser ejecutadas, dependiendo de sus restricciones de ejecución en función de si estas poseen tareas padre las cuales deban ser ejecutadas con anterioridad. Esto ocasiona que este evento también sea utilizado por las entidades Schedulers para devolver una tarea completada a la entidad Engine.

- CLOUDLET_SUBMIT:

Utilizado para el envío de Cloudlets entre entidades. Mediante este evento, las entidades Engine procederán a enviar tareas a las entidades Schedulers que lo soliciten, siempre y cuando se posean tareas las cuales puedan ser procesadas. Además, una vez una entidad Scheduler ha recibido las tareas para ser procesadas, este procederá a utilizar el evento CLOUDLET_UPDATE para realizar el proceso de planificación en función de lo establecido en el algoritmo empleado. Una vez que la planificación ha sido llevada a cabo, la entidad Scheduler procederá a enviar la tarea a la entidad Datacenter mediante el evento CLOUDLET_SUBMIT, para que esta proceda a ejecutar la Cloudlet.

- VM_CREATION_ACK:

Utilizado por las entidades Schedulers para solicitar la creación de VM en los host de sus respectivos Datacenters vinculados, para que así, posteriormente, las Cloudlets puedan ser ejecutadas y procesadas en dichas VM. Además, este mismo evento es utilizado por la entidad Datacenter para comunicar el asentimiento de la creación de estas con o sin éxito.

- VM_DESTROY:

Utilizado por las entidades Schedulers para solicitar a los Datacenter que procedan a destruir las VM creadas. Este evento es generado una vez se ha ejecutado el evento END_OF_SIMULATION, de manera que cuando ya no queden Cloudlets que procesar, ya no será necesario que las VM estén creadas, pudiendo proceder a su destrucción para finalizar la simulación.

- VM_DATACENTER_EVENT:

Evento interno de las entidades Datacenter en el cual proceden a realizar la ejecución de tareas. Debido a que CloudSim no tiene en cuenta la potencia a la hora de realizar el procesado de las tareas, y solo considera tiempos para estas, el proceso de ejecución de una tarea se realizará en un único ciclo, en el cual se calcula el tiempo estimado necesario para completar el procesado de esta. Además, WorkflowSim tiene en cuenta el tamaño de las tareas para considerar el tiempo de ejecución, de manera que cuando una tarea debe ser procesada, calculará, en función del tamaño de esta, el tiempo que esta necesita para ser transmitida a la VM donde se realizará el procesado de esta.

- START_SIMULATION:

Nuevo evento introducido en WorkflowSim. Se ejecuta en el inicio de la simulación, y posee la finalidad de obtener el fichero DAX que contiene la definición de las tareas para que estas puedan ser interpretadas por el simulador, integrando estas en una lista de tareas que poseerán el sistema de restricciones entre sí.

- JOB_SUBMIT:

Nuevo evento introducido en WorkflowSim. La finalidad de este es la de, una vez el fichero DAX ha sido transformado en una lista de tareas, proceder a enviar estas a la entidad Engine, ya que esta entidad se encargará de la gestión de estas, para posteriormente, cuando una entidad Scheduler solicite tareas a la entidad Engine, proceder a, en función de este listado, observar aquellas que puedan ser procesadas y por tanto enviadas a la entidad Scheduler que las ha solicitado.

- CLOUDLET_UPDATE:

Nuevo evento introducido en WorkflowSim. Evento interno de la entidad Scheduler, que posee la finalidad de, una vez una entidad Scheduler ha recibido las tareas solicitadas que deben ser enviadas para ser procesadas a las VM de la entidad Datacenter, realizar el proceso de planificación mediante el que se estudiarán todas las Cloudlets recibidas y todas las VM de la entidad Datacenter, para decidir, en función del algoritmo de planificación establecido a utilizar, para cada Cloudlet, la VM idónea para

realizar el procesado de dicha tarea. Posteriormente a realizar el proceso de planificación, la entidad Scheduler utilizará el evento CLOUDLET_SUBMIT para enviar las Cloudlets que deben ser procesadas a la entidad Datacenter para su ejecución.

- Task y Job:

Para facilitar la comprensión lectora, se procede a indicar las diferencias entre Task and Jobs.

Se entiende por Task una tarea final la cual no puede ser dividida en otras subtareas. Por otra parte, un Job o trabajo consiste en una agrupación de tareas, creando así una bolsa de tareas. De esta forma, se observa que la entidad Engine procede a enviar a las entidades Schedulers Jobs, conteniendo estos una serie de Task, lo que permite al Scheduler dividir dicho trabajo en subtareas o Task, las cuales son asignadas a las VM de los Datacenters para su procesamiento.

- Scheduling Algorithm y Planning Algorithm:

A continuación, se procede a diferenciar *Scheduling Algorithm* de *Planning Algorithm*.

Scheduling Algorithm hace referencia al algoritmo de planificación utilizado por el Scheduler, mediante el que se procede a estudiar las tareas y las VMs disponibles en el Datacenter para proceder a la asignación de estas a dichas VMs. Algunos ejemplos de algoritmos de planificación son RoundRobin, MaxMin y MinMin.

Se entiende por *Planning Algorithm* por la estrategia seguida por el Scheduler a la hora de ejecutar los algoritmos de planificación. Algunos ejemplos de *Planning Algorithm* son los ya citados On-Line y On-Batch.

4.2.4. *WorkflowSim con DVFS*

Surgido de la unión de WorkflowSim y CloudSim con DVFS. WorkflowSim con DVFS [19] añade a WorkflowSim la capacidad de realizar la simulación y cálculo de los consumos realizados por los equipos manteniendo su capacidad para simular flujos de trabajo. Gracias a la implementación de las clases incluidas en CloudSim con DVFS, permite la regulación de las frecuencias de funcionamiento de los CPU mediante el uso de los gobernadores, de manera que, según la carga de trabajo de la CPU y del gobernador dinámico utilizado, podrá variar su frecuencia para proporcionar una mayor potencia de procesamiento o bien un mayor ahorro energético. Como principal ventaja de la unión de ambos simuladores, se otorga la capacidad de realizar estudios y planificación tanto de tiempos como de consumos, permitiendo así estudiar entornos simulados basados en entornos reales, permitiendo observar el consumo energético

realizado para un caso particular de componentes hardware y flujos de trabajo a procesar, permitiendo, de esta forma, proceder a diseñar algoritmos de planificación los cuales se adapten al entorno y circunstancias específicas logrando otorgar un resultado óptimo en el parámetro a optimizar, ya sea este tiempos o consumos.

La unión de ambos simuladores se realizó utilizando WorkflowSim como base, añadiendo y adaptando a este las clases necesarias de CloudSim con DVFS para añadir los modelos de potencia y DVFS, de manera que la estructura de entidades del simulador permanecerá intacta tal cual está presente en WorkflowSim, realizando ciertas modificaciones sobre ciertos componentes del simulador para añadir los componentes de potencia de CloudSim con DVFS, además de otras características las cuales serán expuestas más adelante. A continuación, se exponen de manera detallada la lista de cambios y nuevos añadidos en el simulador WorkflowSim con DVFS.

La principal modificación en el comportamiento de WorkflowSim con DVFS respecto a WorkflowSim es referente al comportamiento de la entidad Datacenter, ahora llamada WorkflowDVFSDatacenter. Esta modificación provoca un cambio del comportamiento del procesado de tareas en los Datacenter (etiqueta 41). El comportamiento pasa a ser similar al presente en CloudSim con DVFS, de manera que, la simulación de ejecución de tareas, en lugar de ser realizada como en WorkflowSim, donde se realizaba en una única iteración, calculando el tiempo estimado para completar su ejecución, poseerá un proceso tal que la simulación corresponderá a la presente en CloudSim con DVFS. Esto implica que, para la ejecución de una tarea, se requerirán múltiples iteraciones, debido a que cada iteración tendrá en consideración la ejecución de sólo 0.01 segundos por defecto, intervalo de ejecución especificado por el parámetro SchedulingInterval. Este tipo de implementación es necesario para poder proceder a realizar un correcto cálculo del consumo energético realizado por la máquina que esté procediendo a realizar la ejecución de una tarea, ya que, dependiendo del gobernador elegido para la simulación de la CPU, la frecuencia de esta variará en función de la carga de trabajo, de manera que esta frecuencia podrá aumentar o disminuir, con el objetivo de perseguir un ahorro energético. Además, se recuerda que dependiendo de la frecuencia de funcionamiento, una CPU tendrá diferentes niveles de consumo, y no solo eso, sino que además, dependiendo de la carga de trabajo a dichas frecuencias, también variará los consumos realizados, de manera que el consumo energético variará en función de la frecuencia de funcionamiento y de la carga de trabajo de esta. Es por ello que para poder realizar un correcto cálculo de la potencia consumida, es necesario realizar el proceso de simulación de procesado de tareas de manera iterativa con un periodo de SchedulingInterval segundos, de manera que de esta forma, podrá ser estimada con mejor precisión y

exactitud el consumo del sistema, además de permitir la variación de la frecuencia de las CPU a medida que avance la simulación del procesado de tareas.

Por otra parte, dado que el simulador utiliza como base WorkflowSim en el cual se han implementado las técnicas de DVFS, el resto de entidades, eventos y su funcionamiento será exactamente igual, existiendo una diferencia en el funcionamiento sólo en cómo son las tareas tratadas a la hora de estas ser ejecutadas, de manera que no será necesario añadir más explicaciones referentes al comportamiento de los eventos en WorkflowSim con DVFS.

Como ya fue mencionado en CloudSim con DVFS, los autores de este simulador introdujeron en este los modelos de potencia. Con estos modelos de potencia, es posible realizar la simulación de procesadores reales en el entorno de simulación, para ello, es necesario realizar un proceso en el cual, un sistema, con un sistema operativo concreto, con hardware concreto, y con un procesador concreto, es pasado por un proceso mediante el cual, dicho procesador de dicho sistema es manualmente probado en cada una de las frecuencias y tensiones a las que este es capaz de trabajar, es decir, el DVFS del sistema operativo es desactivado para así, de esta forma poder realizar un estudio de consumos tanto en reposo como a máxima carga de trabajo para cada uno de las diferentes frecuencias de trabajo que este procesador puede operar. Gracias a los modelos de potencia, es posible realizar la simulación de un Cloud basado en hardware real, el cual resulta de gran utilidad cuando se quiere realizar un estudio preciso de consumos para un hardware específico, permitiendo así programar los algoritmos de planificación de manera tal que el funcionamiento, ya sea en consumos o tiempos sea óptimo para este sistema. Este sistema, aun resultando de gran utilidad, puede resultar tedioso en situaciones en las que no se requiera realizar un estudio sobre un sistema real previo a realizar su implementación, de manera que, en situaciones en las que estos simuladores son empleados con motivo de estudio de los algoritmos de planificación, puede dificultar sobremanera la preparación del entorno de simulación, al requerir un empleo de tiempo excesivo en realizar modelos de potencia de procesadores reales.

En el proceso de desarrollo de WorkflowSim con DVFS, el autor detectó este problema, en el cual era de gran dificultad realizar la preparación de un entorno de simulación no real para realizar el estudio de los algoritmos de planificación, ya fuera para verificar la calidad de los ya existentes o para realizar el desarrollo de nuevos algoritmos. Debido a este problema detectado, el autor procedió a crear un modelo de potencia llamado PowerModelAnalytical.

Este nuevo modelo de potencia, permite crear modelos ficticios con una relativa facilidad, de manera que utiliza una serie de parámetros para especificar las características de estos. Este nuevo modelo procede a calcular dos parámetros necesarios para crear el modelo ficticio, siendo estos parámetros los MIPS, y la potencia consumida. Para calcular estos dos parámetros, el autor procedió a utilizar dos ecuaciones, mediante las cuales procede a calcular la potencia dinámica del procesador, así como los MIPS de este.

Para proceder al cálculo de la potencia dinámica (7) del procesador, procede a utilizar la siguiente ecuación, que está expresada en función de una constante dinámica (K_d), la capacitancia de este (C), su tensión (V) y su frecuencia:

$$P_d = K_d C V^2 f \tag{7}$$

Una vez calculada la potencia dinámica, procede a calcular la potencia total (8) que consumirá dicho procesador mediante la siguiente ecuación, que está expresada en función de la potencia dinámica (P_d) y de una constante estática (K_s).

$$P = \frac{P_d}{1-K_s} \tag{8}$$

Para el cálculo del parámetro MIPS (9), el cual representa la potencia de cálculo de dicho procesador, procede a utilizar la siguiente ecuación:

$$MIPS = \frac{f*IPC}{10^6} \tag{9}$$

El parámetro IPC expresa las instrucciones por ciclo que un procesador es capaz de realizar. El principal problema de este parámetro, por el cual este modelo analítico de potencia no puede ser tomado en consideración para simular sistemas reales, y solo puede ser empleado para realizar la simulación de entornos ficticios con el objetivo de realizar el estudio de algoritmos de planificación, es debido a que el IPC de un procesador no puede ser calculado. Esto se debe a que este parámetro depende de una gran cantidad de características de un procesador, entre las que se encuentran el diseño de este, la arquitectura, tecnología de integración, registros, juegos de instrucciones, así como otros parámetros.

A pesar de que este sistema analítico no puede ser utilizado para realizar la simulación de entornos reales, otorga una gran versatilidad a la hora de realizar estudios sobre los algoritmos de planificación, ya que permite crear entornos ficticios adaptados a la situación sobre la que se quiere estudiar el comportamiento de un algoritmo de planificación, o sobre el cual se quiere diseñar un algoritmo de planificación que otorgue

un rendimiento óptimo, estando basado este entorno ficticio en un entorno real. De esta manera, se podrán simular entornos en los que exista un gran nivel de heterogeneidad entre los diferentes equipos que lo componen, o bien, realizar la simulación de un entorno completamente homogéneo. Además, al permitir establecer parámetros como los MIPS o la potencia consumida, permite realizar la simulación de procesadores ficticios los cuales se ajusten a las características de procesadores reales que pueden ser encontrados en entornos reales, de manera que se podrán simular procesadores de bajo rendimiento y bajo consumo energético, así como procesadores de muy alto rendimiento u otro tipo de características las cuales sean de interés estudiar.

Por otra parte, en el simulador WorkflowSim, se observa la inclusión de algunos algoritmos de planificación de tareas tales como MaxMin y MinMin, los cuales ya han sido explicados con anterioridad en el apartado 3.2.3, donde se explica que estos algoritmos están enfocados a tratar de conseguir que un conjunto de tareas sean procesadas en el menor tiempo posible, para así, de esta forma, obtener los resultados lo antes posible. No obstante, en WorkflowSim con DVFS, se introduce la capacidad de realizar la simulación de consumos, que otorga la capacidad de realizar un nuevo tipo de planificación de tareas basadas en la búsqueda de la minimización de consumos, con el objetivo de reducir costes. Gracias a esta implementación, y con el objetivo de reducir dichos costes, en WorkflowSim con DVFS se integraron nuevos algoritmos de planificación basados en MaxMin y MinMin, quedando estos expuestos a continuación.

- PowerAwareMaxMinSchedulingAlgorithm:

Este algoritmo de planificación, posee un comportamiento similar al algoritmo MaxMin, del que está basado. En primera instancia, este algoritmo procederá a buscar de entre el listado de Cloudlet que han sido enviadas a la entidad Scheduler que utiliza este algoritmo de planificación, aquella que requiera de mayor procesamiento para ser completada. Una vez que esta tarea ha sido encontrada, procederá a asignar ésta a la VM que proporcione el consumo energético menor por su procesado. Este proceso de búsqueda de la tarea más grande será iterativo, de manera que tras cada asignación de una Cloudlet a una VM, procederá a buscar la siguiente del listado de Cloudlets la cual no haya sido asignada, que sea la que más recursos computacionales requiera, procediendo a asignar ésta a una VM en estado ocioso(IDLE).

- PowerAwareMinMinSchedulingAlgorithm:

Posee un funcionamiento similar a MinMin. En primera instancia procede a buscar, de la lista de Cloudlet, aquella que requiera del menor procesamiento para ser completada. Una vez que esta ha sido encontrada, procederá a asignarla a aquella VM la cual

proporcione el menor consumo energético para su ejecución, condicionado a que dicha VM se encuentre en un estado ocioso previo a poder ser asignada a esta. Este proceso se repite de manera iterativa mientras haya tareas que asignar en la lista de tareas y existan VM en estado ocioso, que puedan realizar el procesado de Cloudlets.

A pesar de la inclusión de estos nuevos algoritmos de planificación los cuales tienen como función objetivo la optimización de consumos, los algoritmos de planificación aun cuentan con margen de mejora en la toma de decisiones de qué VM debe realizar la ejecución de una tarea. Este margen de mejora es debido a que, aun otorgando un buen resultado, solo se tendrá en consideración un único parámetro a la hora de realizar la planificación, siendo dicho parámetro el consumo energético realizado por la VM para realizar el procesamiento de la tarea. Aun otorgando un buen resultado este proceso de consideración de un único parámetro para la toma de decisiones, se debe tener en cuenta que otros parámetros de interés no son considerados en esta toma de decisiones, entre los que se encuentran parámetros tales como el tiempo necesario para ejecutar la tarea, los MIPS de la VM, los consumos de potencia y energía, y tamaño de la tarea a procesar. Mediante la consideración de más de un parámetro en el proceso de planificación de tareas en la entidad Scheduler, se podría conseguir una mejora en el proceso de planificación, observándose así la localización del margen de mejora previamente comentado.

No obstante, tomar en consideración más de un parámetro a la hora de realizar el proceso de planificación puede resultar un proceso tedioso y de difícil implementación, de manera que, a priori, puede resultar una opción no viable como punto de mejora en los algoritmos de planificación clásicos. Visto esta dificultad para incluir la consideración de múltiples parámetros en el proceso de planificación, en WorkflowSim con DVFS se implementó un motor borroso, el cual permite realizar dicha toma de decisión en función de más parámetros con una relativa facilidad. Dado que el simulador está implementado en Java, para la implementación del FRBS se procedió a implementar este mediante la librería jFuzzyLogic de Java, que proporciona un proceso sencillo de implementación de un sistema borroso. Mediante esta librería, se consigue la capacidad de implementar algoritmos de planificación los cuales utilizan FRBS, mediante el que se puede implementar dicha planificación multi-parámetro previamente mencionada, considerando así otros parámetros tales como los anteriormente mencionados (MIPS, tiempos, tamaño de tareas, etc.) además de la energía consumida por una VM, logrando así, al tener más parámetros en consideración, realizar un mejor proceso de planificación.

Para este proceso de planificación mediante FRBS, tal y como se explicó en el apartado 3.1.1, se procederá a, en este sistema, especificar qué parámetros serán los antecedentes, y qué parámetros serán los consecuentes, así como definir las funciones de pertenencia para estos, y la base de reglas en la cual se apoyará el motor de inferencia para realizar el proceso de inferencia. En WorkflowSim con DVFS, se procedió a implementar tres nuevos algoritmos de planificación basados en FRBS, debiendo destacar los dos primeros basados en los algoritmos MaxMin y MinMin, explicados a continuación.

- PowerAwareFuzzyMaxMaxSchedulingAlgorithm:

Está basado en el algoritmo PowerAwareMaxMinSchedulingAlgorithm, de manera que su proceso de planificación se divide en dos etapas. La primera etapa de planificación consiste en buscar la tarea que requiera de mayores recursos computacionales para ser ejecutada de las presentes en la lista de tareas a planificar por el Scheduler. Una vez finalizado el proceso de búsqueda de dicha tarea comenzará la segunda etapa de planificación, la cual es común tanto en este algoritmo como en PowerAwareFuzzyMinMaxSchedulingAlgorithm, de manera que se procederá a explicar más adelante, tras explicar la primera etapa de este.

- PowerAwareFuzzyMinMaxSchedulingAlgorithm:

Algoritmo basado en PowerAwareMinMinSchedulingAlgorithm, por lo que su proceso de planificación está dividido en dos etapas. En la primera etapa, la entidad Scheduler procederá a buscar de la lista de tareas a planificar para ser ejecutadas en las VM del Datacenter aquella tarea que requiera de una menor cantidad de recursos para ser completada. Una vez encontrado dicha tarea, proseguirá la planificación en la segunda etapa.

La segunda etapa de planificación de los algoritmos anteriormente expuestos utiliza FRBS para realizar la toma de decisiones. FRBS está implementado de tal manera que cuenta con 5 antecedentes y 1 consecuente. Estos antecedentes son la cantidad de MIPS, la potencia de consumo, el tamaño de la tarea, el tiempo de ejecución de la tarea y la energía consumida por la VM seleccionada para su procesamiento. El consecuente consiste en la tasa de selección, de manera que, irá comprobando para todas y cada una de las VM que estén en estado ocioso en la entidad Datacenter los antecedentes anteriormente expuestos para la tarea seleccionada, de manera que asignará la tarea en cuestión a la VM la cual obtenga la mayor tasa de selección del proceso de evaluación del FRBS.

4.3. Segunda fase: implementación del Meta-Planificador

En esta segunda fase, se procede a explicar detalladamente el proceso llevado a cabo para la creación del simulador con Meta-Planificador, habiendo utilizado como base el simulador WorkflowSim con DVFS.

4.3.1. Introducción al Meta-Planificador

Con objeto de facilitar la comprensión de este TFG al lector, se procede, previo a los apartados donde se detalla la implementación del Meta-Planificador, a realizar una introducción detallada de qué es un Meta-Planificador, explicando su funcionamiento, sus bondades y sus debilidades.

Gracias a los Meta-Planificadores, se consigue la unión de múltiples *datacenters* para ser utilizados de manera conjunta como un único recurso centralizado, en lugar de como varios recursos independientes y descentralizados. Esto permite ahorro económico al poder prescindir de adquirir nuevo un *datacenter* capaz de cubrir las necesidades computacionales que por otra parte son capaces de ofrecer el citado conjunto de *datacenters* funcionando de manera centralizada gracias al Meta-Planificador. A continuación, se expone en la Figura 8 una breve aproximación a la estructura de un Meta-Planificador.

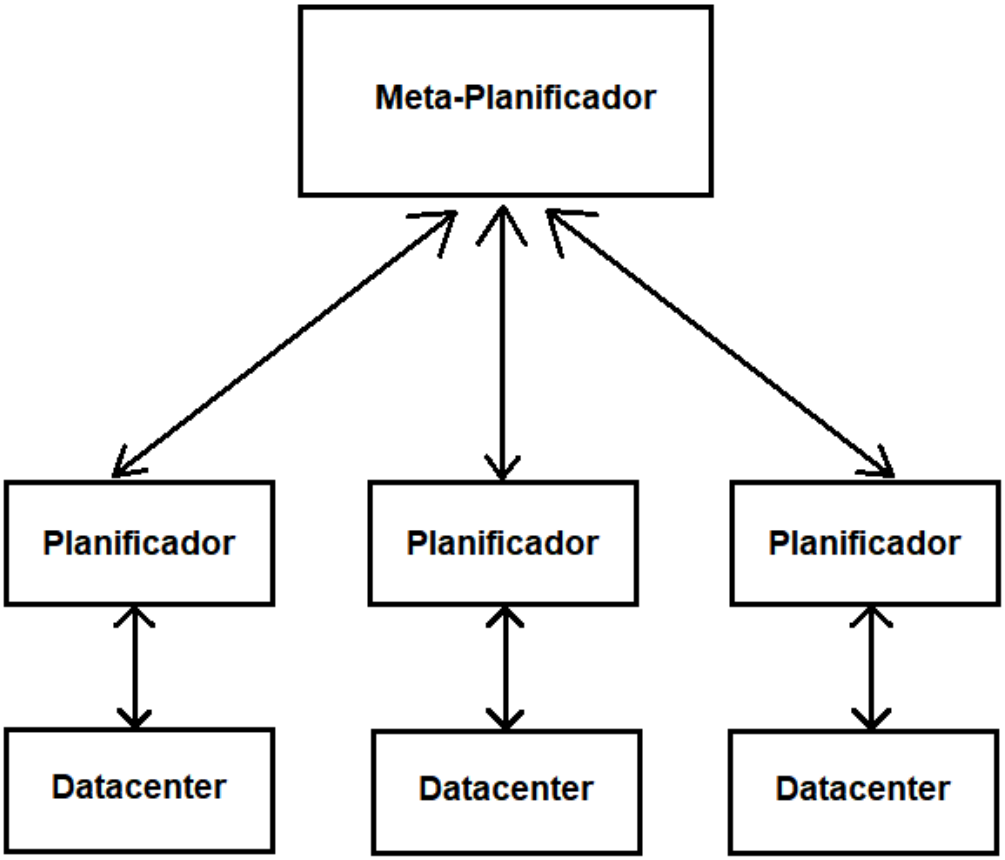

Figura 8: Estructura básica de un Meta-Planificador

Tal y como se muestra en la Figura 8, el Meta-Planificador constituye una entidad de la estructura de un *cloud* posicionándose a un nivel superior respecto a los Schedulers. Este Meta-Planificador, poseerá una lista de tareas a ser repartidas entre los diferentes *datacenters* para su procesamiento, de manera que, para realizar esto, esta entidad procederá a obtener las características de dichos *datacenters* para así proceder a realizar el proceso de planificación mediante el cual realizará la toma de decisiones de hacia qué *datacenter* será enviada cada una de las tareas que esta entidad posee. Una vez que el Meta-Planificador ha realizado dicha toma de decisión, procederá a enviar las tareas las cuales han sido planificadas a ser ejecutadas en cada *datacenter* a sus respectivos planificadores, para que estos, posteriormente, procedan a realizar el proceso de planificación mediante el cual realizarán localmente la planificación de tareas para decidir la asignación y envío de tareas a cada una de las máquinas disponibles para realizar el procesado de dichas tareas. Las tareas procesadas serán devueltas a los planificadores, que procederán a enviar el resultado del procesamiento de dichas tareas completadas al Meta-Planificador.

La principal bondad del Meta-Planificador, tal y como ya ha sido mencionado con anterioridad, es la capacidad de permitir el uso de múltiples *datacenters* de manera no independiente, es decir, de manera que estos funcionen, aproximadamente, como un único CPD. Gracias a dicha entidad Meta-Planificadora, se evita que los diferentes *datacenters* estén aislados, lo que provocaría que, en caso de requerir la ejecución de múltiples conjuntos de tareas, estas deberían ser asignadas y planificadas por parte del usuario, de manera que este decidiera hacia qué *datacenters* deben ser enviadas para su procesado, lo cual puede inducir con facilidad a una incorrecta asignación de dichas tareas, debido a ser realizado de manera manual por el usuario.

El funcionamiento centralizado y no independiente que otorga la inclusión de un Meta-Planificador otorga dicha capacidad de planificación y control automatizado de todos los recursos computacionales disponibles, de manera que los usuarios podrán realizar el envío de los conjuntos de tareas al Meta-Planificador, el cual, en nombre de los usuarios, procederá a realizar una correcta planificación de dichos conjuntos de tareas, gracias al conocimiento del estado de cada uno de los *datacenters*. Con el uso de los algoritmos de planificación a nivel de Meta-Planificación (algoritmos de Meta-Planificación de ahora en adelante), este podrá no solo realizar dicha correcta Meta-Planificación de las tareas, sino que además podrá soportar planificaciones más complejas y avanzadas, en las cuales optimice no solo tiempos, sino también otros parámetros de interés tales como consumos, el cual es uno de los objetivos de este TFG.

Un Meta-Planificador, a pesar de poder realizar la planificación con el objetivo de poder utilizar más de un *datacenter* de manera simultánea, posee la desventaja de que este solo puede acceder a información resumida y general, sin conocer ni poder acceder a información detallada y completa de estos. A causa de no disponer de acceso a información completa del estado, puede implicar que el Meta-Planificador realice malas planificaciones o planificaciones no óptimas, las cuales, por otra parte, no existirían si se prescindiera de utilizar múltiples *datacenter*, ya que el planificador local de un *datacenter* dispone de acceso completo a las características de este, pudiendo así realizar dicha correcta planificación.

A pesar de las debilidades anteriormente expuestas, las bondades que proporciona el utilizar un Meta-Planificador justifican su uso, al proporcionar la capacidad de utilizar un conjunto de *datacenters* de manera centralizada por medio de una única entidad, permitiendo, mediante un correcto algoritmo de planificación, realizar un proceso de planificación tal que arroje unos resultados que, aun no siendo óptimos, permitan utilizar los recursos disponibles de manera que se consiga un resultado lo suficientemente bueno

como para evitar la migración a un único y exclusivo *datacenter*, permitiendo así un ahorro significativo de costes en adquisición y migración a dicho nuevo *datacenter*.

4.3.2. Elección del simulador para la implementación del Meta-Planificador

Para poder realizar la simulación de un entorno de *cloud* con Meta-Planificador, con objeto de poder realizar un estudio de los algoritmos de Meta-Planificación y poder realizar un algoritmo enfocado al ahorro energético, en primera instancia, era necesario disponer de dicho entorno de simulación el cual contase con esta entidad. Para ello, inicialmente se disponían de dos opciones posibles, siendo estas el desarrollo de un entorno de simulación que contara con Meta-Planificadores o, por otro lado, elegir un simulador de los ya disponibles desarrollados de entornos Cloud y proceder a implementar esta nueva entidad sobre este.

Inicialmente, se procedió a estudiar ambas posibilidades. De este estudio, se llegó a la conclusión de que la opción más óptima consistía en realizar la búsqueda de un entorno de simulación de *cloud* sobre el que proceder a realizar la implementación de dicho Meta-Planificador. Por razones obvias, el desarrollo desde cero de un nuevo entorno de simulación *cloud* fue descartado debido a la gran complejidad de desarrollo, debido a la gran cantidad de componentes con los que estos sistemas cuentan, para lo cual se requeriría una gran cantidad de recursos temporales para poder proceder a realizar su desarrollo. Es por ello que se procedió a buscar un entorno de simulación sobre el cual desarrollar dicho entorno, siendo este el motivo por el que se ha procedido en el apartado 3.2 a realizar el estudio de los entornos de simulación citados.

Para poder proceder a desarrollar el sistema de Meta-Planificación, se debía escoger un simulador tal que fuera lo suficientemente modular como para permitir una fácil y rápida implementación de esta nueva entidad, así como la capacidad de realizar estudios de consumo de potencia sobre este, capacidad de trabajar con flujos de trabajo, y fácil implementación de algoritmos de planificación. El simulador por el cual se optó para implementar dicho sistema de Meta-Planificación es el simulador WorkflowSim con DVFS. El motivo por el cual este entorno de simulación ha sido elegido para realizar el desarrollo de este TFG es debido a que cumple con todos los requisitos necesarios para el desarrollo de este TFG.

En primer lugar, dicho simulador cuenta con una estructura de gran modularidad, ya que al estar desarrollado mediante entidades, las cuales representan a los diferentes elementos del sistema Cloud, permite una fácil y rápida implementación de una nueva entidad intermedia entre las entidades Engine y Scheduler, que es dónde estará situado

el Meta-Planificador (se verá detallado más adelante). Además, gracias al sistema de comunicación entre entidades mediante eventos identificados por etiquetas, se permite, una vez más, una gran modularidad en el entorno de simulación, ya que permite adaptar fácilmente las comunicaciones entre entidades, permitiendo añadir nuevos tipos de eventos necesarios para las comunicaciones entre entidades tras la inclusión de esta nueva.

En segundo lugar, dicho simulador cumple con los requisitos de disponer de soporte para simular flujos de trabajo y la capacidad de realizar la medición de consumos en la simulación, lo que permitirá agilizar el proceso de implementación del Meta-Planificador, al no ser necesario implementar dichas características ya implementadas que son necesarias para el estudio de los algoritmos de Meta-Planificación basados en la mejora de consumos.

En tercer lugar, el simulador cuenta con algoritmos de planificación del tipo FRBS mediante jFuzzyLogic con el objetivo de optimizar consumos de potencia, que podrán ser utilizados como base para el desarrollo de los algoritmos de Meta-Planificación, facilitando su implementación, ya que, para el desarrollo de este TFG, resulta de interés el empleo de algoritmos de Meta-Planificación con la función objetivo de optimizar consumos.

En cuarto y último lugar, este simulador permite realizar la simulación de modelos de potencia, lo que permite la simulación de procesadores reales. Además, cuenta con la inclusión e implementación de un modelo de potencia analítico, tal y como se explica en el apartado 3.2.4, de manera que permite realizar la implementación de procesadores no reales con características ficticias, lo que facilita en gran medida el desarrollo y estudio de algoritmos de Meta-Planificación, situación ante la cual no se necesita la implementación de un modelo de potencia basado en un procesador real.

4.3.3. Implementación del Meta-Planificador

En el presente apartado se procede a explicar las modificaciones que han sido necesarias realizar en el simulador WorkflowSim con DVFS para realizar la implementación de la nueva entidad Meta-Planificador. Este apartado se centra principalmente en las modificaciones que han sido llevadas a cabo a nivel de entidades, núcleo del simulador, y comportamiento entre entidades para realizar la citada implementación de la nueva entidad Meta-Planificadora, excluyendo las implementaciones de los algoritmos de Meta-Planificación, realizando sólo algunas

menciones sobre estos, ya que ha sido necesario realizar ciertas modificaciones para poder realizar la posterior implementación de dichos algoritmos.

Para realizar la implementación del Meta-Planificador, ha sido necesario, en primera instancia, realizar la creación de la entidad MetaScheduler, que se encargará de realizar el proceso de Meta-Planificación. Dicha nueva entidad, debe posicionarse entre las entidades Engine y las entidades Schedulers, ya que ahora estas no deben poseer comunicación directa entre sí, sino que deben comunicarse a través de esta nueva entidad, para así, de esta forma, poder implementar el comportamiento de Meta-Planificación en el simulador. Así mismo, las tareas que deben ser enviadas a los *datacenters* para ser procesadas, sean enviadas por la entidad Engine al MetaScheduler, de manera que este procederá a, mediante el algoritmo de Meta-Planificación seleccionado, a realizar la Meta-Planificación de dichas tareas, estudiando estas y el estado de todos los Datacenters del entorno de simulación, para así decidir, para cada Datacenter, el conjunto de tareas que debe procesar, y así, una vez realizado dicho proceso de Meta-Planificación, proceder el MetaScheduler a enviar dicho conjunto de tareas las cuales deben procesar los Datacenters a sus respectivos planificadores locales, para que estos procedan a realizar la planificación local de las tareas.

Se ha modificado el nivel de conocimiento entre entidades, es decir, en el simulador WorkflowSim con DVFS, la entidad Engine conocía la existencia de las entidades Schedulers, de manera que esta poseía un listado de todos los planificadores existentes, así como sus identificadores. De igual manera, las entidades Schedulers conocían la existencia de la entidad Engine, poseyendo el identificador de este. Esto implicaba que cuando una entidad Scheduler necesitaba tareas que procesar, procedía a solicitar a la entidad Engine el envío de tareas a procesar, y así, esta, procedía a enviar posteriormente tareas que procesar a la entidad Scheduler. Con la implantación de la entidad MetaScheduler, dicho flujo de mensajes de eventos ha sido modificado. Ahora, el intercambio de mensajes e información debe ser realizado a través de esta nueva entidad. Esto implica que, ahora, la entidad Engine desconocerá la existencia de las entidades Schedulers, de manera que esta no tendrá información alguna sobre estos ni sus identificadores, teniendo, no obstante, conocimiento de la existencia del MetaScheduler y conociendo su identificador. De igual manera, las entidades Scheduler no conocerán a la entidad Engine ni su identificador, pero sí conocerán a la entidad MetaScheduler y su identificador, teniendo total comunicación con esta. Por otra parte, para que este flujo de información entre las entidades Engine y Schedulers a través de la nueva entidad MetaScheduler sea posible, esta poseerá total conocimiento de la existencia de la entidad

Engine y su identificador, así como de todos los Schedulers y sus respectivos identificadores, de esta manera, se permite dicho flujo de información mencionado.

A continuación, se adjunta una Figura 9 donde se compara el sistema de simulación con y sin Meta-Planificador.

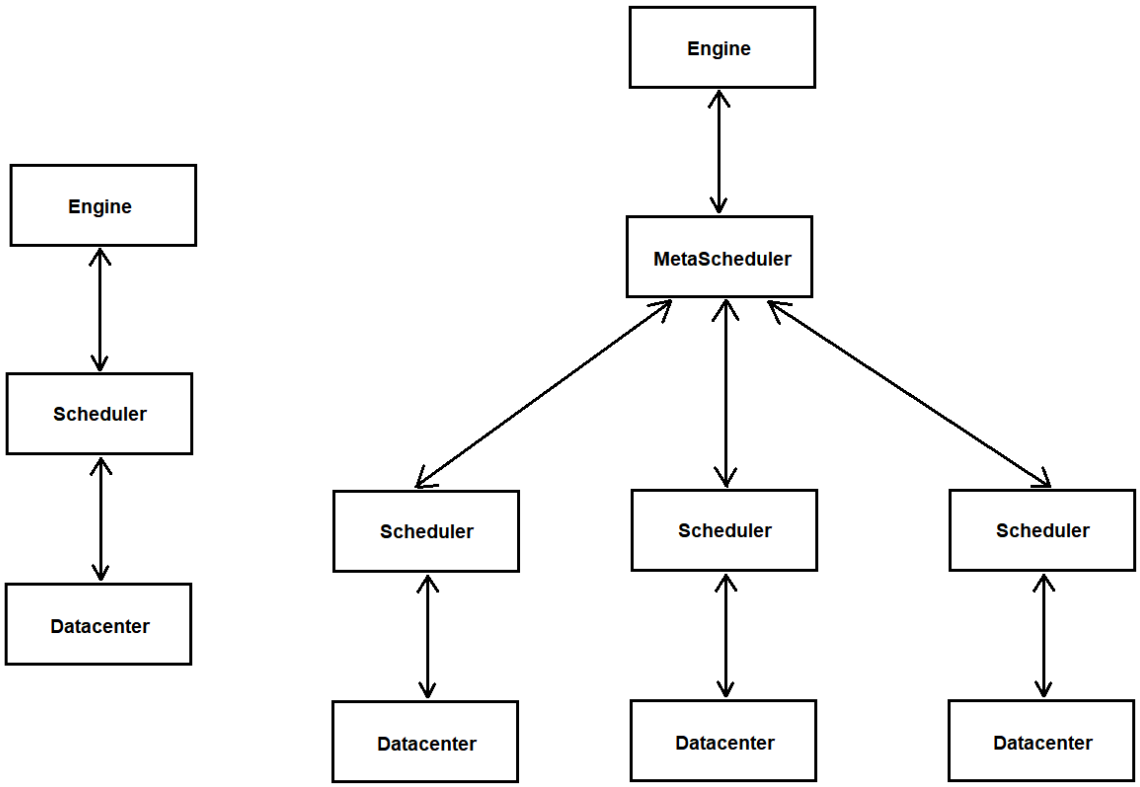

Figura 9: Scheduler vs MetaScheduler

Esta implementación del Meta-Planificador implica una serie de modificaciones en el sistema de eventos, las cuales son expuestas a continuación.

En primer lugar, previamente, cuando una entidad Scheduler necesitaba nuevas tareas que procesar, esta solicitaba a la entidad Engine, de la cual poseía conocimiento total, nuevas tareas a procesar. Esta acción ahora es solicitada a través de la entidad MetaScheduler, lo que implica que, una entidad Scheduler procederá a solicitar nuevas tareas a la entidad MetaScheduler, y esta, procederá a pedir dichas nuevas tareas a la entidad Engine en su nombre.

En segundo lugar, continuando con el comportamiento de la solicitud y entrega de tareas, una vez que una entidad Engine recibe una solicitud de tareas, mientras que previamente esta procedía a enviar dichas tareas directamente a las entidades Schedulers, ahora deberá proceder a enviar estas a la entidad MetaScheduler, que procederá a realizar el proceso de Meta-Planificación para realizar la toma de decisiones

de asignación de tareas a cada Datacenter. Tras dicho proceso, procederá a enviar los conjuntos de tareas a procesar a las entidades Schedulers de los respectivos Datacenters que deben procesar estas.

En tercer lugar, cuando una tarea ha sido ejecutada por una entidad Datacenter y estas son devueltas a su entidad Scheduler, previamente el resultado de estas tareas era enviado directamente a la entidad Engine. Este proceso ahora debe ser realizado mediante el envío de dicho resultado de la ejecución de dichas tareas a la entidad MetaScheduler, entidad que procederá a enviar este resultado a la entidad Engine.

En cuarto lugar, una vez que la entidad Engine no posee más tareas que procesar, previamente esta enviaba, mediante un evento, la solicitud de finalización de simulación a las entidades Schedulers, para que estas procedieran a finalizar su ejecución mediante la destrucción de las VMs creadas en sus respectivos Datacenters. Ahora, este proceso se realiza mediante el envío de dicha solicitud de finalización de simulación a la entidad MetaScheduler, la cual procederá a comunicar a todas las entidades Schedulers que procedan a finalizar la ejecución de la simulación.

A continuación, se exponen todas las clases Java que han sido modificadas y creadas para realizar la implementación del Meta-Planificador.

- Parameters
- WorkflowPlanner_META
- WorkflowParser
- ClusteringEngine_META
- WorkflowEngine_META
- WorkflowMetaScheduler
- WorkflowScheduler_META
- WorkflowSimTags
- GeneralParametersMetaFuzzy
- BaseSchedulingAlgorithm
- CloudSim
- WorkflowDVFSBasicMeta

Una vez citadas las clases las cuales han sido necesarias crear o bien modificar para adaptar e implementar el Meta-Planificador, se procede a explicar detalladamente los cambios realizados en cada una de estas clases.

- Parameters:

Las modificaciones realizadas a esta clase Java son mínimas y no influyen en el comportamiento que esta clase posee al ser ejecutada por el resto de simuladores basados en CloudSim, es por ello que no ha sido necesario crear una nueva clase basada en esta adaptada para el nuevo simulador con Meta-Planificador. Las modificaciones realizadas implican la adición de una lista definida con los diferentes algoritmos de Meta-Planificación disponibles a utilizar por el Meta-Planificador. Además, se añade un nuevo método init() de inicialización de la clase estática Parameters para establecer el algoritmo de Meta-Planificación que será utilizado durante la simulación, para lo cual ha sido implementado un nuevo atributo llamado MetaSchedulingAlgorithm, en el cual se definen la lista de algoritmos de Meta-Planificación.

- WorkflowPlanner_META:

Basado en la clase Java WorkflowPlanner de WorkflowSim con DVFS. Se ha procedido a aplicar cambios mínimos en el funcionamiento de este. Ahora, en lugar de crear una entidad ClusteringEngine, procederá a crear una entidad ClusteringEngine_META, la cual está adaptada para el entorno de simulación con Meta-Planificador, y, además, para la creación de la entidad Parser, se procede a asignar como ID de usuario de dicha entidad Parser el ID de la entidad MetaScheduler. De esta forma, mediante la especificación del MetaScheduler como usuario de las tareas, se procede a, posteriormente, asegurar el envío de dichas tareas a dicha entidad, que realizará el proceso de Meta-Planificación, para realizar el reparto de tareas a las diferentes entidades Schedulers, los cuales realizarán la planificación local de dichas tareas en sus respectivos Datacenters.

- WorkflowParser:

Se ha procedido a aplicar una pequeña modificación en el proceso mediante el cual obtiene la lista de tareas almacenadas en el fichero DAX y convierte estas a un listado de tareas con la que puede trabajar el simulador. La modificación realizada consiste en, durante este proceso de adaptación de tareas, en el cual convierte cada tarea de manera individualizada para que el simulador pueda posteriormente trabajar con estas, se ha procedido a añadir una pequeña sentencia mediante la cual en cada proceso de conversión de dichas tareas, comprobará el tamaño de cada una de estas, realizando la búsqueda de la tarea más pequeña y de la más grande, Debido a que esta información resulta de interés para posteriores procesos de Meta-Planificación basada en reglas borrosas. Dicha información de la mínima y máxima, será almacenada en la clase estática GeneralParametersMetaFuzzy en los atributos maxDaxLength y minDaxLength.

- ClusteringEngine_META:

Basado en la clase java ClusteringEngine de WorkflowSim con DVFS. Posee un funcionamiento similar, consistiendo la modificación aplicada respecto a la clase original en el proceso mediante el cual este originalmente entregaba el listado completo de tareas a la entidad Scheduler. Ahora, esta entidad procede a enviar dicho listado completo al MetaScheduler, para así, posteriormente, proceder este a enviar dicho listado completo de tareas a la entidad WorkflowEngine_META, que procederá a realizar el proceso mediante el cual analizará dicho listado de tareas para observar qué tareas podrán ser o no ejecutadas en función de sus restricciones de ejecución debido a su estructura de ejecución jerárquica con restricciones parentales. Por otra parte, ahora esta entidad procede a crear una entidad WorkflowEngine_META en lugar de una entidad WorkflowEngine, debido a que esta ha sido adaptada para funcionar en el nuevo entorno de simulación adaptado a Meta-Planificación.

- WorkflowEngine_META:

Clase basada en la clase WorkflowEngine del simulador WorkflowSim con DVFS. Para el proceso de adaptación a funcionar con la nueva entidad MetaScheduler, se ha procedido a realizar una serie de modificaciones las cuales son expuestas a continuación.

Eliminación de la lista de Scheduler y adición del MetaScheduler y su ID. Esta acción ha sido llevada a cabo debido a que ahora, la entidad Engine no poseerá acceso directo a las entidades Schedulers, sino que esta poseerá acceso a la entidad MetaScheduler, la cual sí poseerá acceso a los planificadores locales. Esto supone que toda comunicación que deba realizar una entidad Engine con las entidades Schedulers y viceversa, deberá ser realizada a través del MetaScheduler, para que así se pueda realizar el proceso de Meta-Planificación.

En la instanciación y creación de la entidad Engine, se ha agregado la creación de la entidad MetaScheduler, y se especifica a esta el número de Schedulers, para que, posteriormente, dicha entidad MetaScheduler, proceda a crear el número de planificadores especificado. Además, en el proceso de creación, la entidad Engine guardará la entidad MetaScheduler y su identificador en los atributos internos de la clase Engine.

El proceso de creación de VMs ha sido modificado, ya que inicialmente la clase Engine enviaba dicha petición de creación de VM a los Schedulers de manera directa, mientras que ahora, estas listas son enviadas a la entidad MetaScheduler, la cual procederá a enviar cada una de estas a sus respectivos Schedulers para que procedan a crear dichas VM en sus respectivos Datacenters.

Modificación del método bindSchedulerDatacenter. Dicho método realiza la función de vincular a un Scheduler su respectivo Datacenter. Originalmente, el funcionamiento de este consistía en enviar a un determinado Scheduler mediante un evento, el identificador de su respectivo Datacenter, para el cual este debía realizar el proceso de vinculación. Debido a la implementación del Meta-Planificador, este proceso no puede ser realizado directamente desde la entidad Engine, de manera que ahora, la entidad Engine, procede a llamar a un método de la clase MetaScheduler de manera que, dado que este sí posee el listado completo de Schedulers, poder proceder este a llamar al método bindSchedulerDatacenter de la clase Scheduler mediante el que podrá proceder a ordenar a estas que realicen el proceso de vinculación con un determinado Datacenter.

Modificación del proceso de finalización de simulación. Previamente, este proceso el cual es generado por la entidad Engine, procedía a enviar un evento END_OF_SIMULATION a cada una de las entidades Schedulers existentes. Ahora, debido a la implementación del MetaScheduler, esta acción no puede ser directamente realizada por la entidad Engine, de manera que esta procederá a enviar dicho evento a la entidad MetaScheduler, que procederá a reenviar dicho evento a todas las entidades Schedulers, para informar de la no existencia de más tareas, para que así se proceda a finalizar la simulación, realizando el proceso de destrucción de VMs.

Modificación del método submitJobs. Inicialmente, en la entidad Engine de WorkflowSim, esta, ante una petición de nuevas tareas por parte de una entidad Scheduler, procedía a analizar las tareas pendientes por ser procesadas del listado de tareas, para observar cuales pueden y no pueden ser ejecutadas, de manera que, una vez que eran encontradas un conjunto de tareas que podían ser ejecutadas, procedía a enviar dicho conjunto de tareas a la entidad Scheduler la cual había realizado dicha petición de tareas. Debido a la implementación del Meta-Planificador, este proceso ha sido modificado, ya que ahora la entidad Engine desconocerá el listado de Schedulers, de manera que ante una petición de nuevas tareas, la cual le llegará a través de la entidad MetaScheduler, que sí conoce su presencia, procederá a enviar dicho listado de tareas a esta entidad, para que esta proceda posteriormente a realizar el proceso de Meta-Planificación mediante el cual entregará las tareas a los diferentes Schedulers para que realicen la planificación local de estas. Además, anteriormente, el funcionamiento de la entidad Engine en las antiguas versiones del simulador permitía trabajar con múltiples Schedulers, con un funcionamiento primitivo, el cual consistía en un reparto equitativo de las tareas a procesar entre todas las entidades Schedulers presentes en el entorno de simulación, de manera que se realizaba una pobre planificación en el reparto de tareas entre los diferentes Datacenters que constituían el entorno de simulación, motivo por el

cual se ha procedido a incluir a la entidad MetaScheduler, la cual puede realizar planificaciones más complejas basadas en el estado y características de los Datacenters.

- WorkflowMetaScheduler:

Clase basada en la clase WorkflowScheduler de WorkflowSim con DVFS. Dado que un Meta-Planificador es en sí, un planificador, el cual en lugar de planificar tareas para ser procesadas en un Datacenter al cual estará vinculado, realizará un proceso de planificación de planificadores, donde se asignará tareas a estos planificadores en función de las características de sus respectivos Datacenters, ha sido utilizada la clase Java anteriormente especificada. Esto es debido a la gran similitud de comportamiento de esta nueva entidad con la entidad en la que está basada. Para la adaptación de comportamiento a Meta-Planificador, han sido llevadas a cabo una serie de modificaciones las cuales han sido expuestas a continuación.

Modificación del flujo de comunicación entre entidades:

Como parte del proceso necesario para poder establecer la comunicación entre las entidades Schedulers las cuales demanden tareas para ser ejecutadas, en la entidad WorkflowMetaScheduler ha sido implementado un nuevo tipo de evento llamado CLOUDLET_SUBMIT_META. La funcionalidad de este evento es la de permitir que las entidades Schedulers soliciten a la entidad MetaScheduler nuevas tareas, ya que estas no pueden comunicarse directamente con la entidad WorkflowEngine para pedir nuevas tareas. De esta manera, cuando el MetaScheduler reciba dicho evento por parte de algún Scheduler, este procederá a ejecutar el método processCloudletSubmitMeta, mediante el que este procederá a solicitar a la entidad Engine que le envíe nuevas tareas para así proceder a realizar el proceso de Meta-Planificación de tareas mediante el cual asignará, dependiendo del algoritmo de Meta-Planificación utilizado, tareas a las diferentes entidades Schedulers. La entidad MetaScheduler procederá a solicitar a la entidad WorkflowEngine el envío de nuevas tareas mediante el envío de un evento CLOUDLET_SUBMIT.

Tras la modificación del simulador que supone la inclusión de la entidad MetaScheduler, los Schedulers no podrán realizar la petición de tareas directamente a la entidad Engine debido a que ahora, la entidad MetaScheduler se situará como una entidad intermedia entre estas, para así de esta forma poder proceder a realizar el proceso de Meta-Planificación mediante el cual se consigue el proceso de centralización de los diferentes Datacenters del entorno de simulación. Para conseguir esto, como ya ha sido citado, la entidad MetaScheduler se sitúa como una entidad intermedia entre las entidades Engine y Schedulers, de manera que esta poseerá el listado de Schedulers

disponibles del entorno de simulación, incluyendo los identificadores de estos para poder establecer comunicación mediante el sistema de eventos, así como el identificador de la entidad Engine para poder proceder a establecer comunicación con este. De igual manera, las entidades Schedulers ya no poseerán el identificador de la entidad Engine, sino que poseerán el identificador de la entidad MetaScheduler, sucediendo lo mismo para el caso de la entidad Engine, la cual desconocerá los identificadores de los Schedulers y el número de estos, conociendo ahora sólo la presencia de la nueva entidad MetaScheduler, así como su identificador, para poder establecer comunicación con este.

Por otra parte, se ha procedido a modificar el método getScheduler de WorkflowSim con DVFS a getMetaScheduler, que procede a utilizar los algoritmos de Meta-Planificación implementados en este nuevo entorno de simulación.

Finalmente, el proceso de planificación ha sido modificado para ser adaptado al comportamiento del proceso de Meta-Planificación. Esta adaptación ha sido implementada en el método processCloudletUpdate. Ahora, en lugar de planificar las tareas para ser enviadas a las VMs del Datacenter, ha sido modificado para enviar las tareas al planificador correspondiente a este. Para implementar dicha modificación, se ha procedido a crear una lista llamada ScheduledListMeta, en la cual, se almacenará una lista de tareas por cada planificador, de manera que, posteriormente, se procederá a acceder a dicha lista de tareas de cada planificador, obteniendo el identificador de usuario (identificador del planificador) el cual identifica al planificador hacia el que debe ser enviada la lista de tareas en cuestión. A continuación se adjunta un fragmento de código de la clase processCloudletUpdate en el cual se muestra el uso de dicha lista para la obtención de tareas para ser enviadas a su correspondiente planificador.

```
List<List<Job>> listSchedulers = metaScheduler.getScheduledListMeta();
for (int i = 0; i < listSchedulers.size(); i++)
{
      if (!listSchedulers.get(i).isEmpty())
      {
      sendNow(listSchedulers.get(i).get(0).getUserId(),
CloudSimTags.CLOUDLET_SUBMIT, listSchedulers.get(i));
      }
   }
```

El objeto MetaScheduler corresponde al objeto que ha realizado el proceso de Meta-Planificación, el cual ha procedido a planificar las tareas de acuerdo a las características de los Datacenters disponibles, asignando las tareas en una lista que contiene una lista

de tareas por cada Datacenter, de manera que, utilizando el método getScheduledListMeta(), se puede obtener dicha lista planificada, para la que, posteriormente, mediante el bucle *for* mostrado, se obtendrá el total de listas para así, si dichas listas poseen tareas, enviar estas a sus respectivos planificadores mediante el evento CLOUDLET_SUBMIT.

- WorkflowScheduler_META:

Basado en la clase WorkflowScheduler del simulador WorkflowSim con DVFS. Las modificaciones realizadas a esta respecto a su clase original son mínimas, para adaptar su funcionamiento con el objeto de que funcione de manera correcta con la nueva entidad de Meta-Planificación. Estas modificaciones pasan por, en primera instancia, eliminar el identificador de la entidad Engine de estas entidades, debido a que ahora, no podrán establecer comunicación con la entidad Engine, ya que deberán establecer comunicación con la entidad MetaScheduler, para lo cual ha sido implementado el identificador de esta última, que es establecida en el proceso de creación de cada entidad Scheduler.

Por otra parte, ha sido incluido un nuevo tipo de evento llamado CLOUDLET_SUBMIT_META. Este es enviado por parte de las entidades Schedulers a modo de petición de nuevas tareas para su procesado, de manera que, mediante el método submitCloudlets de la entidad Schedulers, procederá a realizar dicha petición de nuevas tareas para su procesado a la entidad Engine. Esta petición se realizará por medio de la entidad MetaScheduler, ya que se recuerda que, debido a la nueva estructura del simulador, las entidades Scheduler no poseerán comunicación directa con la entidad Engine.

- WorkflowSimTags:

Se ha procedido a añadir la nueva etiqueta CLOUDLET_SUBMIT_META previamente mencionada en las entidades WorkflowMetaScheduler y WorkflowScheduler_META, necesaria para establecer la comunicación de peticiones de nuevas tareas por parte de las entidades Schedulers a la entidad Engine a través de la entidad MetaScheduler.

- GeneralParametersMetaFuzzy:

Clase estática de Java creada para disponer de acceso a las entidades Datacenter desde el Meta-Planificador. El motivo por el que la entidad MetaScheduler necesita disponer de acceso a esta lista de Datacenters es debido a que este necesita disponer de conocimiento de no solo el número de Datacenters del entorno de simulación, sino

también del estado de estos, para posteriormente poder utilizar dicha información en el proceso de Meta-Planificación. El proceso de asignación de los Datacenters es detallado en la descripción de las modificaciones aplicadas a la clase CloudSim.

Además, en esta clase ha sido implementado un Map llamado bindedDatacenterToScheduler el cual es utilizado para guardar la asociación Datacenter-Scheduler. Este Map permite, posteriormente, que los algoritmos de Meta-Planificación puedan conocer para cada Datacenter su correspondiente Scheduler, de manera que, una vez realizado el proceso de Meta-Planificación, pueda proceder a asignar dicha tarea para ser enviada a la lista de tareas a enviar para el correspondiente planificador de dicho Datacenter, para que este proceda a realizar el proceso de planificación local.

Por otra parte, se añaden una serie de parámetros que son necesarios para el proceso de Meta-Planificación por medio de FRBS. Estos parámetros son los MIPS máximos y mínimos de los Datacenters, las potencias máximas y mínimas de estos, la longitud máxima y mínima del total de tareas a procesar, los tiempos máximos y mínimos de ejecución de tareas y la energía máxima y mínima consumida por el procesado.

Para la obtención de los tiempos máximos y mínimos de ejecución del total de tareas, se procede a, una vez obtenido los MIPS máximos y mínimos de todos los Datacenters presentes, así como las longitudes máximas y mínimas de tareas, a calcular estos tiempos máximos y mínimos de ejecución de manera que, para la obtención del tiempo máximo de ejecución, se realizará la división de la tarea de mayor tamaño entre el valor mínimo de MIPS, mientras que para la obtención del tiempo mínimo, se realizará la división de la tarea más pequeña entre el valor de MIPS máximos. A continuación se adjuntan las ecuaciones 10 y 11 en las cuales se puede observar el cálculo de dichos tiempos.

$$T_{max} = \frac{L_{max}}{MIPS_{min}} \tag{10}$$

$$T_{min} = \frac{L_{min}}{MIPS_{max}} \tag{11}$$

Por otra parte, para la obtención de la energía máxima y mínima, se realizará, para el caso de la energía máxima, el producto de la potencia máxima y el tiempo máximo de ejecución, mientras que para el cálculo de la energía mínima, se realizará el producto de la potencia mínima por el tiempo de ejecución mínimo. A continuación se adjuntan las ecuaciones 12 y 13 en las cuales se puede observar el cálculo de dichos parámetros de energía.

$$E_{max} = P_{max} * T_{max} \tag{12}$$

$$E_{min} = P_{min} * T_{min} \quad (13)$$

El uso de estos parámetros quedará explicado en el próximo apartado dedicado a los algoritmos de Meta-Planificación empleados en la entidad Meta-Planificador.

- BaseSchedulingAlgorithm:

Esta clase ha tenido que ser modificada para añadir una lista que almacenará el conjunto de Schedulers, que poseerá el listado de todos los planificadores del entorno de simulación. Dicha lista será establecida por la entidad MetaScheduler en el proceso de planificación, de manera que, posteriormente, los algoritmos de planificación, los cuales heredan de dicha clase, podrán disponer de acceso a dicho listado de planificadores para realizar el proceso de Meta-Planificación. Por otra parte, para poder realizar la Meta-Planificación, esta entidad necesitará disponer de acceso a la lista de Datacenters. Debido a que los algoritmos tanto de planificación como de Meta-Planificación heredan de la clase BaseSchedulingAlgorithm, en esta clase ha sido implementado una lista de Datacenters, de manera que el Meta-Planificador, en el método processCloudletUpdate, mediante el cual procede a realizar el proceso de Meta-Planificación, obtendrá de la clase estática GeneralParametersMetaFuzzy el listado de Datacenters con su estado, de manera que, previo a empezar el proceso de Meta-Planificación, procederá a crear un objeto de la clase BaseSchedulingAlgorithm en el cual establecerá la lista de Datacenters de dicha clase mediante la lista de Datacenters obtenida de la clase GeneralParametersMetaFuzzy, para que así, posteriormente, cuando se proceda a ejecutar el proceso de Meta-Planificación, dichos algoritmos dispongan de acceso al conjunto de Datacenters para obtener información del estado de estos para proceder a realizar el proceso de Meta-Planificación. A continuación se adjunta el fragmento de código de la clase WorkflowMetaScheduler donde se procede a asignar dicha lista de Schedulers a la lista schedulerList de la clase BaseSchedulingAlgorithm.

```
BaseSchedulingAlgorithm metaScheduler =
getMetaScheduler(Parameters.getMetaSchedulingAlgorithm());
metaScheduler.setCloudletList(getCloudletList());
metaScheduler.setSchedulerList(getSchedulers());
List<WorkflowDVFSDatacenter> datacenters = new
ArrayList<WorkflowDVFSDatacenter>();
datacenters.addAll(GeneralParametersMetaFuzzy.getDatacenters());
metaScheduler.setDatacenterList(datacenters);

try
{
	metaScheduler.run();
}
catch(Exception e)
{
	Log.printLine("Error in configuring meta scheduler_method");
```

```
        e.printStackTrace();
}
```

Como se observa, se procede, en primera instancia, a utilizar la clase getMetaScheduler, mediante la cual se obtiene el algoritmo de Meta-Planificación especificado en la clase estática Parameters. Posteriormente, se procede a asignar al objeto MetaScheduler de la clase BaseSchedulingAlgorithm la lista de tareas a planificar, la lista de planificadores (Schedulers), y la lista de Datacenters, obtenida de la clase estática GeneralParametersMetaFuzzy (posteriormente en la explicación de las modificaciones de CloudSim, se detalla cómo se asignan dichos Datacenters). Una vez que estos parámetros han sido establecidos sobre dicho objeto MetaScheduler, mediante el método run(), se procederá a ejecutar el algoritmo de Meta-Planificación seleccionado para realizar dicho proceso de Meta-Planificación.

El motivo por el cual serán pasados tanto las entidades Schedulers como las Datacenters al algoritmo de Meta-Planificación es debido a que son necesarios para realizar este proceso. En primera instancia, si el algoritmo precisa utilizarlo, mediante la lista de Datacenters podrá realizar el estudio de los recursos y características de estos, para así observar y decidir cuál debe realizar el procesado de una determinada tarea, realizando este proceso para todas las tareas a planificar. Una vez todas estas han sido planificadas para ser ejecutadas, procederá a, mediante la lista bindedDatacenterToScheduler presente en la clase estática GeneralParametersMetaFuzzy, a conocer para cada Datacenter su correspondiente planificador, para lo cual, una vez conocido, se enviará al respectivo Scheduler de cada Datacenter su correspondiente lista de tareas a ejecutar, para que así cada Scheduler pueda realizar la planificación de dichas tareas a procesar.

- CloudSim:

Modificación del núcleo de CloudSim, el cual es utilizado por todos los simuladores que han surgido como evolución de CloudSim, modificado para añadir el estado de los Datacenters al GeneralParameters

Uno de los problemas surgidos en el desarrollo del simulador con la entidad de Meta-Planificación consistía en la incapacidad de proporcionar acceso los Datacenters a esta entidad, debido a que los Datacenters están bajo el control de sus respectivos Planificadores. Para solucionar este problema, se procedió a, mediante la clase GeneralParametersMetaFuzzy, en la cual se definieron los parámetros anteriormente expuestos, realizar una copia de los objetos que representan a cada uno de los Datacenters, para así, poder acceder al estado de estos desde la entidad MetaScheduler.

Este proceso de obtención y copia de las entidades Datacenters debía realizarse de manera que permitiera en todo momento obtener una información actualizada del estado de todos estos, de manera que esta copia debía realizarse en cada iteración para el nuevo procesado de eventos por parte de todas las entidades. Para proceder a realizar dicha copia iterativa de las entidades Datacenter, se procedió a implementar ésta en el inicio de ejecución de la clase runClockTick del núcleo del simulador programado en la clase Java CloudSim. Esto es así debido a que este componente del simulador es el único que posee acceso completo a todas las entidades del simulador, al ser el encargado de proceder a ejecutar de manera iterativa todas y cada una de las entidades para que procesen sus eventos, de manera que, cada vez que esta clase debe realizar el inicio y ejecución iterativa de todas las entidades para que atiendan sus respectivos eventos, procederá, en primera instancia, previamente a llamar cada entidad individualmente para que realicen dicho procesamiento de eventos, a realizar dicha copia de dichas entidades Datacenter a la clase Java estática GeneralParametersMetaFuzzy. De esta manera, posteriormente, el MetaScheduler podrá realizar el acceso a dichos Datacenters para conocer su estado para así poder utilizar dicha información en el proceso de Meta-Planificación. A continuación se adjunta el fragmento de código en cuestión el cual realiza dicha función, el cual está implementado en el método runClockTick de la clase CloudSim.

```
GeneralParametersMetaFuzzy.initializeDatacenters();
List<WorkflowDVFSDatacenter> datacenters = new
ArrayList<WorkflowDVFSDatacenter>();
List<Integer> datacenterIds = new ArrayList<Integer>();
datacenterIds = GeneralParametersMetaFuzzy.getDatacenterIds();
if(datacenterIds != null)
{
      for (int i = 0; i < datacenterIds.size(); i++)
      {

      datacenters.add((WorkflowDVFSDatacenter)entities.get(datacenterIds.
get(i)));
      }
      GeneralParametersMetaFuzzy.setDatacenters(datacenters);

}
```

Tal y como se observa, se procede a acceder a todas las entidades Datacenters que se encuentran dentro del atributo lista que posee el listado de entidades llamado "entities". Además, para hacer dicha función retro compatible con el resto de simuladores basados en CloudSim, se procede a filtrar el proceso de copiado de las entidades Datacenters en caso de que no se esté utilizando el simulador con Meta-Planificador,

situación en la que se obtendría una lista vacía del proceso de obtención de los IDs de los Datacenters almacenados en GeneralParametersMetaFuzzy.

- WorkflowDVFSBasicMeta:

Por último, para la ejecución del entorno de simulación, ha sido creada una clase llamada WorkflowDVFSBasicMeta, estando ésta basada en la clase WorkflowDVFSBasicNoBrite de WorkflowSim con DVFS. Esta clase ha sido adaptada al funcionamiento del nuevo entorno de simulación, procediendo a utilizar las clases Java anteriormente citadas para la creación de las entidades del simulador. A continuación, se exponen las modificaciones realizadas respecto a WorkflowSim con DVFS.

Capacidad de establecer en la clase estática WorkflowSimMetaFuzzy la ruta al fichero que contiene la totalidad del FRBS (definición de antecedentes, consecuentes, funciones de pertenencia y base de reglas), para permitir su uso con los algoritmos de Meta-Planificación basados en esta tecnología. Posteriormente, en el próximo capítulo, se procederá a explicar los algoritmos de Meta-Planificación.

Especificación del algoritmo de Meta-Planificación a utilizar de los presentes en la lista MetaSchedulingAlgorithm de la clase estática Parameters, así como la posterior inicialización de esta para especificar parámetros tales como el algoritmo de Meta-Planificación y el algoritmo de planificación local.

Creación manual de todos y cada uno de los Datacenters a en el entorno de simulación específico mediante el nuevo método createDatacenterV2. Este nuevo método está basada en el método createDatacenter, al que se ha realizado una serie de modificaciones para permitir la creación paramétrico-manual de los Datacenters con unas características a especificar. Esto permite crear tantos Datacenters como sean necesarios mediante la llamada a este método y, mediante el paso por parámetro de las características a poseer por el Datacenter, se permite la creación de estos con diferentes características a ser empleados en el proceso de simulación. Los parámetros que pueden ser especificados en la creación de un Datacenter son el número de equipos, la tensión del procesador de dichos equipos, su frecuencia de funcionamiento, la capacitancia y su IPC. No obstante, este método ha sido implementado de manera que se procederá a crear los equipos de dicho Datacenter para que este sea heterogéneo, es decir, que cada equipo posea unas características diferentes, para que así exista heterogeneidad en los Datacenters.

Inicialización manual del atributo bindedDatacenterToScheduler del tipo Map de la clase estática GeneralParametersMetaFuzzy, estableciendo manualmente la asociación

de Datacenter y Scheduler mediante sus respectivos identificadores, permitiendo así, posteriormente, conocer esta asociación por parte del MetaScheduler.

4.3.4. Algoritmos de Meta-Planificación

En el presente capítulo se procede a explicar y detallar los diferentes algoritmos de Meta-Planificación creados e implementados para ser utilizados por la nueva entidad MetaScheduler. Se ha procedido a implementar cuatro algoritmos de Meta-Planificación, los cuales están basados en algoritmos ya implementados en el simulador WorkflowSim con DVFS, siendo los algoritmos implementados los siguientes.

- RoundRobinMetaSchedulingAlgorithm
- MinMinMetaSchedulingAlgorithm
- MaxMinMetaSchedulingAlgorithm
- PowerAwareFuzzyMaxMaxMetaSchedulingAlgoritm

A continuación, se procede a detallar el funcionamiento de cada uno de los algoritmos de Meta-Planificación implementados.

- RoundRobinMetaSchedulingAlgorithm

Basado en la clase java RoundRobinSchedulingAlgoritm implementado en WorkflowSim (3.2.3), el cual realiza la función de planificación a nivel local. Este algoritmo de planificación está basado en el algoritmo clásico de Round Robin. Originalmente, el comportamiento de este algoritmo a nivel de planificador local, consistía en el reparto de las tareas disponibles a ser planificadas por el Scheduler entre todas las VMs del Datacenter asociado, de manera que este algoritmo recorrería tarea a tarea, de manera que, para una tarea seleccionada, procedería a observar el estado de la primera VM, y, si esta se encuentra en un estado ocioso, procedería a asignar dicha tarea a dicha VM para su procesamiento. Una vez encontrado para una determinada tarea una VM libre que pueda procesar esta, el algoritmo procedería a obtener la siguiente tarea de la lista ser asignada a una VM para su procesamiento.

Para el caso del Meta-Planificador, este no dispone de una lista de VMs a las que asignar tareas, si no que dispondrá de una lista de planificadores con sus Datacenters asociados a los que asignar estas, de manera que, siguiendo el proceso de Round Robin, procederá a, para cada tarea, asignar un planificador local el cual deba planificar dicha tarea para ser procesada por una VM de su Datacenter. Para realizar este proceso, se ha procedido a utilizar la lista scheduledListMeta (explicada en el capítulo 3.3.3. correspondiente al WorkflowMetaScheduler), que posee una lista de tareas por cada planificador existente, de manera que, a dicha lista, se irán añadiendo las tareas a ser

planificadas posteriormente por los planificadores locales para ser ejecutadas en sus Datacenters asociados. A continuación, se adjunta un fragmento de código del algoritmo RoundRobinMetaSchedulingAlgorithm en el que se observa dicho proceso de asignación de tareas a las listas de tareas a planificar.

```
xS = 0;
for(int j = 0; j < size; j++)
{
	Cloudlet cloudlet = (Cloudlet) getCloudletList().get(j);
	cloudlet.setUserId(schedulerList.get(xS).getId());
	addElementSchedulerListMeta(xS, (Job)cloudlet);
	xS++;
	if (xS == schedulerSize)
	{
		xS = 0;
	}
}
```

Tal y como se observa, dicho algoritmo de reparto de tareas se basa en un bucle *for* el cual irá recorriendo la totalidad de las tareas hasta alcanzar la última tarea de la lista, delimitada por la variable "size", de manera que, mediante la variable "xS", que representará al índice del planificador de la lista de planificadores, procederá a asignar cada una de estas a un planificador. Primeramente, mediante el método setUserId, establecerá el identificador del planificador al que irá dirigida dicha tarea, de manera que, posteriormente, esta será asignada a la lista de tareas correspondiente al planificador al que va dirigido mediante el método addElementSchedulerListMeta.

A continuación, se procede a explicar los algoritmos de Meta-Planificación basados en MaxMin y MinMin, que comparten gran parte de su comportamiento, de manera que se procederá a explicar, para cada algoritmo, sus partes no comunes, las cuales constituyen para ambos su primera parte del código, y, posteriormente, se explicará dicha parte común, con objeto de facilitar la comprensión de dichos algoritmos.

- MinMinMetaSchedulingAlgorithm:

Algoritmo basado en el algoritmo MinMinSchedulingAlgorithm implementado en WorkflowSim (3.2.3). El comportamiento de este será similar al algoritmo MinMin de WorkflowSim para planificadores locales, es decir, procederá a, de la lista de tareas a Meta-Planificar, buscar la tarea más pequeña, para la cual, posteriormente, procederá a realizar la siguiente parte del algoritmo, que consiste en buscar al Datacenter idóneo el que procesar dicha tarea en el menor tiempo posible. Una vez realizado esto, volverá al primer paso, buscando nuevamente otra tarea que sea la más pequeña de la lista de tareas a planificar, realizando este proceso de manera iterativa hasta procesar todas.

- MaxMinMetaSchedulingAlgorithm:

Algoritmo basado en MaxMinSchedulingAlgorithm implementado en WorkflowSim (3.2.3). El comportamiento de este algoritmo es similar al especificado para el algoritmo MinMinMetaSchedulingAlgorithm, con la salvedad de que este algoritmo procederá a buscar iterativamente la tarea más grande en el primer paso, priorizando siempre el procesamiento de las tareas más grandes sobre las más pequeñas.

A continuación, se procede a explicar la parte común de ambos algoritmos.

En primera instancia, se debe tener en cuenta que a diferencia de los algoritmos de planificación locales, en los que los planificadores poseen acceso total a la información del Datacenter, los Meta-Planificadores no deben poseer esta capacidad, sino que deben poseer sólo información resumida. Esto implica que, un Meta-Planificador, no podrá saber los MIPS de manera independiente para cada VM del Datacenter, sino que conocerá la totalidad de los MIPS de este. De igual manera sucede para el estado de las VM, dado que no podrá saber el estado individual para cada una de estas, no sabiendo si está o no libre, aunque sí conocerá el total de VMs que están en estado ocioso en el Datacenter.

Se debe tener en cuenta que no es posible obtener de los Datacenter la información resumida, de manera que, dado que es necesario que el MetaScheduler trabaje con dicha información resumida, se ha procedido a implementar en cada algoritmo de Meta-Planificación una serie de métodos para simular dicha obtención resumida. Para llevar esto a cabo, en primera instancia, se procederá a obtener la información completa y total de los Datacenters por medio de la clase GeneralParametersMetaFuzzy, de manera que, una vez obtenida esta información completa, se procederá a crear la información resumida con la que posteriormente los algoritmos de Meta-Planificación trabajarán. Para obtener dicha información resumida, se ha creado tres métodos llamados calculateTotalIdleVms, calculateTotalVmsMips y calculateMipsPerVm, expuestos a continuación.

- calculateTotalidleVms:

```
private void calculateTotalIdleVms()
{
	int dSize = datacenterList.size();
	for(int i = 0; i < dSize; i++)
	{
		int idleVms = 0;
		List<CondorVM> vmList = datacenterList.get(i).getVmList();
		for(CondorVM vm : vmList)
		{
			if(vm.getState() == WorkflowSimTags.VM_STATUS_IDLE)
```

```
                {
                        idleVms++;
                }
            }
            TotalIdleVms.add(idleVms);
        }
}
```

Como se observa, este algoritmo recorre el listado de Datacenters creando una lista que poseerá el valor de las VMs que se encuentran libres por cada Datacenter, valor utilizado posteriormente en el proceso de Meta-Planificación de ambos algoritmos basados en MinMin y MaxMin.

- calculateTotalVmsMips:

```
private void calculateTotalVmsMips()
{
      int dSize = datacenterList.size();
      for(int i = 0; i < dSize; i++)
      {
            double vmMips = 0;
            List<CondorVM> vmList = datacenterList.get(i).getVmList();
            for(CondorVM vm : vmList)
            {
                  //It has to calculate the amount of free MIPS (VM that
are in the idle state)
                  if(vm.getState() == WorkflowSimTags.VM_STATUS_IDLE)
                  {
                        vmMips += vm.getMips();
                  }
            }
            TotalVmsMips.add(vmMips);
      }
}
```

Como se puede observar, dicho algoritmo recorre el total de VMs de cada Datacenter creando una lista que posee el valor total de MIPS por cada Datacenter, valor utilizado en el proceso de Meta-Planificación.

- calculateMipsPerVm:

Método que se encarga de una vez obtenido el total de MIPS por Datacenter, realizar una estimación de los MIPS por cada VM del Datacenter mediante el cálculo de la media de dicho valor, para así, posteriormente, proceder a utilizar este medio de MIPS en el proceso de Meta-Planificación.

Finalmente, el algoritmo en cuestión, procede a utilizar los parámetros anteriormente mostrados para el proceso final de Meta-Planificación, de manera que, una vez que ha sido escogida una tarea de la lista de tareas a ser planificada, procederá a buscar el Datacenter del listado de Datacenters el cual cuente con al menos una VM libre, y

además cuente con mayor cantidad de MIPS en los MIPS medios estimados por cada VM del Datacenter.

Por otra parte, debe ser tenido en cuenta que para poder realizar una correcta Meta-Planificación, una vez que una tarea es preparada para ser asignada al Planificador de un determinado Datacenter, será necesario, manualmente, realizar un ajuste de los recursos disponibles de este. Dicho ajuste consiste en proceder a eliminar una VM del listado que mantiene el número de VM libres, así como restar el valor medio estimado de MIPS por VM del total de MIPS del Datacenter. Este proceso es necesario para poder proseguir realizando un correcto proceso de planificación, ya que, si este proceso no es llevado a cabo, en el proceso de Meta-Planificación, siempre se percibirán los Datacenters con las mismas características, sin importar cuantas tareas hayan sido puestas en la cola de envío a dicho Datacenter, pudiendo suceder que, por ejemplo, si el Datacenter cuenta con 10 VM en estado ocioso, se proceda a enviar más de 10 tareas para ser procesadas, siendo esto no posible, ya que se recuerda, que en WorkflowSim y CloudSim, se considera que una VM solo podrá ejecutar una única tarea al mismo tiempo.

- PowerAwareFuzzyMaxMaxMetaSchedulingAlgoritm:

Último algoritmo implementado a nivel de Meta-Planificación, estando este basado en el algoritmo PowerAwareFuzzyMaxMaxSchedulingAlgorithm implementado en WorkflowSim con DVFS. La particularidad de este algoritmo es la de, como ya fue especificado en el apartado 3.2.4, es la de utilizar un FRBS para el proceso de planificación. Al igual que ocurre con los algoritmos anteriores, este ha sido adaptado para funcionar a nivel de Meta-Planificador, habiendo sido necesario realizar la implementación de métodos extra a parte de los ya especificados anteriormente para proceder a realizar la simulación de obtención de información resumida de los Datacenters.

En primera instancia, el FRBS implementado cuenta con un total de 6 antecedentes, así como de un consecuente. Los antecedentes utilizados son los mismos mencionados en el apartado 3.2.4 para la implementación del algoritmo de planificación local FRBS, con la particularidad de que se ha añadido la utilización del Datacenter para ser tenida en cuenta en el proceso de Meta-Planificación. La inclusión de este parámetro es de gran importancia, ya que permite así que se priorice la selección de aquellos Datacenters que cuenten con una mayor cantidad de recursos libres, para así evitar saturar a aquellos Datacenters que de por sí ya cuenten con un alto nivel de utilización.

Los antecedentes utilizados quedan especificados a continuación, así como los métodos que han sido implementados en el algoritmo de Meta-Planificación para realizar su uso en el FRBS.

- MIPS:

Parámetro mediante el cual se permite la priorización de selección de aquellos Datacenters que cuenten con una mayor cantidad de recursos computacionales. Dado que este parámetro es utilizado a nivel de Meta-Planificación se realiza la obtención del valor medio de MIPS por VM en cada uno de los Datacenters del entorno de simulación. Para la obtención del valor de este parámetro, se ha procedido a utilizar el método calculateMipsPerVm anteriormente citado en MaxMin y MinMin.

A modo de ejemplo, se expone a continuación la Figura 10 donde se muestran las funciones de pertenencia para el antecedente MIPS. Para el resto de antecedentes, sus funciones de pertenencia son tal y como se muestra en esta figura.

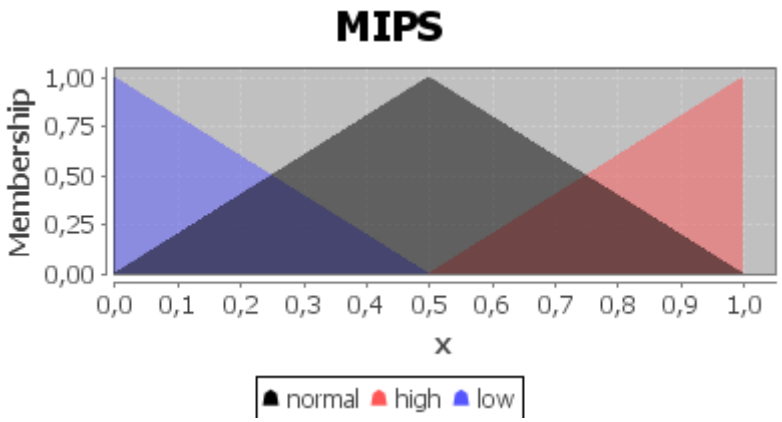

Figura 10: antecedente MIPS

- Utilización:

Valor introducido para evitar sobre saturar un Datacenter ya de por sí saturado, permitiendo priorizar la selección de aquellos Datacenters que, aun no contando con tanta potencia y/o realizando un mayor nivel de consumo, cuenten con un alto nivel de disponibilidad, por lo que deba priorizarse la planificación de tareas a estos, en lugar de a los que ya están saturados. Para la obtención del nivel de utilización se ha implementado un método llamado calculateDatacenterUsage, que permite la obtención del nivel de utilización de todos los Datacenters mediante la creación de una lista que almacenará dicho nivel de utilización de los Datacenters del entorno de simulación. El valor de

utilización es calculado de tal forma que se obtendrá un valor de 0 para una utilización del 0% y de 1 para una utilización del 100% del Datacenter.

- Potencia:

Dado que este valor debe ser utilizado en el ámbito de Meta-Planificación, se ha procedido a crear un método llamado calculateMeanDatacenterPower mediante que se procede a obtener la media de consumo de potencia de todas las VM que constituyen el Datacenter, para utilizar este valor como una estimación de consumo realizado por cada Datacenter ante la ejecución de una tarea. Este valor es utilizado para la obtención del consumo de energía del Datacenter, ambos valores los cuales permiten enfocar la optimización de la Meta-Planificación orientada a una disminución de los consumos.

- Longitud:

Valor obtenido directamente de la tarea a planificar. Este valor permite enfocar el algoritmo de planificación para priorizar la selección de una tarea en función de su longitud para ser procesada, permitiendo, por ejemplo, establecer un mayor nivel de selección de las tareas más grandes en aquellos Datacenters que dispongan de mayor cantidad de recursos, pudiendo ser estos tanto en MIPS como en recursos libres (utilización).

- Tiempo de ejecución:

Valor calculado mediante la división de la longitud de la tarea a procesar y el valor medio de MIPS estimado por Datacenter. La utilidad de este parámetro reside en cómo se enfocará la planificación de tareas en el FRBS, permitiendo enfocar la planificación de estas priorizando el menor tiempo de ejecución sin tener en cuenta otros parámetros, teniendo en cuenta otros parámetros, o bien no considerando este parámetro como prioritario, pudiendo enfocar la planificación a la optimización de otros parámetros. Este parámetro es obtenido como proceso previo a obtener el nivel de selección de una tarea en un determinado Datacenter.

- Energía:

Este parámetro queda expresado en función del tiempo de ejecución y de la potencia media estimada consumida por dicho Datacenter para su procesamiento, de manera que no ha sido necesario realizar la implementación de un nuevo método para su cálculo, ya que este parámetro será calculado a través del producto del tiempo de ejecución de la tarea por dicho equipo y de la potencia media consumida por dicho Datacenter, que han

sido calculados con anterioridad. El valor de este parámetro es calculado como proceso previo a obtener el nivel de selección de una tarea en un determinado Datacenter.

Por otra parte, el consecuente para este FRBS implementado consiste en el nivel de selección que se le otorgará a un determinado Datacenter en cumplimiento con las reglas de la base de datos.

Para cada uno de los antecedentes implementados se han implementado tres funciones de pertenencia, siendo estas bajo, medio y alto, para establecer en función del valor de entrada, el grado de cumplimiento con una determinada regla, mientras que para el consecuente se ha procedido a implementar un total de 5 funciones de pertenencias, siendo estas muy bajo, bajo, normal, alto y muy alto, para así disponer de más resolución y precisión a la hora de realizar la toma de decisiones.

Además, como fue indicado en el apartado 3.1.1, en los sistemas FRBS se debe trabajar con los antecedentes y consecuentes normalizados, es decir, con valores comprendidos, para este caso, entre 0 y 1, de manera que se ha procedido a utilizar un método llamado normaliseAntecedents que procede a normalizar los valores anteriormente especificados.

Para la normalización de los antecedentes, se utiliza unos atributos creados en la clase estática GeneralParametersMetaFuzzy que realizan la función de almacenar los valores máximos y mínimos de cada uno de estos, valores utilizados para establecer el rango en el cual variará cada antecedente, siendo representando el valor 0 por el valor mínimo almacenado correspondiente a cada uno de los antecedentes, así como el 1, el cual es representado por el valor más alto almacenado. Así, los valores utilizados para almacenar dichos valores son los siguientes.

- maxDatacenterMips
- minDatacenterMips
- maxPower
- minPower
- maxDaxLength
- minDaxLength
- maxTime
- minTime
- maxEnergy
- minEnergy

Como ha sido mencionado en el apartado 3.3.3 en referencia a la modificación de la clase WorkflowParser, los valores máximos y mínimos para la longitud de tareas son establecidos durante el proceso de conversión de las tareas almacenadas en el fichero DAX al listado de trabajos comprendido por el simulador.

El resto de valores máximos y mínimos son obtenidos en la primera ejecución del algoritmo de Meta-Planificación PowerAwareFuzzyMaxMaxMetaSchedulingAlgoritm, donde se procede a obtener y establecer en la clase estática GeneralParametersMetaFuzzy los valores máximos y mínimos para los MIPS y la potencia por medio de los métodos calculateMips y calculatePower que han sido creados para dicho propósito. Por otro lado, los valores máximos y mínimos de tiempos de ejecución y energía son obtenidos a través de los valores anteriormente obtenidos y los valores de longitudes máximas y mínimas obtenidas mediante la entidad WorkflowParser, obteniendo estos valores tal y como fue expresado en el apartado 3.3.3 en la descripción de la clase GeneralParametersMetaFuzzy.

5. RESULTADOS Y DISCUSIÓN

En el presente capítulo se procede a analizar los resultados obtenidos de las diferentes estrategias de Meta-Planificación implementadas en WorkflowSim con DVFS. Para la obtención de los resultados, se ha creado un escenario de simulación con unas características específicas, donde se han utilizado los algoritmos de Meta-Planificación creados en el desarrollo del nuevo simulador, de donde se han obtenido unos resultados de tiempos de ejecución y consumos que han sido analizados y comparados.

5.1. Escenario de simulación

En primera instancia, para poder realizar el análisis del simulador y los algoritmos de Meta-Planificación, ha sido necesario crear un escenario de simulación sobre el cual obtener los resultados a analizar.

El escenario de simulación planteado cuenta con un total de tres Datacenters, cada uno con su respectivo planificador local, teniendo éstos características diferentes entre sí, estando constituidos todos estos por un total de 20 máquinas físicas (hosts). En el escenario planteado, el primer Datacenter ha sido implementado enfocado en representar un Datacenter antiguo reutilizado, que cuenta con unas prestaciones reducidas y un alto nivel de consumo. El segundo Datacenter cuenta con unas características tales que se asemejen a un Datacenter moderno de grandes prestaciones, por lo que este poseerá un alto nivel de MIPS, al igual que un nivel de consumo elevado. Por último, el tercer

Datacenter ha sido diseñado con el objetivo de simular un Datacenter moderno de bajas prestaciones pero de muy bajo consumo. Por otra parte, cada uno de estos Datacenters han sido implementados para que cada uno de los host presentes contará con un único procesador de un único núcleo, además de que se ha creado una única VM por host. Gracias a la implementación de estos tres Datacenters, se tiene un escenario variado sobre el que realizar el estudio y comportamiento de los algoritmos de Meta-Planificación y poder observar su funcionamiento, pudiendo comparar los resultados obtenidos.

A continuación, se adjuntan unas tablas en las que se puede observar las características de cada Datacenter basadas en sus respectivos hosts.

Primer Datacenter:

Host	**Frecuencia máxima (GHz)**	**Tensión máxima (V)**	**Capacitancia (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

Tabla 6: Características del primer datacenter

Segundo Datacenter:

Host	**Frecuencia máxima (GHz)**	**Tensión máxima (V)**	**Capacitancia (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

Tabla 7: Características del segundo datacenter

Tercer Datacenter:

Host	**Frecuencia máxima (GHz)**	**Tensión máxima (V)**	**Capacitancia (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

Tabla 8: Características del tercer datacenter

Como se puede observar, con el presente entorno de simulación, se posee un total de 60 hosts, en los que se ha creado una única VM por host, haciendo un total de 60 VM. Dado que el funcionamiento del simulador permite el procesamiento de una única tarea por VM, se dispone de la capacidad de realizar un total de 60 ejecuciones de tareas simultáneamente.

Como se utiliza DVFS, ha sido necesario establecer un gobernador para el control de frecuencias y tensiones, habiendo utilizado OnDemand, que permite un ajuste óptimo de las frecuencias de los procesadores según su carga de trabajo.

Con referencia al escenario de tareas a ejecutar por el entorno de simulación, se ha elegido el DAX Montage_100, que cuenta con un total de 100 tareas, siendo estas suficientes para comprobar la calidad de los algoritmos de planificación.

Por último, a nivel de planificación, se debe de diferenciar dos escenarios, planificación local, y Meta-Planificación. Dado que el objetivo de este TFG es el de realizar una optimización máxima referente a consumos, se debía proceder a realizar la elección de un algoritmo de planificación local tal que otorgara un resultado óptimo en el proceso de planificación local referente a consumos, de manera que se ha procedido a utilizar el algoritmo PowerAwareMaxMinSchedulingAlgorithm especificado en el capítulo 3.2.4. Por otra parte, dado que el otro objetivo de este TFG es la implementación y estudio de los algoritmos de Meta-Planificación, se ha obtenido los resultados de simulación para los cuatro algoritmos de Meta-Planificación implementados en este simulador especificados en el capítulo 3.3.4.

5.1.1. Base de reglas para el escenario de simulación

En referencia a la preparación del escenario de simulación, el algoritmo de Meta-Planificación PowerAwareFuzzyMaxMaxMetaSchedulingAlgorithm está basado en un sistema FRBS, lo que implica que, para dicho algoritmo, con el objetivo de obtener un rendimiento óptimo en nuestro escenario de simulación, será necesario adaptar este.

El proceso de adaptación de dicho algoritmo para nuestro escenario pasa por la creación de una base de reglas específica y concreta. Dicha base de reglas creada cuenta con un total de 84 reglas, habiendo sido estas enfocadas con el objetivo de optimizar consumos. Para esto, el conjunto de reglas han sido escritas priorizando el antecedente energía, por lo que, de manera generalizada, siempre y cuando un Datacenter concreto posea como resultado de la ejecución de una tarea un nivel de energía bajo, este será priorizado, siempre y cuando, este no se encuentre en un estado de saturación, es decir, siempre que su nivel de utilización no sea alto.

Para la obtención de esta base de reglas, se ha procedido a crearla manualmente, es decir, no se ha utilizado ningún sistema de aprendizaje de reglas para obtenerla, sino que se ha realizado el estudio del escenario de simulación, observando las características de los Datacenters y las tareas, creando dichas reglas para que estén adaptadas al escenario de simulación utilizado.

Dado que se cuenta con tres Datacenters y se tiene como función objetivo optimizar el consumo sin castigar a los Datacenters saturados con más tareas a menos que sea necesario, para el proceso de desarrollo de dicha base de reglas se ha procedido a dividir ésta en tres grupos, los que a su vez, han sido divididos en otros tres subgrupos. Estos grupos de reglas han sido desarrollados teniendo en cuenta las características de cada Datacenter utilizado en el escenario de simulación, es decir, dado que se tiene tres Datacenters con características diferentes, se ha procedido a crear un grupo de reglas para cada Datacenter, de manera que estas sean acordes a estos. Posteriormente, para proseguir con el proceso de organización y creación de dicha base de reglas, se ha dividido dichos grupos de reglas de cada Datacenter en función de su nivel de utilización, de esta forma, ha sido posible definir los conjuntos de reglas acordes para cada Datacenter en función de su nivel de utilización con la función objetivo de optimizar consumos.

A continuación, con el objetivo de facilitar la comprensión de la base de reglas implementada para el escenario utilizado, se muestra y detalla un total de 18 reglas,

siendo estas 6 reglas por cada Datacenter, estando divididas en dos reglas por cada nivel de utilización.

- Primer Datacenter:

En primera instancia, se muestran 6 reglas divididas en tres grupos de dos basados en su nivel de utilización para el primer Datacenter. Se recuerda, que este consiste en un Datacenter que cuenta con un nivel de prestaciones bajo, realizando un consumo elevado. Debido a estas características de bajo rendimiento y alto consumo, todas las reglas definidas para este Datacenter, tal y como se puede observar a continuación, poseerán para ambos antecedentes MIPS y power, que representan los niveles de prestaciones y consumos, estarán establecidos a un valor bajo (*low*) para MIPS y alto (*high*) para power.

- Nivel de utilización bajo:

1. `IF MIPS IS low AND utilization IS low AND power IS high AND length IS low AND time IS low AND energy IS low THEN selection IS high`
2. `IF MIPS IS low AND utilization IS low AND power IS high AND length IS normal AND time IS normal AND energy IS normal THEN selection IS normal`

Tal y como se observa, para estas dos reglas seleccionadas de las establecidas para el nivel de utilización bajo, se aprecia que, en el caso de la primera, para una condición en la cual la longitud de una tarea sea baja, y conlleve un tiempo de ejecución bajo, si a pesar del alto consumo del Datacenter, debido al bajo tiempo de ejecución de la tarea, el consumo de energía resultante es bajo, se otorgará una tasa de selección alto (*high*), la cual no es máxima, ya que la tasa de selección máxima es muy alto (*very_high*), debido a que se debe priorizar la selección de aquellos Datacenters los cuales cuenten con mayores recursos computacionales, ya que previsiblemente ejecutarán las tareas en menor tiempo que aquellos de bajas características, implicando un muy probable bajo consumo de energía al requerir poco tiempo para su ejecución. Por otra parte, para el caso de una tarea de longitud normal que requiera un tiempo de ejecución normal y suponga un consumo de energía normal, se otorgará, a pesar de su baja utilización, una tasa de selección normal, ya que interesa que otro Datacenter en las mismas condiciones de nivel de utilización y que cuenta con más prestaciones pueda posicionarse con un mayor nivel de preferencia para su selección.

- Nivel de utilización medio:

1. `IF MIPS IS low AND utilization IS normal AND power IS high AND length IS low AND time IS low AND energy IS low THEN selection IS normal;`
2. `IF MIPS IS low AND utilization IS normal AND power IS high AND length IS normal AND time IS normal AND energy IS normal THEN selection IS low;`

Como se puede apreciar, se han seleccionado las dos mismas reglas seleccionadas anteriormente, cambiando sólo el nivel de utilización del Datacenter, para lo que se observa que para la primera regla que antes poseía una tasa de selección alta, ha sido bajada a normal, así como la segunda regla, que ha sido bajada su tasa de selección a baja. El objetivo de realizar esta planificación a nivel de reglas para su selección, es el de evitar la selección de un Datacenter de prestaciones limitadas, que posee un nivel de utilización intermedio, el cual podría entrar fácilmente en un estado de utilización alto próximo a saturación con la asignación de unas pocas tareas más, por lo que se trata de priorizar la selección de un Datacenter que en las mismas condiciones de utilización cuente con unas mayores prestaciones.

- Nivel de utilización alto:

1. `IF MIPS IS low AND utilization IS high AND power IS high AND length IS low AND time IS low AND energy IS low THEN selection IS low;`
2. `IF MIPS IS low AND utilization IS high AND power IS high AND length IS normal AND time IS normal AND energy IS normal THEN selection IS very_low;`

Seleccionadas una vez más las dos mismas reglas cambiando sólo el nivel de utilización, se observa cómo, una vez más, se ha bajado la tasa de selección de estas reglas, relegando su tasa de selección a baja y muy baja, para, de igual manera ha sido explicado anteriormente, priorizar el uso de un Datacenter de mayores prestaciones en la misma condición de utilización.

- Segundo Datacenter:

Para este segundo Datacenter, se realizará el mismo procedimiento mostrando un total de 6 reglas divididas según el nivel de utilización. Tal y como ha sido descrito en el capítulo 5.1, este corresponde a un Datacenter moderno de altas prestaciones y alto consumo, de manera que, como se podrá mostrar a continuación, todas las reglas definidas para este contarán con un valor alto (*high*) establecido para los antecedentes MIPS y power, que representan prestaciones y consumos.

- Nivel de utilización bajo:

1. `IF MIPS IS high AND utilization IS low AND power IS high AND length IS low AND time IS low AND energy IS low THEN selection IS very_high;`
2. `IF MIPS IS high AND utilization IS low AND power IS high AND length IS high AND time IS normal AND energy IS normal THEN selection IS very_high;`

Como se observa con estas dos reglas establecidas para el nivel de utilización bajo, se tiene que, para el caso de la primera, ante una longitud de pequeño tamaño, que

implique que el tiempo de ejecución es bajo, si la energía consumida para realizar dicho procesamiento de la tarea es bajo, la tasa de selección será máxima, representada por el valor muy alto, mientras que, en el caso de la otra regla, para una tarea de gran longitud, que requiera un tiempo de ejecución normal en el presente Datacenter y un consumo normal de energía, se seguirá estableciendo un nivel de selección máximo, ya que, previsiblemente, en los otros dos Datacenters de bajas características, el procesado de una tarea de gran longitud, incluso para el caso del Datacenter de bajo consumo, supondrá una gran cantidad de tiempo para procesar esta, de manera que, dado que la energía queda expresada mediante el producto de la potencia por el tiempo, provocará un consumo de energía alto.

- Nivel de utilización medio:

1. `IF MIPS IS high AND utilization IS normal AND power IS high AND length IS low AND time IS low AND energy IS low THEN selection IS very_high;`
2. `IF MIPS IS high AND utilization IS normal AND power IS high AND length IS high AND time IS normal AND energy IS normal THEN selection IS high;`

En referencia al conjunto de reglas expuesto para el nivel de utilización medio, se baja el nivel de selección del Datacenter en el caso de tareas de gran tamaño que requieran un tiempo de ejecución normal y un consumo de energía normal, puesto que no interesa que ya no resulta de un alto interés que un Datacenter que posee un nivel de utilización medio procese tareas que le pueda llevar un tiempo medio en su ejecución dado que podría provocar que este entrara en estado de saturación si más tareas son enviadas a este. Por otra parte, se observa que si una tarea es de tamaño pequeño y el consumo de energía para su procesamiento es bajo, sigue resultando de interés que esta sea procesada por dicho Datacenter, ya que tanto el tiempo de ejecución como el consumo de energía será bajo, por lo tanto, se le otorga una tasa de selección muy alta.

- Nivel de utilización alto:

1. `IF MIPS IS high AND utilization IS high AND power IS high AND length IS low AND time IS low AND energy IS low THEN selection IS high;`
2. `IF MIPS IS high AND utilization IS high AND power IS high AND length IS high AND time IS normal AND energy IS normal THEN selection IS normal;`

Por último, se observa que para el nivel de utilización alto, se procede, una vez más, a bajar la tasa de selección de este Datacenter, estableciendo un nivel de selección alto para las tareas pequeñas que requieran de poco tiempo de ejecución y poco consumo de energía y una tasa de selección normal para el caso de una tarea de gran tamaño la cual requiera un tiempo de ejecución y consumo de energía normal. Tal y como se observa, a

pesar del alto nivel de utilización del Datacenter, se sigue priorizando con niveles normales y altos el uso de este, lo que se debe a las altas prestaciones de este Datacenter, el cual sigue siendo idóneo para el procesado de tareas, ya que debido a dichas altas prestaciones, conllevará que el tiempo requerido para ejecutar tareas sea mucho más bajo que el del resto de Datacenters, provocando así un menor nivel de consumo.

- Tercer Datacenter:

Para el tercer y último Datacenter, se realiza el mismo proceso mostrado anteriormente, analizando 6 reglas divididas en 3 grupos de dos reglas en función de su nivel de utilización. Tal y como fue citado en el capítulo 5.1, mediante este Datacenter se simula uno moderno de muy bajo consumo, que está, precisamente, orientado al bajo consumo, por lo que poseerá unas bajas características, de modo que, para las reglas mostradas y el resto de reglas implementadas para este Datacenter, se podrá observar que tanto para el antecedente de MIPS como de power se ha establecido un valor bajo (*low*).

- Nivel de utilización bajo:

1. `IF MIPS IS low AND utilization IS low AND power IS low AND length IS low AND time IS low AND energy IS low THEN selection IS high;`
2. `IF MIPS IS low AND utilization IS low AND power IS low AND length IS normal AND time IS normal AND energy IS normal THEN selection IS normal;`

Para este tercer Datacenter, tal y como se puede observar, se han establecido las mismas reglas establecidas en el primer Datacenter, variando sólo el nivel de consumo de potencia a bajo, pero estableciendo, en las mismas circunstancias, distinta tasa de selección. Esto se debe a que, a pesar de poseer un nivel bajo de consumo, ya que posee un nivel bajo de prestaciones, el procesado de tareas de media y gran longitud requerirán de un tiempo relativamente alto o muy alto para su procesamiento, por lo cual tenderá a realizar un consumo más alto que el realizado por un Datacenter de altas prestaciones como el establecido en el segundo Datacenter, por lo que, para este caso, incluso en el nivel de utilización bajo se partirá de unas tasas de selección altas y normales, para las dos reglas mostradas.

- Nivel de utilización medio:

1. `IF MIPS IS low AND utilization IS normal AND power IS low AND length IS low AND time IS low AND energy IS low THEN selection IS normal;`
2. `IF MIPS IS low AND utilization IS normal AND power IS low AND length IS normal AND time IS normal AND energy IS normal THEN selection IS low;`

Para la situación del nivel de utilización medio, se han seleccionado las dos mismas reglas anteriormente seleccionadas, variando solo el nivel de utilización, de manera que, como se puede mostrar, debido a que el nivel de utilización se ve incrementado, se procede a penalizar dicho aumento en el uso del Datacenter para su selección, dando oportunidad a que para una tarea de estas características, un Datacenter de mayores prestaciones disponga de mayor prioridad, ya que previsiblemente realizará un menor consumo de energía al requerir un menor tiempo para su ejecución.

- Nivel de utilización alto:

1. `IF MIPS IS low AND utilization IS high AND power IS low AND length IS low AND time IS low AND energy IS low THEN selection IS low;`
2. `IF MIPS IS low AND utilization IS high AND power IS low AND length IS normal AND time IS normal AND energy IS normal THEN selection IS very_low;`

Por último, se observa que, nuevamente, seleccionadas las dos mismas reglas que en los dos casos anteriores, para el caso del nivel de utilización alto en el Datacenter, se procede a, una vez más, reducir su tasa de selección, para así priorizar la selección de aquellos Datacenter que posean un nivel de prestaciones mayor.

5.2. Resultados

En este último capítulo se muestran los resultados obtenidos de la simulación en el escenario utilizado anteriormente definido. Se recuerda que, para la obtención de los resultados se ha procedido a utilizar a nivel de planificación local el algoritmo PowerAwareMaxMin, mientras que a nivel de Meta-Planificación se ha procedido a utilizar los siguientes algoritmos:

- RoundRobinMetaSchedulingAlgorithm
- MinMinMetaSchedulingAlgorithm
- MaxMinMetaSchedulingAlgorithm
- PowerAwareFuzzyMaxMaxMetaSchedulingAlgoritm

Como se puede observar a continuación en la Tabla 9, los peores resultados han sido obtenidos para todos los escenarios mediante el algoritmo de Meta-Planificación basado en Round Robin, de manera que dicho algoritmo ha sido utilizado como algoritmo base, especificando el nivel de mejora y ahorro en consumos porcentualmente en referencia a dicho algoritmo de Meta-Planificación. El motivo por el que este algoritmo basado en Round Robin arroja los peores resultados se debe a que este no realiza ningún proceso de planificación, sino que realizará un reparto equitativo de todas las tareas que debe

planificar el Meta-Planificador entre todos los Datacenters disponibles en el escenario de simulación.

Algoritmo	Resultados				Ahorro (%)			
	Tiempo (s)	Potencia total (W)	Potencia media (W)	Energía (Wh)	Tiempo	Potencia total	Potencia media	Energía
RoundRobin	134.88	274119,62	6,77	76,14	-	-	-	-
MinMin	76,47	141819,12	6,18	39,39	43.31	48.26	8.71	48.27
MaxMin	75,81	140935,51	6,19	39,15	43.79	48.56	8.57	48.58
Fuzzy	73,16	114548,71	5,22	31,82	45.76	58.21	22.89	58.21

Tabla 9: Resumen de resultados

Observando los resultados expuestos en la Tabla 9, se observa que el algoritmo basado en Round Robin arroja los peores resultados, mientras que los algoritmos basados en MinMin, MaxMin y Fuzzy consiguen mejorar los resultados obtenidos mediante el algoritmo basado en Round Robin para todos los parámetros medidos.

Con el objetivo de poder facilitar el estudio y comprensión de los resultados, se adjuntan una serie de figuras, en las que se muestran, para las figuras Figura 11, Figura 12 y Figura 13, los resultados obtenidos para cada uno de los parámetros medidos, mientras que en la Figura 14 se muestran los niveles de ahorro y mejora obtenidos por los algoritmos MinMin, MaxMin y Fuzzy respecto a Round Robin. El motivo por el que se han separado los valores de los resultados temporales y de potencia en tres gráficas es debido a la gran diferencia en el rango de valores obtenido de dichos resultados, lo que imposibilitaría la percepción del resto de resultados tras la inclusión de la potencia total. De igual manera que, aun excluyendo este, no se podrían apreciar los datos de consumos medios de potencia, debido a la misma razón expuesta.

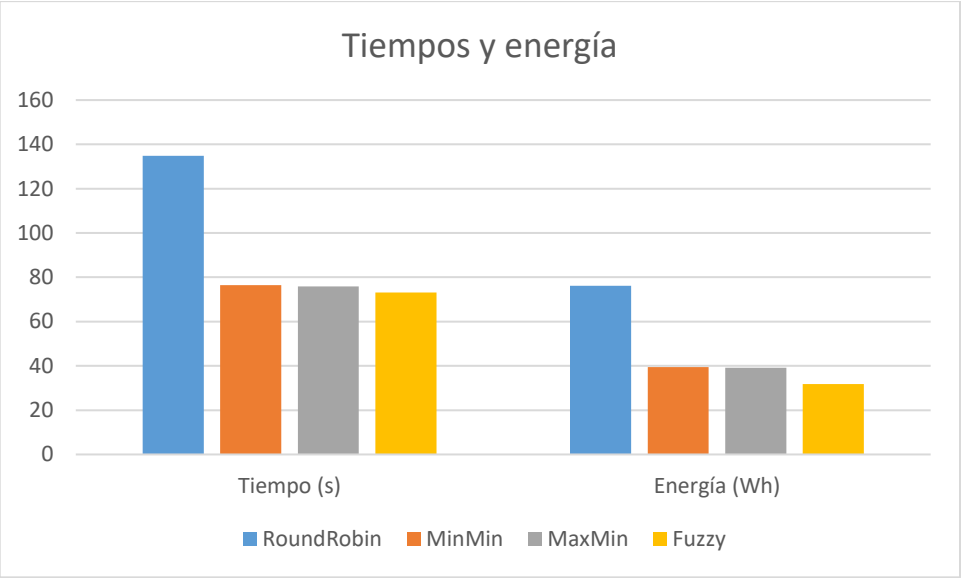

Figura 11: Tiempos y energía

Observando los datos expuestos en la Figura 11, se aprecia como existe un gran nivel de mejora entre el algoritmo de Meta-Planificación basado en Round Robin y el resto de algoritmos, mostrando y apreciando como este primero otorga unos resultados mejorables en cuanto al proceso de planificación, ya que el resto de algoritmos consiguen completar la ejecución de la simulación en unos tiempos considerablemente reducidos, así como una sustanciosa mejora de los consumos energéticos.

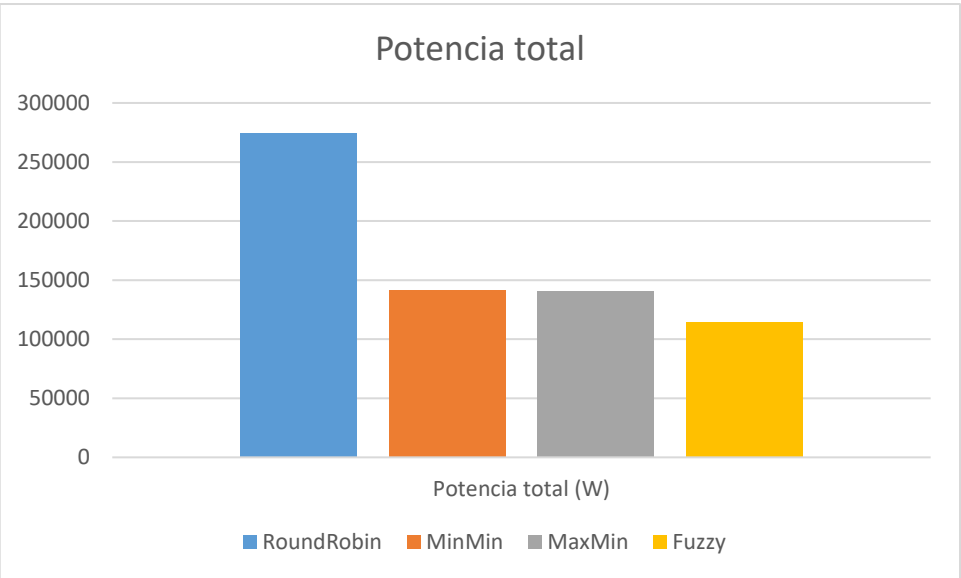

Figura 12: Potencia total

Observando la Figura 12 sobre consumo total de potencia, se observa nuevamente como los algoritmos implementados MinMin, MaxMin y Fuzzy otorgan unas mejoras

significativas respecto al algoritmo básico basado en Round Robin, observando unas mejoras aproximadas del 50%.

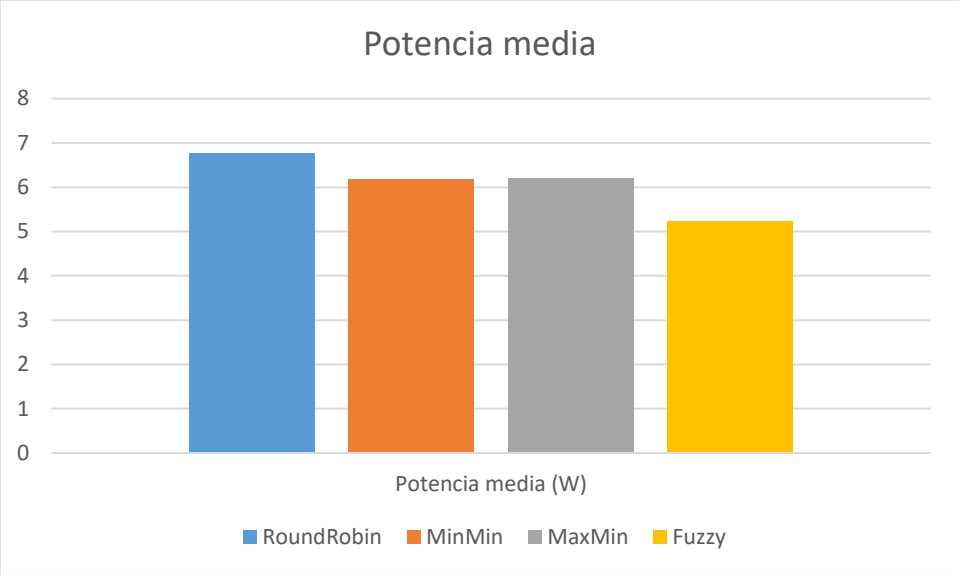

Figura 13: Potencia media

En referencia a la potencia media consumida, tal y como se observa en la Figura 13, igualmente se aprecia mejoras en este, a pesar de que estas mejoras no son tan significativas como las mostradas para los casos anteriormente expuestos. Esta modesta mejora en comparación con los parámetros anteriores obtenidos es debido a que se debe a un valor medio, el cual es obtenido por medio de la siguiente ecuación (14).

$$P_{media} = \frac{P_{Total}}{T*N} \quad (7)$$

Siendo *T* el tiempo total de ejecución, y *N* el número de Datacenter del escenario de simulación. Dado que los parámetros de tiempo total de ejecución y consumo total son directamente proporcionales, ocasionará que el valor de la potencia media sea similar entre los diferentes algoritmos de Meta-Planificación expuestos, propiciando dicha leve mejora entre estos.

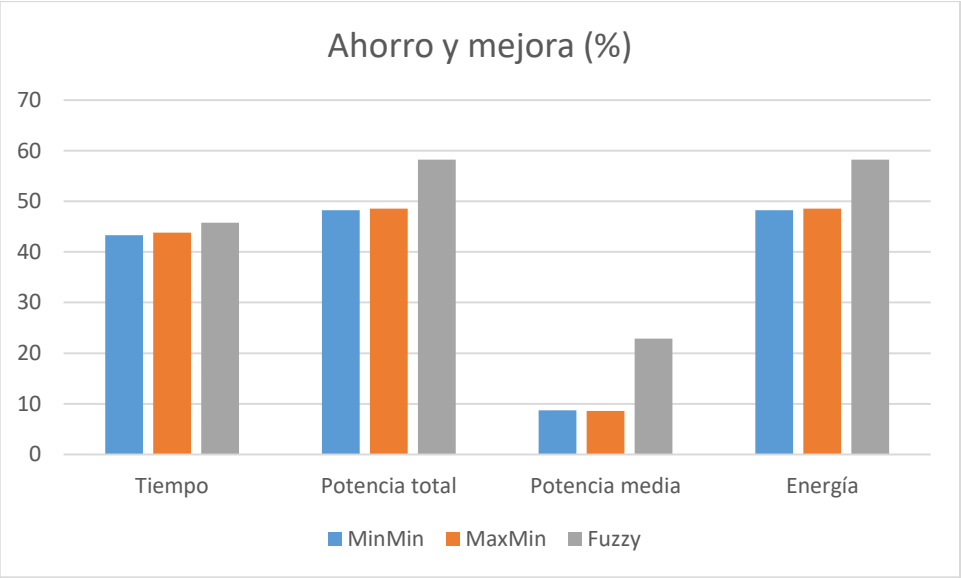

Figura 14: Ahorro y mejora de la Meta-Planificación respecto a Round Robin

Finalmente, para proporcionar una mejor apreciación de las mejoras conseguidas en todos los algoritmos expuestos respecto a Round Robin, se adjunta la Figura 14 en la que se pueden apreciar las mejoras porcentuales logradas con cada uno de los diferentes algoritmos de Meta-Planificación implementados. Tal y como se puede observar, los niveles de mejora obtenidos para todos los algoritmos son significativos, especialmente, en el parámetro más importante, el cual es el consumo de energía, ya que este está relacionado con el consumo total y el tiempo de ejecución, de manera que unas mejoras cercanas al 50% para los algoritmos MinMin y MaxMin, y unas mejoras próximas al 60% para el algoritmo basado en FRBS suponen unos muy importantes resultados. Además, se observa que, de todos los algoritmos implementados, el algoritmo con el que se obtienen los mejores resultados es mediante el algoritmo basado en FRBS, que consigue unas mejoras porcentuales de próximas al 10% respecto a los algoritmos MinMin y MaxMin, demostrando así la existencia de margen de mejora para los algoritmos actuales de Meta-Planificación, permitiendo obtener mejores resultados a los obtenidos mediante estos por medio de sistemas FRBS.

6. CONCLUSIONES

En este último capítulo, se procede a exponer las conclusiones obtenidos por medio del desarrollo de este TFG, las cuales son expuestas a continuación.

En primera instancia, se ha cumplido de manera satisfactoria el objetivo de desarrollar un simulador que permite la utilización de múltiples *datacenters* de manera centralizada.

Esto permite una mejor utilización de los recursos al disponer de un elemento de Meta-Planificación que gestionará dichos recursos de manera centralizada, recursos que, por otra parte, de no ser implementados bajo un Meta-Planificador, no podrían ser utilizados con el mismo nivel de eficiencia, ya que los usuarios que requirieran de la ejecución de tareas necesitarían realizar la selección de un *datacenter* para el procesado de tareas, lo que podría inducir a una selección de *datacenter* incorrecta por parte del usuario, y podría suponer unos tiempos de ejecución y consumo elevados para el procesamiento de tareas, debido a, por ejemplo, un alto nivel de utilización del *datacenter*, o unas bajas prestaciones, situación la cual se consigue evitar gracias a la implementación de un Meta-Planificador, el cual dispone de conocimiento suficiente sobre todos los *datacenters* gestionados por este, de manera que podrá realizar un correcto uso de los recursos gracias a una correcta planificación de las tareas a ejecutar, permitiendo así a los usuarios realizar el envío de las tareas a ejecutar al Meta-Planificador, el cual garantizará una ejecución óptima de estos y evitando que los usuarios tengan que disponer de conocimiento sobre los *datacenters*.

Además, se aprecia cómo la implementación de un correcto algoritmo de Meta-Planificación puede conseguir una mejora sustancial en la gestión y planificación de los recursos y tareas por parte del Meta-Planificador, consiguiendo unos niveles de mejoría considerables respecto a algoritmos de Meta-Planificación más básicos. Así mismo, se observan las virtudes de los algoritmos basados en FRBS, ya que estos disponen de la capacidad de poder trabajar con varias fuentes de información o antecedentes para realizar la planificación de tareas, permitiendo así obtener unos resultados mucho más precisos y eficientes para el entorno específico para el que ha sido implementado. Además, tal y como se ha mostrado en el capítulo de resultados, se ha alcanzado de manera satisfactoria el objetivo de optimización de consumos, mostrando como este algoritmo FRBS consigue unos niveles de ahorro aproximados del 10% frente a otros algoritmos de Meta-Planificación basados en estrategias clásicas.

Finalmente, se debe tener en cuenta que los resultados han sido obtenidos en un entorno de simulación, lo que puede suponer la existencia de un relativo margen de error por el cual las mejoras finales obtenidas no sean tan significativas como las mostradas. No obstante, el simulador implementado es una potente herramienta de investigación para el campo de la Meta-Planificación, ya que permite disponer de un simulador muy flexible que permite la simulación de todo tipo de escenarios, permitiendo incluso, mediante los modelos de potencia, realizar la simulación basada en procesadores reales, lo que puede permitir un gran ahorro del costo económico para la investigación de estos algoritmos, ya que evita la necesidad de realizar la instalación de una infraestructura real

de *Cloud computing*. Por otra parte, gracias a este entorno simulado, el tiempo requerido para la ejecución de un escenario y obtención de los resultados se verá considerablemente reducido en comparación al requerido en un entorno real, ya que este sí deberá realizar el procesado completo real de las tareas, en lugar de la estimación realizada por el simulador, permitiendo un rápido estudio de diferentes escenarios y algoritmos.

7. LÍNEAS FUTURAS

Durante el desarrollo del presente TFG, se ha detectado una serie de líneas futuras de desarrollo mediante las cuales proporcionar una mejora al simulador desarrollado.

Desarrollo de nuevos algoritmos de Meta-Planificación FRBS, aumentando la complejidad de estos. Se propone la inclusión de nuevos antecedentes mediante los cuales medir el tiempo total de utilización, la longitud total de las tareas ejecutadas, y el total de tareas ejecutadas. Esto permitirá una optimización en la cual se tenga en cuenta valores históricos del estado de un *datacenter* para el proceso de Meta-Planificación.

Implementación de sistemas de aprendizaje de FRBS. De esta manera, se conseguiría una fácil y automatizada adaptación optimizada de las bases de reglas de los algoritmos FRBS a todo tipo de escenarios. Para dicho sistema de aprendizaje, se propone el empleo de *Pittsburgh Approach* [20], KASIA [4], Michigan [5] y KARP [20].

8. REFERENCIAS BIBLIOGRÁFICAS

[1] Brian Hayes, BT OGRAPH, YR Morgens – Cloud Computing - Communications of the ACM, 2008

[2] Oscar Cordón. Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases

[3] jFuzzyLogic - http://jfuzzylogic.sourceforge.net/html/manual.html

[4] R.P. Prado, Sebastián García Galán, J. E. Muñoz Expósito, A.J. Yuste Delgado Knowledge Acquisition in Fuzzy Rule Based Systems with Particle Swarm Optimization IEEE-TRANSACTIONS ON FUZZY SYSTEMS. Volume 18, Issue 6, December 2010, Pages 1083-1097

[5] J. Holland (1975). Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor.

[6] Kennedy, J., Eberhart, R. C., and Shi, Y., Swarm intelligence San Francisco: Morgan Kaufmann Publishers, 2001.

[7] Rong Ge, Xizhou Feng, Kirk W. Cameron .Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1559986

[8] J. Li and J. F. Martíınez,"Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiproces-sors,"

[9] C. Piguet, C. Schuster, and J.-L. Nagel, "Optimizing Archi-tecture Activity and Logic Depth for Static and Dynamic Power Reduction," The 2nd Annual IEEE Northeast Workshop on Circuits and Systems, pp. 41-44, June 2004.

[10] V Pallipadi, A Starikovskiy - Proceedings of the Linux Symposium, 2006 – The ondemand governor http://mathdesc.fr/documents/kernel/linuxsymposium_procv2.pdf#page=223

[11] Virtual Machine Migration https://pubs.vmware.com/vsphere-50/index.jsp?topic=%2Fcom.vmware.wssdk.pg.doc_50%2FPG_Ch11_VM_Manage.13.2.html

[12] VMware http://www.vmware.com/

[13] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms

[14] Rodrigo N. Calheiros, Rajiv Ranjan, César A. F. De Rose, and Rajkumar Buyya. CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services

[15] Tom Guérout, Thierry Monteil, Georges Da Costa, Rodrigo Neves Calheiros, Rajkumar Buyya, Mihai Alexandru. Energy-aware simulation with DVFS

[16] K Thulasiraman, MNS Swamy - Graphs: Theory and Algorithms, 1992 - Acyclic Directed Graphs

[17] W Chen, E Deelman. Workflowsim: A toolkit for simulating scientific workflows in distributed environments

[18] H Khazaei, J Misic, VB Misic - Performance of Cloud Centers with High Degree of Virtualization under Batch Task Arrivals

[19] Cotes-Ruiz, I. T., Prado, R. P., García-Galán, S., Muñoz-Expósito, J. E., & Ruiz-Reyes, N. (2017). Dynamic Voltage Frequency Scaling Simulator for Real Workflows Energy-Aware Management in Green Cloud Computing. PloS one, 12(1), e0169803.

[20] Smith, S. F., 1980. A learning system based on genetic adaptive algorithms. Ph.D. thesis, Pittsburgh, PA, USA.

[21] García-Galán, R.P. Prado, J. E. Muñoz Expósito Swarm Fuzzy Systems: Knowledge Acquisition in Fuzzy Systems and its Applications in Grid Computing IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING. Accepted.

9. GLOSARIO DE TERMINOS

Antecedentes: utilizado en FRBS. Hace referencia a las variables de entradas del FRBS.

Algoritmo de (meta) planificación: referido al proceso software el cual establece la estrategia de planificación en una entidad (meta) planificadora.

BUSY: referido a un estado de un CondorVM la cual está realizado el procesamiento de una tarea.

CloudInformationService: entidad de CloudSim y derivados. Realiza la función de servidor de registro de las entidades del tipo Datacenter para poder ser utilizadas por las entidades de planificación.

Cloudlet: utilizado en CloudSim y derivados. Clase java la cual representa a una tarea básica a ser ejecutada por un Datacenter.

CLOUDLET_RETURN: evento utilizado por las entidades Datacenter para devolver una Cloudlet ejecutada a una entidad Scheduler. Además este evento también es utilizado por las entidades Scheduler para enviar dichas tareas a la entidad Engine o MetaScheduler, y desde la entidad MetaScheduler a la entidad Engine.

CLOUDLET_SUBMIT: evento utilizado por las entidades Engine, MetaScheduler y Scheduler, para enviar Cloudlets desde la entidad Engine a la entidad MetaScheduler, de esta a las entidades Schedulers, y por último, estas a sus respectivos Datacenters.

CLOUDLET_UPDATE: evento de WorkflowSim y derivados. Este evento, interno de los Schedulers, provoca la planificación de las Cloudlets recibidas a ejecutar.

CloudSim: simulador de *cloud computing* el cual permite una primera y muy básica aproximación a la simulación de *cloud computing*.

CloudSim con DVFS: entorno de simulación derivado de CloudSim al cual se ha integrado la tecnología DVFS para realizar la medición de consumos por la ejecución de tareas.

CloudSimShutdown: entidad del entorno de simulación CloudSim y derivados. Realiza la función de finalizar la simulación.

ClusteringEngine: entidad de WorkflowSim y derivados. Crea las entidades WorkflowEngine y WorkflowScheduler.

ClusteringEngine_META: evolución de ClusteringEngine para funcionar con WorkflowMetaScheduler.

CondorVM: evolución de la clase Vm la cual permite guardar el estado de la VM, tal como IDLE o BUSY.

Consecuente: utilizado en FRBS. Hace referencia a la variable de salida del FRBS.

Conservative: tipo de gobernador dinámico el cual utiliza un umbral superior y otro inferior de carga de trabajo para decidir si la frecuencia y tensión de funcionamiento de la CPU debe ser aumentada o disminuida.

CPD: acrónimo de Centro de Procesamiento de Datos. Se refiere a un conjunto de host unificados y centralizados con el objetivo de realizar el procesamiento masivo de tareas.

CPU: acrónimo de *Central Processor Unit* o unidad central de procesamiento. Referido al componente hardware de un host el cual realiza el procesamiento de tareas.

Datacenter: hace referencia a un CPD. Voz inglesa.

Datacenter (entidad): entidad de CloudSim. Realiza la función de un *datacenter* o CPD.

DatacenterBroker: entidad de CloudSim. Realiza la función de planificación de tareas sobre una entidad Datacenter.

DAG: acrónimo de *Directed Acyclic Graph* o gráfico acíclico dirigido. Utilizado para establecer un orden de prioridad en la ejecución de tareas.

DAX: DAG escritos siguiendo una estructura XML.

DB: acrónimo de Data Base o base de datos. Contiene la definición de las MF de cada uno de los antecedentes y consecuentes del FRBS.

Defuzzyficador: componente del FRBS. Realiza la función de desnormalización del consecuente.

DVFS: acrónimo de *Dynamic Voltage and Frecuency Scaling* o tensión dinámica y escalado de frecuencia. Referido a las técnicas utilizadas por los CPUs modernos las cuales permiten la variación automática de la tensión y frecuencia de funcionamiento en función de la carga de trabajo.

END_OF_SIMULATION: evento ejecutado tras la ejecución de todas las tareas. Supone el fin de la simulación

Entidad: hace referencia a cada uno de los elementos de los simuladores CloudSim y derivados los cuales realizan la creación y atención de eventos.

Evento: mensaje utilizado para establecer la comunicación entre entidades.

fileSize: tamaño de almacenamiento de una Cloudlet previo a su procesado.

FL: acrónimo de *Fuzzy Logic* o lógica borrosa. Referido a la capacidad de trabajar con valores no precisos.

FRBS: acrónimo de *Fuzzy Rules Based System* o sistema basado en reglas borrosas. Realiza el empleo de FL para la solución de problemas. Compuesto por los componentes fuzzyficador, defuzzyficador, motor de inferencia y KB.

Fuzzyficador: componente del FRBS. Realiza la función de normalización de los antecedentes.

GeneralParametersMetaFuzzy: clase estática la cual incluye una serie de parámetros utilizados por el MetaScheduler.

Gobernador: utilidad software implementada a nivel de *Kernel* del SO el cual establece las frecuencias y tensiones de funcionamiento para una CPU

Gobernador dinámico: tipo de gobernador el cual mantiene unos valores fijos y constantes de tensión y frecuencia de funcionamiento.

Gobernador estático: tipo de gobernador el cual permite variaciones en la tensión y frecuencia de funcionamiento en función de la carga de trabajo.

HaaS: acrónimo de *Hardware as a Service* o hardware como un servicio. Tipo de *cloud* en el cual se entrega el hardware al usuario final, disponiendo este de responsabilidad máxima frente a gestión y configuración e implementación.

Host: Equipo el cual cuenta con la capacidad de poder procesar tareas.

Host (simulador): utilizado en CloudSim y derivados. Representa a un host el cual realiza el procesado de tareas. Está constituido por uno o varios Pe, RAM, almacenamiento, y ancho de banda.

IaaS: acrónimo de *Infraestructure as a Service* o infraestructura como un servicio. Tipo de cloud en el cual el usuario final dispondrá de la capacidad de gestión y configuración del entorno cloud sin acceso al hardware.

IDLE: se refiere al estado ocioso de una CondorVM la cual no está realizando ningún procesamiento, y, por tanto, se le podrá asignar una tarea a ser procesada.

IPC: acrónimo de *Instructions Per Cycle* o instrucciones por ciclo. Referido a las instrucciones por ciclo que un Pe o CPU puede realizar.

jFuzzyLogic: librería java para la implementación de un FRBS.

JOB_SUBMIT: evento de WorkflowSim y derivados. Utilizado tras el evento START_SIMULATION, provoca el envío de dicha lista de Cloudlets a la entidad Engine.

KASIA: acrónimo de *Knowledge Acquisition with a Swarm Intelligence Approach* o adquisición de conocimiento con un enjambre inteligente. Persigue mediante la utilización de PSO realizar el aprendizaje de una DB de un FRBS.

Kernel: referido al núcleo de un SO, el cual realiza ciertas funciones de control y comunicación a bajo nivel con el hardware del host.

KB: acrónimo de *Knowledge Base* o base de conocimiento. Compuesto por la DB y la RB.

MaxMinMetaSchedulingAlgorithm: evolución del algoritmo MaxMinSchedulingAlgorithm para funcionar en el MetaScheduler.

MaxMinSchedulingAlgorithm: algoritmo de planificación el cual prioriza la ejecución de las tareas de mayor longitud (length) en aquellas VM las cuales disponen de unas mayores prestaciones o MIPS. Optimiza los tiempos de ejecución.

Meta-Planificación: proceso de planificación realizado a nivel de Meta-Planificador. Consiste en la asignación de tareas a los planificadores de los respectivos CPDs o *datacenters* en función de sus características.

Meta-Planificador: planificador de planificadores que realiza la función de Meta-Planificación.

MetaScheduler: referido al Meta-Planificador. Voz inglesa.

MF: acrónimo de *Membership Fuction* o función de pertenencia. Utilizado en lógica borrosa para delimitar los diferentes rangos de valores para los antecedentes y consecuentes.

MinMinMetaSchedulingAlgorithm: adaptación de MinMinSchedulingAlgorithm para funcionar en el MetaScheduler.

MinMinSchedulingAlgorithm: algoritmo de planificación el cual prioriza la ejecución de las tareas con menor longitud (length) en aquellas VM las cuales disponen de unas mayores prestaciones o MIPS. Optimiza los tiempos de ejecución.

MIPS: acrónimo de millones de instrucciones por segundo. Hace referencia a las prestaciones o capacidad computacional que puede ofrecer un host, VM, o cualquier otro dispositivo inteligente con capacidad para el procesamiento de tareas.

Motor de inferencia: utilizando la KB y los antecedentes de un sistema, se encarga de proporcionar una salida o consecuente a este.

Length: referido a la longitud de una Cloudlet. Dicha longitud especifica los recursos computacionales requeridos para ejecutar dicha Cloudlet.

onDemand: tipo de gobernador dinámico el cual utiliza un umbral superior de carga de trabajo y un contador para la regulación de la frecuencia y tensión de funcionamiento.

On-Batch: método de planificación el cual realiza el acopio de una cierta cantidad de tareas o *Cloudlets* para realizar el proceso de planificación de dicho conjunto de tareas.

On-Line: método de planificación en el cual el planificador realizará la planificación de una tarea o Cloudlet tan pronto como esta sea recibida.

outputSize: tamaño de almacenamiento de una Cloudlet tras su procesado.

PaaS: acrónimo de *Platform as a Service* o plataforma como un servicio. Tipo de *cloud* el cual proporciona al usuario final control total sobre el SO, otorgando a este control completo sobre su gestión y mantenimiento.

Parser: entidad de WorkflowSim y derivados. Realiza la lectura y conversión de tareas definidas en un fichero DAX a una lista de Cloudlets con las cuales pueden trabajar los simuladores WorkflowSim y derivados.

Pe: utilizado en CloudSim y derivados. Clase java la cual representa la simulación de un procesador o CPU.

Performance: gobernador estático el cual establece los valores de frecuencia y tensión de una CPU a su valor máximo.

Pittsburgh Approach: Persigue el aprendizaje de una KB de un FRBS por medio del empleo de algoritmos genéticos.

Planificación: proceso de planificación mediante el cual se realiza la asignación de tareas a los diferentes equipos del CPD o *datacenter* asociado a dicho planificador. Utiliza diferentes algoritmos de planificación para llevar a cabo dicha tarea.

Planificador o planificador local: entidad que realiza el proceso de planificación.

PowerAwareFuzzyMaxMaxMetaSchedulingAlgoritm: adaptación del algoritmo de planificación PowerAwareFuzzyMaxMaxSchedulingAlgorithm para funcionar en el MetaScheduler.

PowerAwareFuzzyMaxMaxSchedulingAlgorithm: basado en el algoritmo de planificación PowerAwareMaxMinSchedulingAlgorithm. Persigue la optimización de energía haciendo uso de FRBS.

PowerAwareFuzzyMinMaxSchedulingAlgorithm: basado en el algoritmo de planificación PowerAwareMinMinSchedulingAlgorithm. Persigue la optimización de energía haciendo uso de FRBS.

PowerAwareMaxMinSchedulingAlgorithm: evolución del algoritmo MaxMin el cual persigue la optimización de consumos.

PowerAwareMinMinSchedulingAlgorithm: evolución del algoritmo MinMin el cual persigue la optimización de consumos.

PowerDatacenter: utilizado en CloudSim con DVFS y derivados. Evolución de la entidad Datacenter de CloudSim, la cual ha sido adaptada para funcionar con DVFS.

PowerDatacenterBroker: utilizado en CloudSim con DVFS y derivados. Evolución de la entidad Broker de CloudSim, la cual ha sido adaptada para funcionar con DVFS.

PowerHost: utilizado en CloudSim con DVFS y derivados. Evolución de la clase java *Host*, la cual ha sido adaptada para funcionar con DVFS.

PowerModelAnalytical: modelo de potencia de WorkflowSim con DVFS, el cual permite definir de manera paramétrica las características de un Pe.

PowerVm: utilizado en CloudSim con DVFS y derivados. Evolución de la clase java Vm, la cual ha sido adaptada para funcionar con DVFS.

PowerSave: gobernador estático el cual establece la frecuencia y tensión de funcionamiento de una CPU a su valor mínimo.

PSO: acrónimo de *Particle Swarm Optimization* u optimización por enjambre de partículas. Persigue la maximización o minimización de una función a través de un enjambre de partículas. Dicho enjambre procederá a compartir información con el resto de la población con el objetivo de hallar el mejor resultado posible.

RAM: acrónimo de *Random Access Memory* o memoria de acceso aleatorio. Referida a la memoria principal de un host o VM utilizada para el almacenamiento temporal de tareas en ejecución.

RB: acrónimo de Rule Base o base de reglas. Constituye el conjunto de reglas el cual establece el comportamiento o consecuente para unos valores particulares de los antecedentes.

REGISTER_RESOURCE: evento ejecutado por entidades del tipo Broker, Scheduler y *Datacenter* para proceder a su registro en la entidad CloudInformationService

RESOURCE_CHARACTERISTICS: evento utilizado por las entidades Scheduler para solicitar las características de una entidad Datacenter.

RESOURCE_CHARACTERISTICS_REQUEST: evento utilizado por las entidades Engine y *Scheduler* mediante el cual la entidad Engine conocerá la existencia de las entidades *Schedulers*.

RoundRobinSchedulingAlgorithm: algoritmo de planificación básico el cual no realiza ninguna asignación inteligente de tareas. Realiza un reparto equitativo-circular de Cloudlets entre todas las VM de un Datacenter.

RoundRobinMetaSchedulingAlgorithm: adaptación del algoritmo de planificación RoundRobinSchedulingAlgorithm para funcionar en el MetaScheduler.

SaaS: acrónimo de *Software as a Service* o aplicación como un servicio. Tipo de *cloud* el cual ofrece una aplicación o conjunto de aplicaciones como un servicio.

Scheduler: referido a un planificador. Voz inglesa.

SchedulingInterval: valor temporal utilizado por la entidad PowerDatacenter para fijar el intervalo de tiempo de simulación por iteración.

SO: acrónimo de sistema operativo. Referido al conjunto software básico el cual permite la iteracción de otros softwares con el conjunto hardware de un host.

START_SIMULATION: evento de WorkflowSim y derivados. Ejecutado en el inicio, realiza la conversión de un DAX a un listado de Cloudlets.

Tag: etiqueta de CloudSim y derivados la cual sirve para identificar el tipo de evento.

UserSpace: gobernador estático el cual permite establecer la frecuencia y tensión de funcionamiento manualmente.

VM: acrónimo de *Virtual Machine* o máquina virtual. Referido a un sistema completo ejecutado sobre un host específico para la cual se le realiza la asignación de una serie de recursos de dicho host.

Vm (simulador): clase java utilizada en CloudSim y derivados. Representa a un VM la cual es ejecutada sobre un host.

VM_CREATION_ACK: evento utilizado por las entidades Schedulers para solicitar la creación de VMs en los Datacenters. En el orden inverso, asiente positiva o negativamente la creación de dichas VMs.

VM_DATACENTER_EVENT: evento interno de los Datacenters mediante el cual procede a realizar la ejecución de Cloudlets.

VM_DESTROY: evento utilizado por los Schedulers para solicitar la destrucción de las VMs en los Datacenters.

WorkflowDatacenter: entidad de WorkflowSim y derivados. Supone una evolución de la entidad *Datacenter* de CloudSim. Realiza la función de centralización de *hosts* para el procesamiento de tareas.

WorkflowEngine: entidad de WorkflowSim y derivados. Almacena el conjunto de tareas totales a ejecutar, controlando cuales han sido ejecutadas y cuales pueden proceder a ser ejecutadas. Envía tareas ejecutables a las entidades WorkflowScheduler para su planificación y recibe el resultado de dichas tareas una vez han sido ejecutadas.

WorkflowEngine_META: evolución de WorkflowEngine para funcionar con WorkflowMetaScheduler.

WorkflowMetaScheduler: entidad del simulador desarrollado la cual realiza la función de Meta-Planificador. Procederá a realizar un reparto inteligente de Cloudlets entre los Schedulers existentes haciendo uso de los algoritmos de Meta-Planificación

WorkflowPlanner: entidad de WorkflowSim y derivados. Crea las entidades Parser, ClusteringEngine, WorkflowEngine, y WorkflowScheduler en el inicio de la simulación. Además llama a la entidad Parser para su ejecución en el inicio.

WorkflowPlanner_META: evolución de WorkflowPlanner para funcionar con WorkflowMetaScheduler.

WorkflowScheduler: entidad de WorkflowSim y derivados. Supone una evolución de la entidad DatacenterBroker, realizando el proceso de planificación de Cloudlets para ser ejecutadas en un WorkflowDatacenter asociado.

WorkflowScheduler_META: evolución de WorkflowScheduler para funcionar con WorkflowMetaScheduler.

WorkflowSim: evolución de CloudSim el cual permite la simulación de conjuntos de tareas con dependencias entre sí por medio del uso de DAX.

WorkflowSim con DVFS: mejora de WorkflowSim la cual incluye la tecnología DVFS para la medición de consumos.

WorkflowSimTags: clase java la cual incluye la definición de los nuevos tipos de eventos introducidos en WorkflowSim.

XaaS: acrónimo de X as a Service o cualquier cosa como un servicio. Hace referencia a cualquier cosa como un servicio, pudiendo ser este del tipo SaaS, PaaS, IaaS, o HaaS