Universidad Zaragoza

Trabajo Fin de Grado

Sistema de ejecución de funciones serverless en el continuum de la nube con una gestión del consumo energético inteligente

Adaptative energy aware system for serverless function in cloud continnum

Autor

Íñigo Aréjula Aísa

Directores

María Canales Compés

Rafael Tolosona Calasanz

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2022

RESUMEN

En la actualidad, debido a la escasez de energías fósiles y la variabilidad de las renovables las fluctuaciones en el precio de la energía son significativas. Tradicionalmente, los sistemas computacionales ha priorizado maximizar sus prestaciones (tiempo de ejecución o *throughput*), mientras que el ahorro energético solo se considera una vez garantizadas las prestaciones. Sin embargo, con el incremento de los precios, limitar el gasto energético es inevitable. En este trabajo fin de grado, se considera un tipo de aplicación distribuida que genera datos continuamente para su procesamiento. Para ello, se considera un centro de datos con gran capacidad computacional y pequeños clústers de capacidad limitada, próximos a las fuentes de datos. El sistema sigue una estrategia de scheduling completamente distribuido: los datos generados en la fuente se llevan al clúster de poca capacidad más próximo, si hay posibilidad de procesar los datos, se ejecutan ahí; si no, se transmiten al centro de datos.

Además, el sistema cuenta con un modelo energético y monitoriza la carga de trabajo y el coste de la energía, de manera que si el coste de la energía es bajo, el sistema maximiza las prestaciones; en caso contrario, el sistema se reconfigura, para no superar el coste máximo permitido. A diferencia de las definiciones tradicionales de los niveles de calidad de servicio, cuando el coste de la energía exceda lo permitido, *las prestaciones se degradan lo necesario*. La degradación viene dada por los requisitos de la aplicación: reduciendo la precisión del procesamiento o aumentando el tiempo de ejecución.

Para conseguir esto, primero se analizan los distintos factores (en los niveles de aplicación, *middleware* y recursos hardware) que contribuyen al consumo energético. Después, experimentalmente, se construye un modelo energético para cada clúster. En ejecución, mediante un algoritmo voraz, el controlador de cada clúster determina cuál es la configuración energética que maximiza el *throughput*, garantizando siempre las restricciones energéticas. Para validar la aproximación, se ha utilizado una aplicación de análisis de vídeo y se han realizado varios experimentos en una infraestructura compuesta por dos clústers: uno con raspberry pis y otro, con rol de centro de datos, con máquinas virtuales. Los experimentos muestran cómo el sistema es capaz de adaptarse a las restricciones de energía y consumir menos, disminuyendo el *throughput* y/o la precisión de los procesamientos.

Índice

Capítulo 1

Introducción y objetivos

Recientes estimaciones indican que el gasto energético global por computación y las redes de comunicación podría ser del 20 % según [1]. Cada año se desarrollan nuevas aplicaciones y servicios que demandan una mayor cantidad de capacidad computacional, así como las redes de comunicación y los sistemas de almacenamiento que estas necesitan. Se estima que los centros de datos pueden ser responsables de hasta un 3 % de la emisión de CO2 [1]. Por otra parte, en la actualidad hay un periodo de transición en las fuentes de energía, de unas fuentes provenientes de materiales fósiles y altamente contaminantes a unas fuentes de energía limpias y renovables. Además, hay dos factores que hacen que la transición no sea una cuestión fácil. Por un lado, debido a los problemas logísticos y de inversión en la extracción de las energías fósiles, y a la situación geopolítica, la diferencia en el coste de la energía cuando se priman las energías fósiles o las renovables puede ser significativa. De manera que, los periodos en los que se pueda fundamentar la producción energética en renovables tendrán un coste energético relevantemente menor que cuando la producción energética se tenga que apoyar en fuentes fósiles. Por otro lado, en la actualidad no somos eficientes almacenando la energía que producimos, por lo tanto, los periodos en los que hay energía barata y abundante no se pueden aprovechar para almacenar la energía que se produce (y utilizarla cuando su coste se eleve)

Sin duda, el consumo energético de los sistemas computacionales se ha analizado exhaustivamente en el pasado en todos los ámbitos arquitecturales, desde el hardware hasta las aplicaciones software: por ejemplo, el consumo energético de los centros de datos ha sido objeto de estudio [2]. También se ha investigado el consumo energético en las infraestructuras distribuidas emergentes, como son la computación en la nube [3] o la más reciente computación perimetral [4] (*edge computing* en inglés). Además, existen multitud de estudios que analizan la distribución de tareas por recursos (*scheduling*) de manera que el consumo energético se minimice [5]. En el ámbito de las aplicaciones software, se ha propuesto la utilización de técnicas de computación

aproximada con el objetivo de reducir el tiempo de ejecución y, así, reducir el consumo energético [6]. Incluso las administraciones públicas realizan ya acciones destinadas a impulsar la eficiencia energética en la implementación de aplicaciones informáticas. Buen ejemplo de ello es el programa de certificación energética para aplicaciones informáticas desarrollado por el gobierno alemán [1].

Sin embargo, probablemente no se le ha dado la importancia suficiente al estudio del consumo energético en los sistemas computacionales. Las cuestiones energéticas se han relegado a un plano claramente secundario,a favor de una mayor atención hacia los aspectos relacionados con el rendimiento de los sistemas (medido en términos de tiempo de ejecución y *throughput*, fundamentalmente). Algunos ejemplos de esto se reflejan en diversos trabajos [7, 8], en los cuales se intenta reducir el consumo energético de un centro de datos, minimizando el número de violaciones de la calidad del servicio: por tanto, la energía se considera como un aspecto secundario. Por otro lado, también se observa un tratamiento similar en los planes de estudio de los grados y másteres en informática: desde el principio, los estudiantes aprenden de qué manera los componentes de un sistema contribuyen a su rendimiento. Es significativo, quizá, que en los temarios o en los libros de texto de Ingeniería del Software no aparezca la energía. Por tanto, el primer aprendizaje que se recibe sobre la construcción de software no cuenta con aspectos relacionados con el consumo energético. Seguramente, con la crisis energética actual todo esto va a tener que cambiar.

En este trabajo fin de grado se plantea construir un sistema distribuido que sea capaz de adaptarse al contexto energético de forma autónoma durante su ejecución. Para ello, en primer término, se analiza qué factores contribuyen al consumo energético en los distintos niveles arquitecturales de los sistemas distribuidos: en el nivel de la aplicación, en el de la gestión de los recursos y en el del hardware. En el nivel de aplicación se considera un tipo de aplicación cuya ejecución acepta distintos grados de precisión, controlable mediante parámetro. En este proyecto se va a utilizar una aplicación desarrollada por el mismo autor durante las prácticas de iniciación a la investigación realizadas en el grupo COSMOS. La aplicación, denominada *pymemo*, clasifica y detecta objetos en vídeos, utilizando la biblioteca *opencv*. La precisión en la aplicación puede variar si en lugar de aplicar la clasificación y la detección a cada frame, esta primera solo se aplica al primer frame de una subsecuencia de frames consecutivos que se consideran *similares*. En la gestión de los recursos, se opta por una estrategia de *scheduling* completamente distribuida, en la que cada tarea se ejecuta en los recursos computacionales cerca de las fuentes de datos (*edge*), siempre que haya disponibilidad.

[1] `https://produktinfo.blauer-engel.de/uploads/criteriafile/en/DE-UZ%20215-202001-en-Criteria-2020-02-13.pdf`

Si no la hay, se ejecutará en el centro de datos. Para reducir el consumo energético en este ámbito se considera reducir la utilización de los recursos, y por consiguiente, incurriendo probablemente en un tiempo de espera, o bien incurriendo en una menor precisión. Finalmente, en el ámbito del hardware se considera variar la frecuencia hasta encontrar una que consiga reducir el consumo energético.

Para comprender de qué manera afectan todos estos factores primero se realiza un conjunto de experimentos, combinando todos los parámetros que pueden afectar al consumo energético (*parameter sweep*). A partir de estos experimentos se crea un modelo de consumo energético que contempla un amplio abanico de escenarios y que se utiliza por el sistema en tiempo de ejecución para la toma de decisiones, de manera que cuando el coste energético aumenta el sistema puede desencadenar distintas acciones que degradan la precisión o aumentan el tiempo de ejecución. Cuando el coste energético disminuye, el sistema opta por acciones que contribuyen a minimizar el tiempo de ejecución. Para realizar estas acciones, el sistema sigue un bucle MAPE-K (monitorización, análisis, planificación y ejecución) y un algoritmo voraz para encontrar la configuración que mejor pueda satisfacer los requisitos energéticos o de prestaciones en cada momento.

Para demostrar el funcionamiento se ha configurado una infraestructura distribuida, que consta de un clúster perimetral (compuesto de *raspberry pi*) y un pequeño clúster a modo de centro de datos. En este último se han instalado kubernetes y openfaas. En el clúster perimetral se han instalado kubernetes y servidores http. En ambos se han realizado determinados experimentos en los que puede observarse cómo el sistema puede adaptarse a los distintos contextos energéticos.

El trabajo se ha realizado en el marco del grupo de investigación COSMOS, en colaboración con los profesores María Canales (Ingeniería Telemática de la Universidad de Zaragoza), Alejandro Calderón (Arquitectura de Computadores de la Universidad Carlos III de Madrid) y Gabriel González (Insight Insitute de University College Cork, Irlanda). Todo el código desarrollado en este proyecto se puede encontrar en [2]. Se debe remarcar que en el repositorio *function* se encuentra el código de pymemo, el cual no es parte del trabajo, sino de las prácticas realizadas por el alumno.

El resto de esta memoria se ha estructurado de la siguiente forma: en el Capítulo 2 se describen brevemente conceptos, técnicas y herramientas utilizadas para la elaboración de este trabajo. En el Capítulo 3 se propone una arquitectura para el sistema de este trabajo, y se analizan sus componentes, por otro lado en el Capítulo 4 se presenta el modelo de consumo energético del sistema, analizando qué factores contribuyen al consumo energético, con qué métodos y herramientas puede obtenerse el modelo y

[2] `https://github.com/TFG-arejula27`

cómo puede utilizarse el modelo durante la ejecución. Seguidamente, en el Capítulo 5 se presentan unos experimentos para validar la propuesta. Finalmente en el Capítulo 6 se presentan las conclusiones y el trabajo futuro.

Capítulo 2

Conceptos, Metodología y Tecnología

2.1. IoT, Fog and Cloud computing

IoT La Internet de los cosas (*Internet of Things*, IoT) describe objetos físicos (o grupos de objetos) con sensores o incluso con capacidad de procesamiento, software, servicios y otras tecnologías que se conectan e intercambian datos con otros dispositivos y sistemas a través de Internet u otras redes de comunicación. En la actualidad, existen dispositivos capaces de generar grandes cantidades de métricas, las cuales deben ser transmitidas y procesadas. Este conjunto de dispositivos es muy heterogéneo en muchos aspectos: conectividad, protocolo, métricas, etc. La información que generan los dispositivos IoT no es útil como tal, sino que requiere un tratamiento en aplicaciones IoT, donde a partir del conjunto de datos y un procesamiento se obtiene información útil.

Para el caso de este trabajo, se considera el uso de sensores como fuentes de generación continua de datos que tienen que enviarse a los recursos computacionales para realizar un análisis o procesamiento. Sin duda, la monitorización y el análisis masivos y continuos permite a los sistemas evaluar la situación en tiempo real y tomar acciones (como subir la temperatura, activar una bomba de agua) mediante actuadores o comunicar la situación a un ser humano para que decida qué acción tomar (activar una alarma, crear un informe, etc.).

Cloud De acuerdo con la definición oficial de cloud computing propuesta por la entidad NIST, cloud computing [1] es un modelo que permite un acceso adecuado, ubicuo y bajo a un conjunto de recursos configurables (entendiendo CPU, almacenamiento, red de comunicación, aplicaciones o servicios como recursos). Esto no es más que un

[1] `https://csrc.nist.gov/publications/detail/sp/800-145/final`

paradigma donde los recursos son abstraídos y proporcionados por un proveedor, de tal forma que el usuario no tiene que preocuparse de sus limitaciones ni gestiones, realizando únicamente un pago por el uso. Este paradigma es totalmente contrario al tradicional, donde el usuario debe gestionar sus máquinas virtuales y sufragar costes independientemente de su rendimiento. El sistema *Cloud* aporta una gran flexibilidad y dinamismo a los recursos hardware. Un elemento fundamental en el *Cloud* es la encapsulación de recursos y aislamiento de estos. Todos los procesos se ejecutan sobre un conjunto de máquinas físicas. El método de virtualización más tradicional es realizado mediante máquinas virtuales, éstas simulan una máquina física sobre otra, delegando la gestión de recursos a la máquina virtual. Otro método es mediante contenedores: éstos son una encapsulación de software, el cual comparte recursos con el sistema operativo anfitrión pero esta completamente aislado del resto de procesos.

Fog computing Los sistemas *Cloud* están en centros de datos. Sin embargo, con la proliferación de los sensores, y las redes móviles las fuentes de datos se encuentran en cualquier ubicación. Por tanto, transmitir todos los datos generados a un centro de datos en una localización particular no es razonable desde un punto de vista de la escalabilidad. Además, con las distancias entre las fuentes de datos y los centros de datos aparecen latencias e incrementos en los tiempos de respuesta. Por todo ello, se ha planteado la utilización de recursos computacionales que se encuentran cerca de las fuentes de datos. Pero dichos recursos suelen ser de menor capacidad en comparación con los recursos de los centros de datos, por lo que para mantener la calidad del servicio en términos de prestaciones la precisión de los procesamientos tiene que reducirse. Este modelo crea una relación jerárquica entre los distintos componentes, tanto del *Cloud* como del *IoT*, estableciendo distintos niveles de servicio en cada uno de ellos.

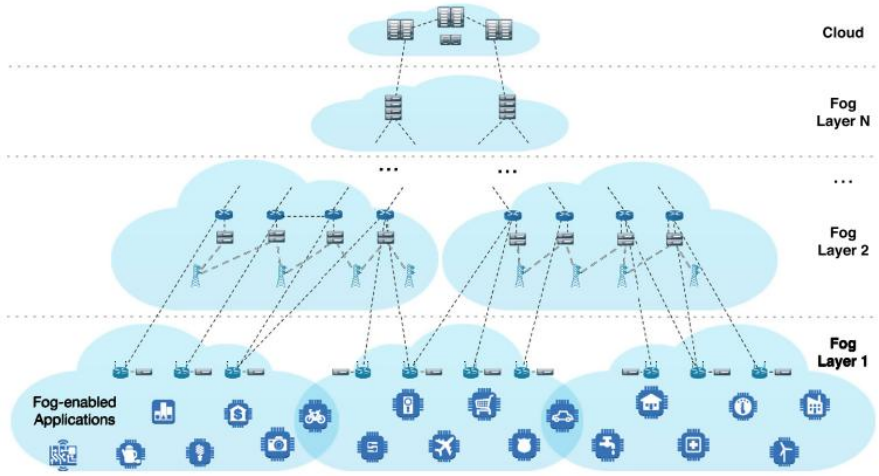


Figura 2.1: Modelo Arquitectural de Fog Computing, extraído de [9]

Este modelo se denomina computación en la niebla o *Fog computing* [4]. La figura 2.1 muestra un ejemplo de la arquitectura: esta tiene distintos niveles desde los dispositivos que obtienen los datos (nivel inferior), hasta el *Cloud* (superior). Se aprecia una arquitectura jerárquica, donde los niveles inferiores están muy cercanos al usuario o a las fuentes de datos, convergiendo hacia los niveles superiores. Esta arquitectura añade una complejidad elevada a nuestro sistema debido a su gran heterogeneidad, por lo que dificulta su gestión. Además, las soluciones tradicionales centralizadas son difícilmente aplicables, por lo que el *scheduling* se realiza de forma distribuida, estrategia que se ha adoptado en este trabajo.

2.2. Serverless computing: Kubernetes and OpenFaaS

Serverless computing, faas *Serverless* es un nuevo modelo de computación, que según IBM [2] se define como un modelo de aplicación en el cual los desarrolladores pueden construir y ejecutar aplicaciones distribuidas sin necesidad de gestionar y asignar servidores. Durante la ejecución, solo se paga por lo que se consume (se factura habitualmente por milisegundo o fracción) y no por los ciclos en los que la máquina no está ejecutando nada. Este nuevo modelo tiene muchas ventajas como la abstracción de los recursos: así, el desarrollador puede centrarse en la aplicación sin preocuparse de la infraestructura. Además, este modelo facilita el escalado horizontal y la alta disponibilidad de las aplicaciones mediante la auto replicación de recursos. Todo esto se da en el entorno denominado *Cloud*, con proveedores como Amazon Web service o Microsft Azure.

Dentro de los modelos de *serverless* se encuentra el modelo de Function as a Service (FaaS) [10], que consiste en ofrecer la ejecución de código como una función, cuya invocación se realiza de forma síncrona o asíncrona, como reacción a un evento. Muchas veces estos términos se confunden y se entienden como sinónimos, pero *Serverless* es mucho más amplio, haciendo referencia a cualquier tipo de recurso como código, almacenaje o enrutadores.

Faas puede verse como una combinación del antiguo paradigma de arquitectura orientada a servicios, donde se trata de abstraer la implementación ofreciendo una interfaz de servicios y RPC (llamada de procedimientos remotos), donde el desarrollador utiliza funciones conociendo únicamente su especificación, invocándolas mediante el protocolo RPC, sin necesidad de conocer en qué servidor se alojan las funciones ni su implementación.

[2] `https://www.ibm.com/cloud/learn/serverless`

Este nuevo modelo aporta muchas ventajas, como el desacoplamiento de las funciones, la gestión individual de ellas (ya que modificar una función no implica perder disponibilidad en las demás), la fácil gestión de versiones, la abstracción de ellas y la falta de necesidad de gestionar la infraestructura donde se ejecutan. Pero también tienen ciertos inconvenientes, posiblemente debidos a lo prematuro que es este paradigma. El código es menos legible (en vez de invocar a una función se hace una llamada http) y es difícil crear flujos de invocaciones complejos, teniendo que ser la mayoría de ellos secuenciales.

kubernetes y OpenFaaS Para poder gestionar los recursos en el *Cloud* se necesita un orquestador. Este será el encargado de asignar los contenedor a las distintas máquina, así como de crear, configurar y eliminar los despliegues. En este trabajo se usará Kubernetes [11]: como se definen en su web, es un orquestador para automatizar los despliegues, el escalado y la gestión de contenedores en el *Cloud*. Kubernetes permite especificar mediante archivos de texto el despliegue de aplicaciones y este se encargará de que siempre se encuentren en un estado correcto definido: esto significa que estará continuamente monitorizando el estado de cada aplicación desplegada, y si por ejemplo una se encuentra en una máquina caída, esta automáticamente se lanzará en otra para estar disponible. Existen muchas implementaciones de kubernetes, siendo la original k8s. Para este proyecto se ha considerado la alternativa k3s[3]: esta versión es mucho más ligera, por lo que permite su ejecución en las *raspberries* (las cuales no tienen recursos suficientes para ejecutar la versión original). Además, que consume menos energía, manteniendo una coherencia con la propuesta inicial.

Mediante Kubernetes se desplegará OpenFaas [12]: este framework será el que se utilizará para el despliegue de nuestra aplicación Faas, y se encargará de su despliegue, disponibilidad, acceso y obtención de métricas. Mediante su aplicación de linea de comandos se crearán las imágenes de los contenedores las cuales se desplegarán [13], permitiendo así que como desarrolladores se limiten a escribir el código de las funciones que queramos implementar, delegando las tareas citadas previamente.

2.3. Mediciones de Energía y Procesadores: Intel y ARM

Para crear y validar nuestro modelo se han realizado experimentos midiendo la potencia sobre los distintos niveles del sistema. En los sistemas distribuidos la hetereogenidad de los componentes es muy alta: los ordenadores que se emplean tienen

[3] `https://k3s.io/`

procesadores con diversas arquitecturas. Las máquinas utilizadas han sido raspberries con procesadores ARM, mientras que el centro de datos usa un procesador Intel. Por ello, la forma de medir la potencia es distinta.

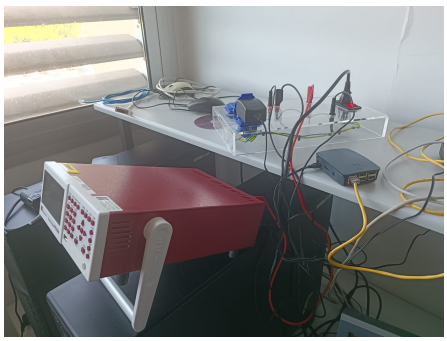
Intel Los procesadores Intel modernos [4] cuentan con una funcionalidad denominada Inter Hardware P-State (HWP), por la que el procesador intenta ajustar la frecuencia de la CPU y el voltaje para que sean los más adecuados para satisfacer las prestaciones, a la vez que se consume la menor cantidad de energía posible. Bajo el sistema operativo Linux Ubuntu, existen herramientas que permiten controlar el comportamiento de HWP. En particular, existen 5 políticas que pueden seleccionarse para modificar el comportamiento de HWP —*conservative*, *ondemand*, *schedutil*, *powersave* y *performance*—, al mismo tiempo que se pueden habilitar o deshabilitar las frecuencias denominadas turbo. Por ejemplo, en la política *performance* se priman las prestaciones frente a la energía, por lo que la CPU se suele fijar a una alta frecuencia; mientras, que en la política *powersave* se prima el ahorro energético. Por otra parte, la política *userspace* permite al administrador de la máquina fijar como máximo una frecuencia dada.

Para medir la potencia utilizada por las máquinas, se ha utilizado la herramienta *powerstat*: este programa utiliza datos expuestos por el kernel de Linux en procesadores Intel. Se han considero correctas las mediciones de energía por software según el trabajo [14]. Esta herramienta no solo nos ofrece la potencia en vatios, sino también otras métricas de gran utilidad como la frecuencia del procesador o los C-state del procesador.

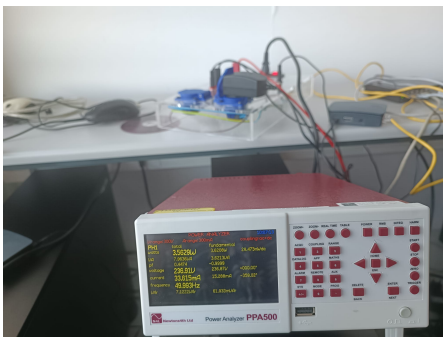
Para realizar estos experimentos de forma automatizada se creó una herramienta en Go que permite realizar las mediciones, volcar los resultados en un fichero csv, configurar la frecuencia del procesador y ejecutar el programa a medir [5].

ARM La herramienta powerstat no recoge métricas en CPUs con arquitectura ARM, por lo que las mediciones de la potencia hay que realizarlas con un dispositivo hardware. La medición se hizo interponiendo un multímetro entre la fuente de corriente y la propia *rasberry* como se muestran en las figuras 2.2b y 2.2a. De esta forma se ejecutan los programas y se modifican las frecuencia de forma manual desde un terminal, y las mediciones se guardarán en ficheros csv, conectando el multímetro a un ordenador mediante el software para windows del fabricante.

[4] `https://smackerelofopinion.blogspot.com/`
[5] `https://github.com/arejula27/measurepymemo`



(a) Lateral



(b) Frente

Figura 2.2: Medición con multímetro

2.4. Virtual Box y Vagrant

A la hora de realizar los experimentos se han creado entornos totalmente controlados y replicables. Vagrant [15] permite definir una infraestructura virtual mediante ficheros de texto,es decir, un conjunto de máquinas virtuales con sus características hardware y su red de interconexión. Desde Vagrant se crearon las máquinas virtuales, se automatizó la creación del clúster de Kubernetes y se realizó el despliegue de OpenFaas. Vagrant ofrece una gran variedad de hipervisores: se optó por *virtual box*, debido a la familiaridad que se tenía con dicho software frente a otros, además de encontrar más documentación sobre él. Esta automatización se aplicó al servidor que simulan el centro de datos, pero no al clúster de *raspberries*, ya que como prueba de concepto se trató de desplegar de forma sencilla y a partir de los recursos disponibles, realizando la instalación de forma manual.

2.5. Trabajo Relacionado

La gestión del consumo de la energía en los centros de datos puede dividirse en tres grupos principalmente: (i) escalado dinámico del rendimiento [16, 17], que en general tiene que ver con ajustar la frecuencia de los procesadores para ahorrar energía; (ii) heurísticas reactivas [8, 18] que redistribuyen máquinas virtuales en función de la utilización de las máquinas físicas, evitando que unas máquinas tenga alta utilización y otras baja, y (iii) heurísticas predictivas [19, 20] que intentan disminuir la utilización de las máquinas en función de un modelo estadístico de históricos de ejecución, de manera que se intenta predecir la carga de trabajo y qué utilización va a suponer, para así redistribuir la carga de manera que la utilización sea la misma en todas las máquinas.

En general, todos estos trabajos tratan de reducir la utilización, como en nuestro trabajo. En este trabajo se consideran todos los factores que pueden contribuir al

consumo energético en todos los niveles arquitecturales. El contexto del trabajo es una aplicación particular que genera datos continuamente que tienen que procesarse. Para ello, se dispone de un conjunto de máquinas distribuidas y dedicadas para la aplicación. En ese sentido, es factible obtener un modelo experimentalmente que capture el comportamiento de todos esos factores, de manera que el modelo pueda explotarse en tiempo de ejecución. En los trabajos en que se gestiona la energía ajustando la frecuencia, existen algunos de ellos en los que se explora la estructura de una tarea [21] para ajustar la frecuencia en consonancia. En nuestro caso, consideramos la tarea y su contexto de ejecución para ajustar el gasto energético de la tarea (ajustando su umbral, *threshold* en inglés). Sin duda, la diferencia más importante entre nuestro trabajo y el resto es que nos planteamos un contexto energético en el que el precio de la energía puede fluctuar significativamente, de manera que cuando el precio energético comience a subir, nuestro sistema es capaz de ajustar los parámetros para seguir operando sin superar los umbrales de consumo energético que el sistema se puede permitir. Reducir el coste energético implica, necesariamente, aumentar el tiempo de ejecución, reducir la precisión de las ejecuciones o una combinación de ambos. Las aplicaciones en sus acuerdos de la calidad del servicio (*Service Level Agreement* en inglés) podrán establecer qué degradación es tolerable, el sistema tomará las decisiones adecuadas en función de ello. Si ninguna configuración energética permite realizar el procesamiento sin superar el coste energético, nuestro sistema llegará incluso a detenerse.

Capítulo 3

Arquitectura del Sistema

3.1. Pymemo: Aplicación de Vigilancia

Como aplicación para este trabajo se utilizará pymemo. Es importante remarcar que *esta aplicación no es parte del esfuerzo de este trabajo fin de grado*, únicamente hacemos uso de ella para ejemplificar cómo una aplicación puede adaptar su consumo energético.

Aplicación pymemo La aplicación recibe como entrada un vídeo, como salida clasifica y detecta los objetos que aparecen en el vídeo. Para ello utiliza *deep learning*, OpenCV y Python. La aplicación se llama Pymemo por una razón muy simple: está implementada en python, y su característica principal es la aproximación mediante memorización. Es un proyecto FOSS con licencia MIT, es accesible en este repositorio [1].

Su funcionamiento es el siguiente: por cada frame del vídeo (i) clasifica los objetos en el frame y (ii) detecta movimiento. El reconocimiento de objetos está basado en una red neuronal convolucional denominada *Inception*, descrita en [22] y con el código accesible en este repositorio [2].

Como la fase de clasificación toma la mayoría del tiempo de ejecución, se le añade una fase de memorización. Tras la ejecución, pymemo muestra el tiempo que ha necesitado para realizar la clasificación, una etiqueta para cada frame y el método de clasificación del frame (hit = 0) o memorizado (hit = 1). Al final de la ejecución muestra un resumen del análisis de todo el vídeo con métricas globales.

[1] `https://github.com/acaldero/pymemo/`
[2] `https://github.com/JimmyHHua/opencv_tutorials`

Approximate Computing y memorización

Una de las características de pymemo es su capacidad de liberar recursos a cambio de obtener resultados con menor precisión. Esto es fundamental para el proyecto, ya que permite al sistema liberar recursos y obtener un ahorro energético a nivel de aplicación.

Este ahorro se obtiene mediante computación aproximada y memorización: esto significa que si en una subsequencia de frames se encuentran frames consecutivos que son *similares* el procesamiento de clasificación solo se aplicará al primer elemento de la subsequencia. La precisión dependerá, por tanto, de qué umbral se considere para que dos frames dados se consideren *similares*. Para ilustrar el efecto del threshold en las prestaciones y en el error que se introduce se realizaron distintos experimentos cuyos resultados se recogen en la Figura 3.1. Se puede observar cómo al aumentar el threshold porcentualmente, la mejora en las prestaciones es mayor que el porcentaje añadido de error. Así, con un threshold 50, se incurre en un 12 % adicional de error, pero se mejoran las prestaciones en un 40 %.

En este caso particular, no se puede caracterizar el error de forma analítica: este depende mucho de los frames que componen el vídeo. No es objeto de este trabajo fin de grado determinar exactamente cuál puede ser el error. Se utiliza a modo de ejemplo de ejecución de un tipo de aplicación que puede reducir su precisión para reducir el tiempo de ejecución.

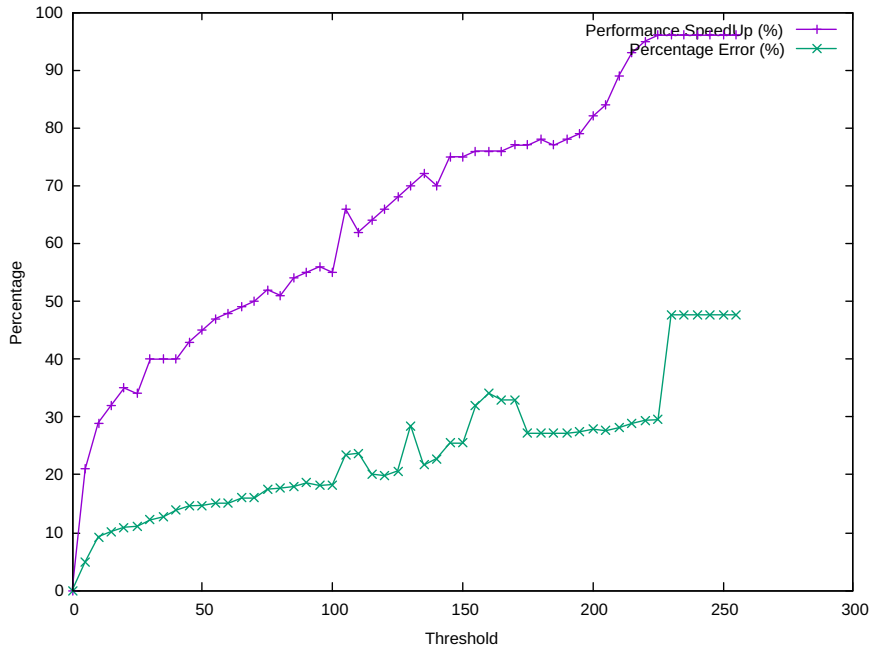


Figura 3.1: Porcentaje de mejora de las prestaciones de pymemo y portentaje de error añadido en pymemo en función del threshold

3.2. Infraestructura Computacional Distribuida

La infraestructura de este trabajo fin de grado se ha creado siguiendo una arquitectura de aplicaciones IoT, donde su procesado se ha distribuido tanto en *Cloud* como en *Edge*. Se ha planteado un esquema jerárquico en tres capas, como se muestra en la figura 3.2: la primera son los sensores IoT, unas cámaras de seguridad que se limitan a grabar y enviar el vídeo a su capa contigua. La siguiente capa corresponde al *Edge*, un clúster de *raspberries*, que recibirá la petición de tratamiento de vídeo. Finalmente, en la última capa se encuentra el centro de datos, un clúster formado por máquinas virtuales donde se considerará que los recursos son infinitos. Cada capa tiene un único punto de entrada, que denominaremos *gestor de peticiones*, el cual decide si la petición se trata en en el propio clúster o es enviada a un nivel superior.

Para simplificar el proyecto, y debido a los recursos disponibles nos centraremos únicamente en una rama, donde las cámaras son emuladas, el clúster *edge* esta formada por tres *raspberries* que están localizadas en un laboratorio del edificio Ada Byron de la universidad de Zaragoza. El clúster del centro de datos es un conjunto de máquinas virtuales en un servidor del mismo laboratorio.

Scheduling Para garantizar la escalabilidad, se escogió una estrategia de *scheduling* totalmente distribuida: no existe un estado global del sistema, sino que las peticiones al llegar a un clúster deciden si se ejecutan en el clúster o continúan al siguiente nivel de la jerarquía. Independientemente de estas decisiones de las aplicaciones, en cada clúster, existe un componente gestor de recursos que se encarga de facilitar las máquinas y, además, de gestionar el consumo energético. Toda la gestión del clúster se realiza desde el gestor de recursos. Este es consciente de todos los recursos de los que dispone, del estado del entorno (precio de la luz y error máximo permitido), y del estado del propio clúster (peticiones aceptadas, tiempos de ejecución y gasto energético).

Como consecuencia del *scheduling* la petición de procesamiento de un vídeo (metadatos) y el vídeo (dato) pueden seguir dos caminos alternativos, como muestra la Figura 3.3 siendo estos A1-A2 o B1-B2-B3. El camino A1-A2 se daría cuando hay disponibilidad el clúster formado por las *raspberries*, en dicho caso, la petición de IoT llega el gestor de peticiones de las *raspberries* (A1) y este aceptando la petición la ejecutará en la *raspberry* (A2). En caso de no disponer de recursos suficientes seguirá el flujo B1-B2-B3, en el cual tras recibir la solicitud de ejecución del IoT (B1) el gestor de peticiones detectará la imposibilidad, por lo que delegará la petición al gestor de peticiones del centro de datos (B2). Este a su vez comprobará la disponibilidad de recursos: en caso de tener suficientes ejecutará la solicitud (B3) y en caso contrario

esperará hasta tenerla.

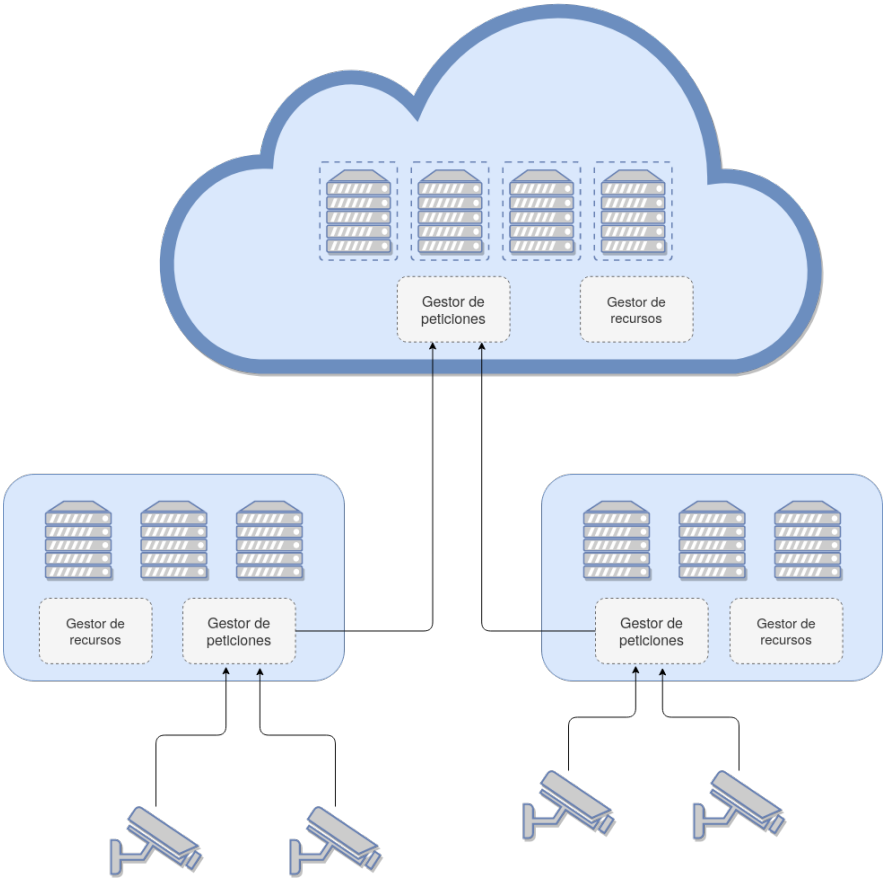


Figura 3.2: Diagrama estático - Arquitectura del sistema

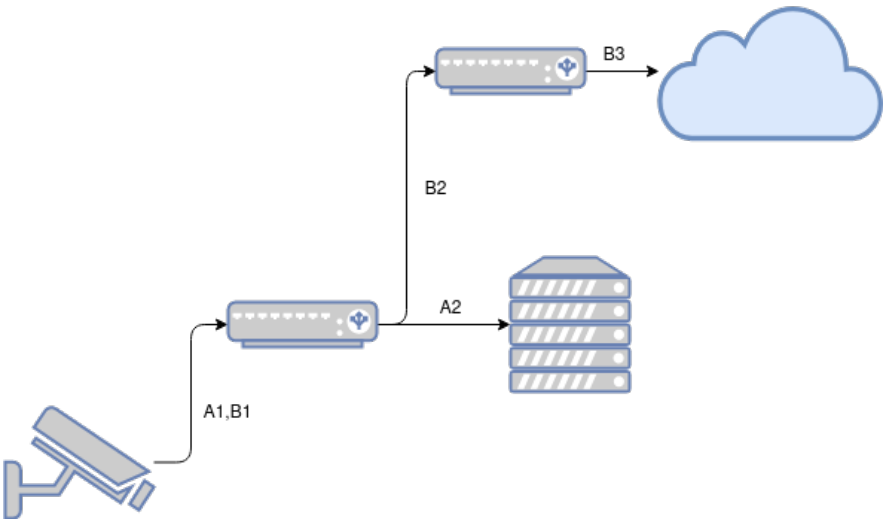


Figura 3.3: Diagrama dinámico - Arquitectura del sistema

Despliegue de pymemo La aplicación se desplegó en ambos tipos de clúster pero de forma distinta: en el clúster con las máquinas virtuales se desplegó con OpenFaas, en

este caso la modificación sobre pymemo fue mínima, simplemente se eliminó la llamada a la función main, y en el fichero de configuración de la aplicación de linea de comandos de OpenFaas se indicó cual era la función *main*, y la generación del contenedor a partir de una plantilla personalizada con todas las dependencias ya instaladas.

En el clúster formado a partir de *raspberries* no fue posible desplegar OpenFaas debido a la gran limitación de recursos computacionales. Una vez identificado el problema se trato de realizar una aplicación alternativa similar pero más ligera: esta idea se descartó dada su complejidad. Finalmente se optó por crear un servidor web que enlace una url con una invocación a pymemo, simulando así una aplicación faas. Este servidor web fue desplegado sobre un *deployment* de Kubernetes y expuesto por un servicio *NodePort*. El despliegue de ambos clústers están disponibles en un repositorio [3] de Github.

Se escogió un despliegue serverless frente a uno más tradicional por la flexibilidad que permitía. Cada elemento es muy aislado de los demás y su gestión es muy sencilla a través de orquestadores como Kubernetes. Un ejemplo de esta ventaja es la capacidad de autoescalar las funciones, mientras que en un despliegue monolítico tendría un servidor central encargado de diversas funciones, una función serverless únicamente es un contenedor con una función encapsulada, de esta forma podemos liberar o ocupar recursos de una forma mucho más precisa.

3.3. Calidad de Servicio para pymemo

Habitualmente, muchos trabajos establecen los parámetros de la calidad de servicio en torno al tiempo de ejecución de una tarea o al *throughput* (número de tareas procesadas por unidad de tiempo). Con la aparición del paradigma de la nube, muchas veces ambos parámetros se combinan con el coste económico.

En las aplicaciones en las que o para las que se generan datos continuamente y cuyo procesamiento tiene que hacerse en *streaming*, se ha propuesto que la calidad de servicio se mida respecto del *throughput* [23, 24]. Por tanto, para la definición de la calidad de servicio de pymemo se va a considerar una aproximación similar. En general, se pretende maximizar el *throughput* en pymemo, es decir, se pretende que el número de pymemos procesados sea máximo, haciendo el menor uso de recursos posible.

En un contexto como el actual, el precio de la energía puede fluctuar de una manera impredecible. Por ejemplo, supóngase el caso en el cual se dispone de placas solares para autoconsumo: cuando se dan condiciones solares adecuadas, se puede alimentar un clúster de computadores. En ese caso, el coste puede considerarse prácticamente

[3] `https://github.com/arejula27/TFG-deployment`

cero. Sin embargo, si las condiciones para utilizar las placas solares no son adecuadas y se tuviera que adquirir la energía a precio de mercado, el precio podría ser muy elevado. En este contexto, este trabajo fin de grado define la calidad de servicio para una aplicación del tipo de pymemo. Se pretende, por lo tanto, que la calidad de servicio pueda definirse de forma diferente en función del contexto energético. Cuando el coste de la energía esté próximo a cero, el sistema podrá realizar los cálculos con la mayor precisión posible, maximizando el *throughput* y consumiendo toda la energía necesaria. Sin embargo, cuando el coste de la energía sea elevado, el sistema podrá degradar la precisión de los cálculos e incluso parar completamente la ejecución y apagar las máquinas si fuera necesario. La Tabla 3.1 recoge estas ideas y propone cuatro escenarios diferentes. Cuando la energía sea barata, el sistema podrá operar sin restricciones energéticas y proporcionar la máxima precisión. Por contra, se proponen tres estrategias distintas que pueden permitir limitar el consumo energético cuando la energía sea cara. La estrategia b) consiste en que el coste económico de la potencia consumida (potencia instantánea o bien potencia media) no supere un máximo C. La estrategia c) consiste en que el coste económico de la energía para ejecutar un pymemo no supere un coste máximo C, y la estrategia d) consiste en que la potencia consumida no supere un máximo dado P.

No se pretende, por tanto, minimizar el consumo energético, sino no superar un máximo de consumo energético y tratar de maximizar el *throughput* habida cuenta de las restricciones energéticas. Cuando se introduzcan las restricciones energéticas, el sistema podrá optar por dos vías para reducir el consumo, que podrán incluso combinarse: reducir la precisión de los cálculos o introduciendo un tiempo de espera. Por lo tanto, en términos generales, muy probablemente el *throughput* podrá verse impactado, o bien lo hará la calidad de los cálculos. Todos estos escenarios pueden tener sentido en distintos contextos de escasez energética y proporcionarán al sistema distintas posibilidades de adaptación, tal y como se discutirá más adelante.

Tabla 3.1: Calidad de Servicio para un contexto fluctuante entre escasez y abundancia de energía

Energía a coste 0	a) Sin restricciones energéticas Máxima precisión
Energía cara	b) El coste económico de la potencia consumida <C Precisión degradada
	c) El coste económico de la energía para 1 pymemo <C Precisión degradada
	d) La potencia consumida <P

Capítulo 4

Modelos de Consumo Energéticos

4.1. Factores que Contribuyen al Consumo Energético Computacional

En el trabajo se han considerado aquellos factores de las distintas capas del sistema que afectan al consumo energético de un sistema computacional. Cada clúster tiene unas características propias e intrínsecas, por lo que es necesario desarrollar un modelo específico para cada uno. Además de que los modelos obtenido han sido elaborados de forma experimental en entornos controlados, ejecutando únicamente la aplicación *pymemo*.

A la hora de crear el modelo se deberán especificar los factores de cada capa que tienen un impacto en el consumo. La forma de hallar dichos factores recae sobre la formula de la potencia (ecuación 4.1). Esta depende de dos variables, la intensidad y el voltaje, siendo este último constante en cada máquina y especificado por la fuente de alimentación. En algunos casos se puede bajar el voltaje de los procesadores, pero no siempre es una opción recomendable. Lo que sí se puede hacer es reducir la frecuencia de la CPU y reduciendo así la intensidad. Además, para reducir la intensidad también se puede disminuir la ocupación de la CPU.

$$P = V * I \tag{4.1}$$

También es importante remarcar la implicación del tiempo en el sistema, el coste energético total depende de la potencia y del tiempo total de ejecución (ecuación 4.2).Al reducir la potencia mediante la disminución de la frecuencia, los tiempos de ejecución aumentarán, por lo que es fundamental que la disminución de la potencia sea en mayor proporción que el aumento del tiempo

$$E = P * t \tag{4.2}$$

En la Tabla 4.1 se muestran los factores que afectan a la intensidad de la máquina en las distintas capas.

Tabla 4.1: Factores que afectan a la intensidad de la máquina en sus distintas capas

Capa	**Factores**
Máquinas físicas	Frecuencia CPU
Scheduling / Gestión recursos	Ocupación máxima permitida
Aplicación	Precisión de los calculos

Máquinas físicas La primera capa corresponde con las propias máquinas físicas. Al modificar la frecuencia del procesador bajaremos la intensidad de la máquina, por ende requerirá menor potencia y utilizando menos energía.

Scheduling / Gestor de recursos En esta capa se limitará el número de máquinas activas de cada clúster por cuestiones energéticas. Hay que distinguir si el clúster solo se compone de máquinas físicas (por ejemplo, en este caso el clúster de *raspberry pis*) o si el clúster se compone, además, de máquinas virtuales. Cuando múltiples máquinas virtuales se ejecutan en un servidor físico concurrentemente, reduciendo el número de máquinas virtuales que coexisten en el mismo servidor físico, se reduce el consumo energético del servidor físico. El número de máquinas virtuales que coexisten en la misma máquina física no solo afecta al consumo energético, sino que además influye gravemente en el tiempo de ejecución, debido al efecto de la interferencia. Este efecto se produce por el hecho de que hay recursos en el hardware (como el bus de acceso a la memoria) que están compartidos por todas las máquinas virtuales. La jerarquía de memoria también puede tener efectos significativos en las prestaciones.

Aplicación El último nivel a tener en cuenta es el de aplicación. Para ello deberemos contar con una aplicación capaz de modificar su comportamiento, pudiendo aumentar su ahorro energético a cambio de disminuir su precisión en los cálculos. La idea es aplicar aproximación para minimizar la cantidad de cómputo requerida por la aplicación, y de esta forma reducir la intensidad. En este proyecto hemos usado una aplicación de análisis de vídeos de cámaras de vigilancia, la cual obtiene el ahorro de guardar en memoria cache resultados precalculados: en este caso el ahorro sufre una gran influencia por el vídeo que analice, ya que depende de la similitud entre los frames y no solo depende de la propia aplicación. Para obtener los datos del modelo se realizaron pruebas modificando el valor del umbral (*threshold*) en 4 valores: 0,50,10,150, siendo 0 no utilizar memorización y 150 utilizarla en gran medida, pero con un error aceptable (20 % como muestra la figura 3.1).

El uso del error no influye únicamente en la potencia del sistema, sino que también reduce los de tiempos de ejecución, disminuyendo así el coste energético y aumentando el número de peticiones por segundo que puede aceptar el sistema. En la gráfica 4.11

se aprecia como la tendencia es reducir la energía aumentando el error, y en la gráfica 4.7 se muestra que a mayor error mayor *throughput*. En la tabla 4.2, se ve claramente el efecto del error en los tiempos de ejecución.

4.2. Modelo Energético del centro de datos

El modelo energético del centro de datos se obtuvo sobre un servidor localizado en un laboratorio del edificio Ada Byron de la universidad de Zaragoza. Este ordenador tiene un procesador Intel(R) Xeon(R) Bronze 3106 CPU 1.70GHz, el cual tiene una arquitectura x86_64, 16 CPUs, un hilo por núcleo, una memoria RAM con 125 GB y 9.4 GB de memoria swap. Su rango de frecuencias es de 0.8GHz a 1.70GHz. El sistema operativo utilizado es 18.04.6 LTS (GNU/Linux 5.3.0-7648-generic). En él se desplegaron máquinas virtuales mediante la herramienta Vagrant con 3 GB de memoria cada una y 3 núcleos en el procesador. En las máquinas virtuales se ejecutó Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-171-generic) como sistema operativo.

Con el conjunto de máquinas virtuales se creó un clúster de Kubernetes, donde se ejecuta OpenFaas, dando así disponibilidad a *pymemo* en un modelo *serverless*.

Sobre este clúster virtual se creó el modelo energético para la máquina física a partir del método de barrido de parámetros, es decir, se ejecutó *pymemo* variando todas las posibles combinaciones de la terna *threshold*, ocupación y frecuencia, obteniendo así sus métricas asociadas como la potencia y tiempo de ejecución. Para eliminar resultados espurios el experimento se repitió varias veces.

Los experimentos revelaron los siguientes observaciones. Cuando se disminuye la frecuencia, se observa que la potencia consumida siempre es menor, por lo que se producirá un menor gasto energético por segundo, como se observa en las Figuras 4.1, 4.2, 4.3 y 4.4. Además, al disminuir la frecuencia, hay también una tendencia a que aumente el tiempo de ejecución, por ello el *throughput* disminuye. Este efecto puede verse en las Figuras 4.5, 4.6, 4.7 y 4.8. También se observa el fenómeno de la interferencia en las prestaciones. Cuando se usan máquinas físicas, incrementos en el número de máquinas físicas utilizadas tiene un impacto proporcional en el *throughput*. En este caso, al incrementar el número de máquinas virtuales, en la misma máquina

Threshold	Tiempo (s)
0	360.943
50	277.755
100	181.626
150	141.941

Tabla 4.2: Tiempos de ejecución con frecuencia 1,2 GHz y 3 ejecuciones simultaneas

físcia, el *throughput* no aumenta en la misma proporción, sino que aumenta menos, llegando a disminuir en algunos casos, como puede verse en la diferencia de *throughput* con 1,6GHz y *threshold 0* en las Figuras 4.7 y 4.8. Típicamente, cuando se plantean modelos teóricos para optimizar el coste energético o las prestaciones, en la mayoría de ellos se asume que el tiempo de ejecución es una constante y se ignoran los efectos que tiene la interferencia en las prestaciones.

Otro aspecto importante es que, al variar la frecuencia, el tiempo de ejecución (o bien el *throughput*) y la potencia, no varían en la misma proporción. Por lo tanto, puede haber casos en los que, por ejemplo, a una menor frecuencia, se tenga un mayor tiempo de ejecución, pero con un menor consumo energético total. Por ejemplo, en la Figura 4.9, se observa que en la configuración con mayor frecuencia de 1,7GHz y *threshold* 0, la ejecución de 2 *pymemos* en paralelo tiene un coste energético total de 14.367 J y un *throughput* 0,0085 *pymemos* / s. Mientras que si observamos el coste energético total para la frecuencia 1,6 GHz y *threshold* 0 es de 12.550 J y el *throughput* 0.0084 *pymemos* / s, lo que supone un ahorro energético de un 12 % y una disminución del *throughput* del 1 %. Esto es, algunas veces, reduciendo la frecuencia de la CPU, se puede reducir el consumo energético de la ejecución de la carga de trabajo, incrementando un poco el tiempo de ejecución (disminuyendo el *throughput*). Sin embargo, no siempre es así y, además, la tendencia general es que al seguir disminuyendo la frecuencia, se llega a un punto en el que aumenta mucho más el tiempo de ejecución y, por tanto, ya no solo no hay ahorro energético, sino que incluso hay un mayor consumo. Por ejemplo, en ese mismo caso de 2 *pymemos* en paralelo, si reducimos la frecuencia a 0,9 GHz, el coste energético es 16.992 J, mucho mayor, y el *throughput*, mucho menor, 0.0055 *pymemos*/s.

Este mismo efecto de la variación en el consumo energético al reducir la frecuencia puede verse en las Figuras 4.9, 4.10, 4.11 y 4.12. En estas figuras se puede apreciar que una menor frecuencia la tendencia es que se reduzca el consumo energético, pero con una mayor reducción de la frecuencia, al aumentar el tiempo total de ejecución, termina por aumentar el gasto total energético.

Por otra parte, es también significativo el impacto que tiene la reducción de la precisión en la ejecución de *pymemo* en el tiempo de ejecución, como se aprecia en las Figuras 4.5, 4.6, 4.7 y 4.8 y, en consecuencia, en el coste energético total de las Figuras 4.9, 4.10, 4.11 y 4.12.

El modelo energético consistirá en un conjunto de elementos formados por un valor de *throughput*, la potencia asociada y la configuración que lo garantiza (terna ocupación, frecuencia, *threshold*).

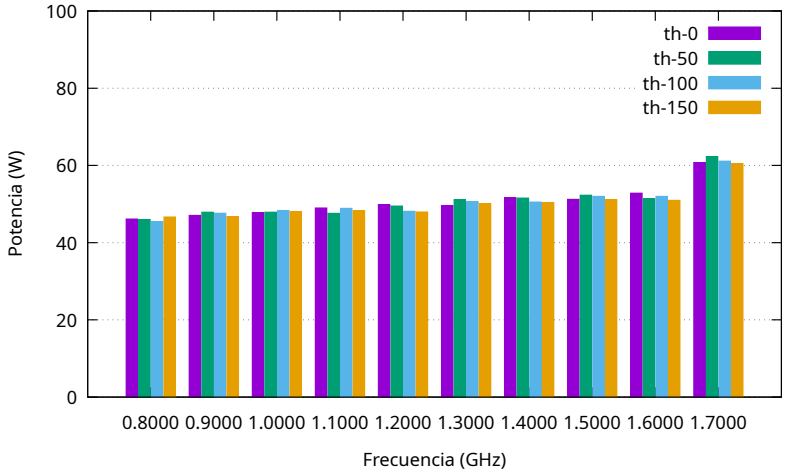


Figura 4.1: Evolución de la potencia respecto a la frecuencia y threshold con 2 ejecuciones paralelas de pymemo

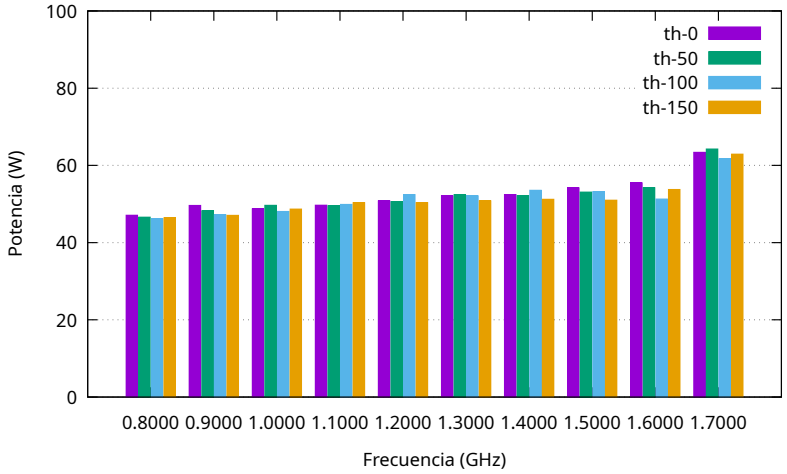


Figura 4.2: Evolución de la potencia respecto a la frecuencia y threshold con 3 ejecuciones paralelas de pymemo

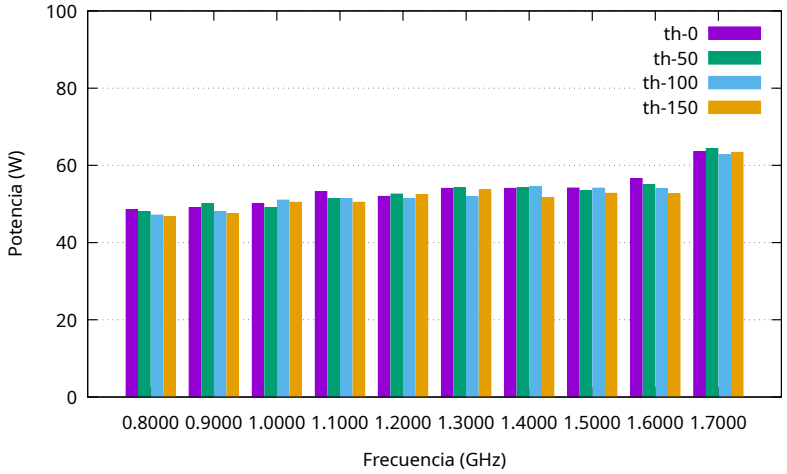


Figura 4.3: Evolución de la potencia respecto a la frecuencia y threshold con 4 ejecuciones paralelas de pymemo

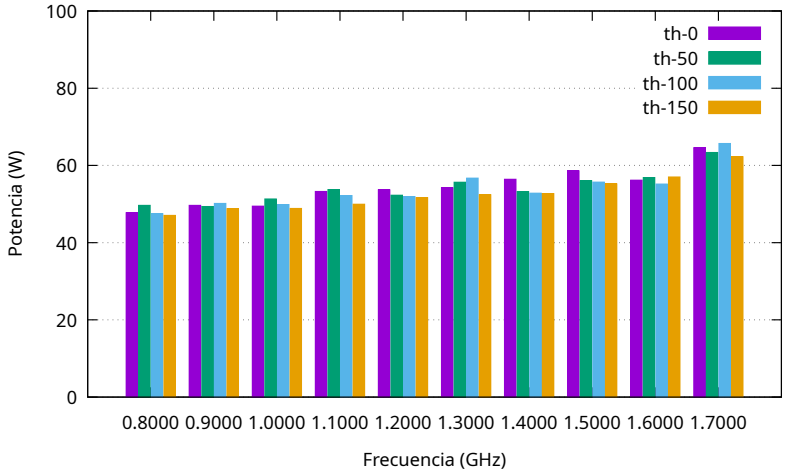


Figura 4.4: Evolución de la potencia respecto a la frecuencia y threshold con 5 ejecuciones paralelas de pymemo

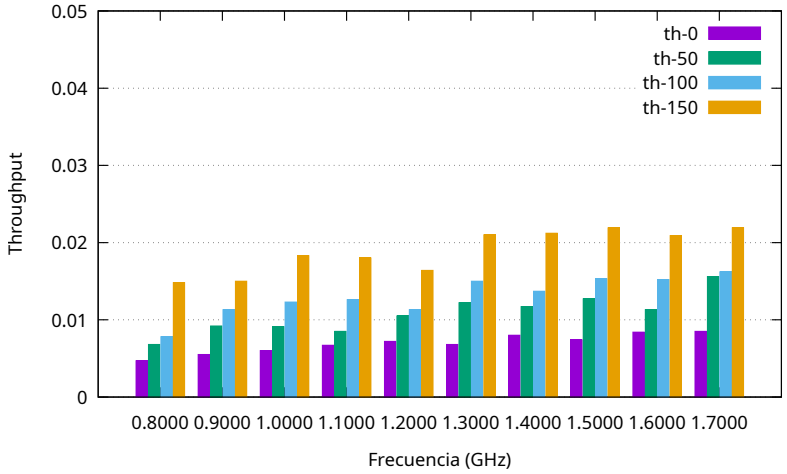


Figura 4.5: Evolución del throughput respecto a la frecuencia y threshold con 2 ejecuciones paralelas de pymemo

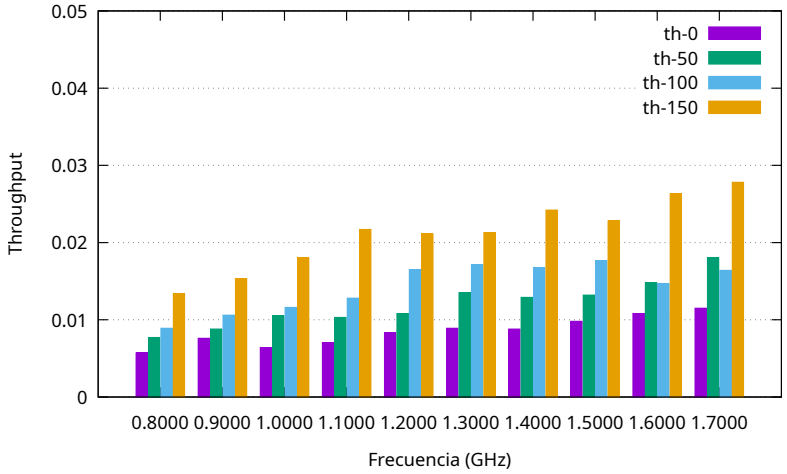


Figura 4.6: Evolución del throughput respecto a la frecuencia y threshold con 3 ejecuciones paralelas de pymemo

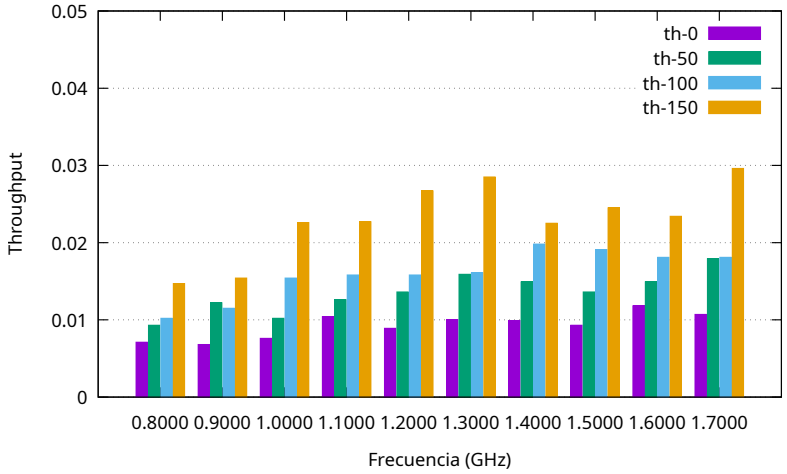


Figura 4.7: Evolución del throughput respecto a la frecuencia y threshold con 4 ejecuciones paralelas de pymemo

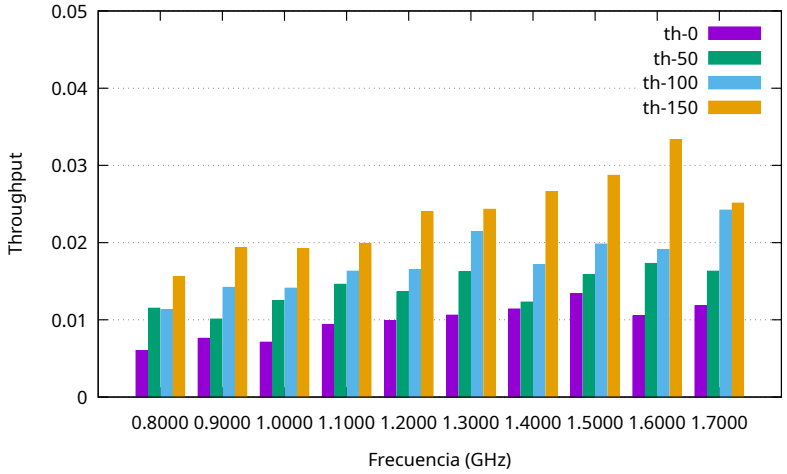


Figura 4.8: Evolución del throughput respecto a la frecuencia y threshold con 5 ejecuciones paralelas de pymemo

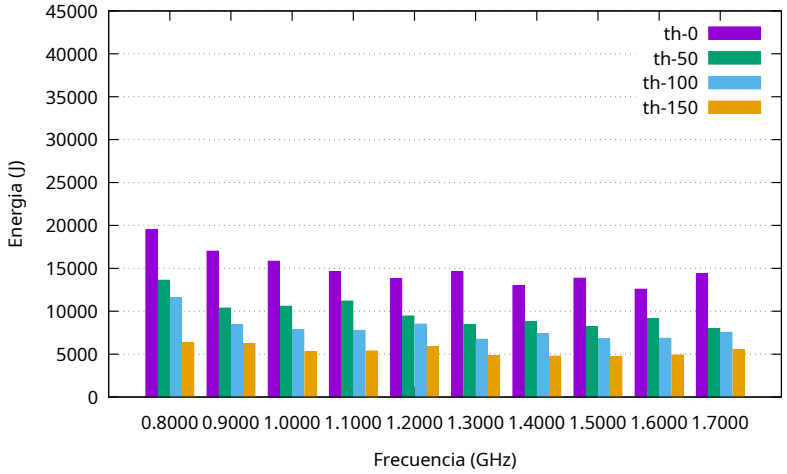


Figura 4.9: Evolución de la energía respecto a la frecuencia y threshold con 2 ejecuciones paralelas de pymemo

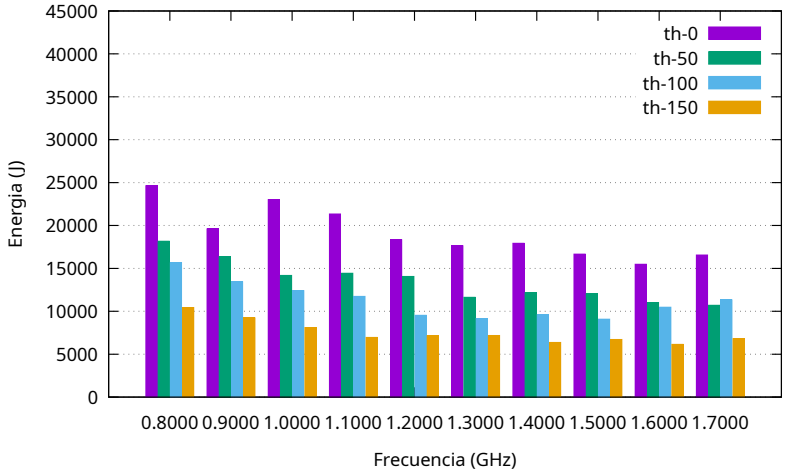


Figura 4.10: Evolución de la energía respecto a la frecuencia y threshold con 3 ejecuciones paralelas de pymemo

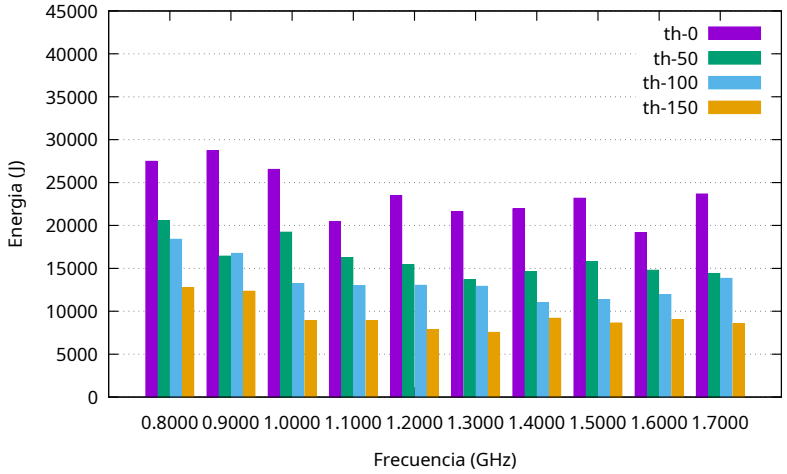


Figura 4.11: Evolución de la energía respecto a la frecuencia y threshold con 4 ejecuciones paralelas de pymemo

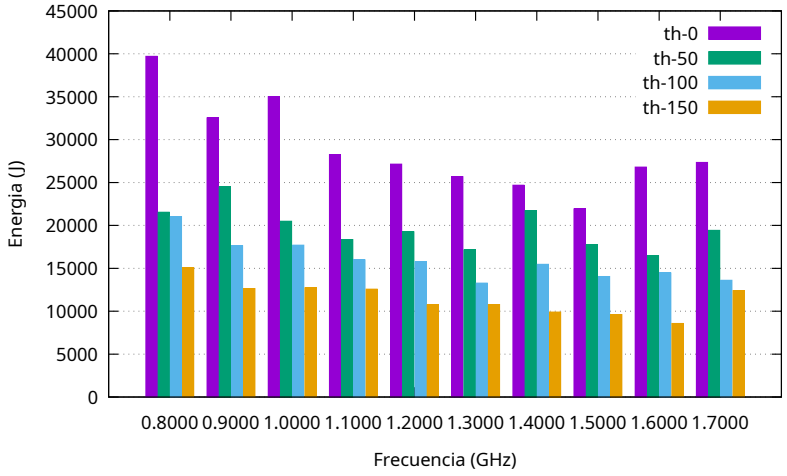


Figura 4.12: Evolución de la energía respecto a la frecuencia y threshold con 5 ejecuciones paralelas de pymemo

4.3. Modelo energético edge

El modelo del *edge* se ha obtenido también mediante el barrido de parámetros. Dicho barrido se ha realizado en una *raspberry* pi 3 B+ localizada en el laboratorio del edificio Ada Byron de la universidad de Zaragoza. Las características de la máquina son una CPU Cortex-A53 con cuatro núcleos, un hilo por núcleo, una frecuencia de 0,6 GHz a 1,2 GHz, una memoria RAM de 973Mi y 99Mi de memoria *swap*. En un principio se trató de montar un clúster Kubernetes entre varias *raspberries* y desplegar ahí OpenFaas, pero debido a las pocos recursos de las máquinas fue imposible lograr desplegar este último. Por ello, se utilizó una única *raspberry* para medir, en la cual se ejecuto un servidor http a través el cual se podía invocar *pymemo*. Otro problema es la medición de la potencia. Como se explica en la sección 2.3, las *raspberries* no permiten obtener datos de la potencia por software, así que hubo que utilizar un multímetro y hacer el proceso de forma manual.

En un principio se realizaron medidas a la menor frecuencia y sin error, pero debido a los altos tiempos de ejecución (en torno a la hora) se decidió limitar las frecuencias a las dos más altas, donde el tiempo de ejecución se reducía a la mitad. Como se ve en la gráfica de la figura 4.14, la frecuencia tiene poco impacto en la potencia, pudiendo considerarse que ambas frecuencias generan la misma potencia. En la gráfica de la figura 4.13 se aprecia la influencia de la frecuencia en la energía total de una ejecución de *pymemo*, siendo esta menor a 1.1 GHz. En cuanto a las peticiones que ejecuta por segundo en la *raspberry*, se ha determinado que en los casos con *threshold* 0,50 y 100 son mayores con frecuencia igual a 1,1GHz. En cuanto a las peticiones ejecutas por segundo el mejor valor se obtiene con la configuración 1,1GHz a *threshold* 100, siendo la segunda mejor opción la otra frecuencia con el mismo *threshold*. En el caso de la frecuencia 1.2 GHz, tiene el mismo *throughput* para los *threshold* 20 y 50. La frecuencia 1.1Ghz tiene el mayor *throughput* pero también el menor.

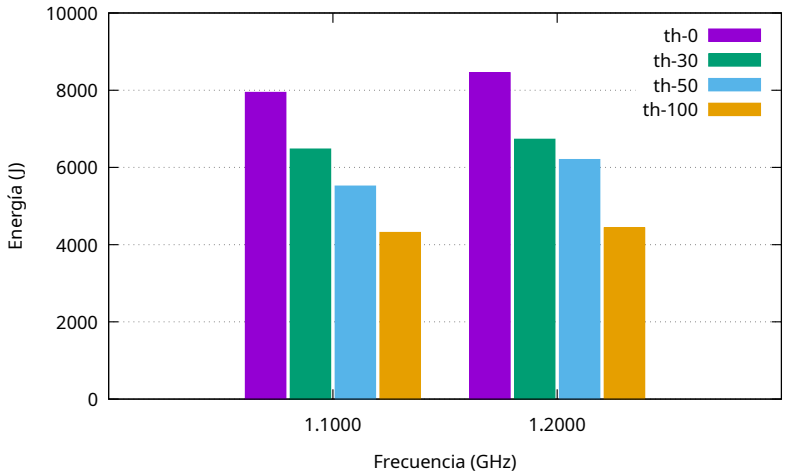


Figura 4.13: Evolución de la energía respecto a la frecuencia y threshold en una raspberry

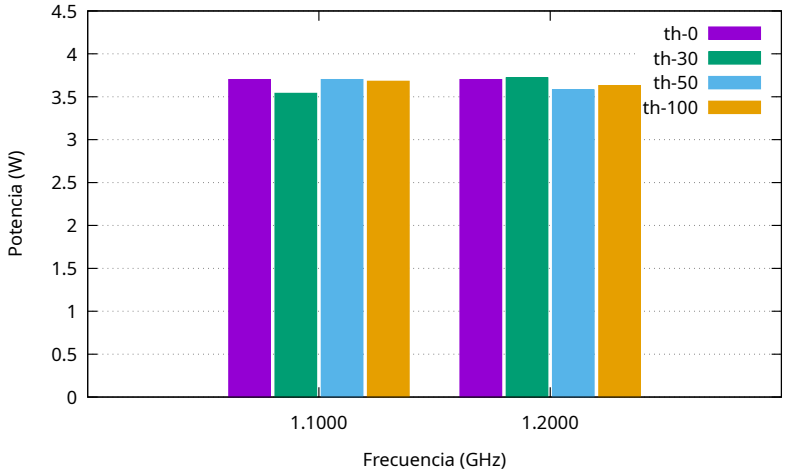


Figura 4.14: Evolución de la energía respecto a la frecuencia y threshold en una raspberry

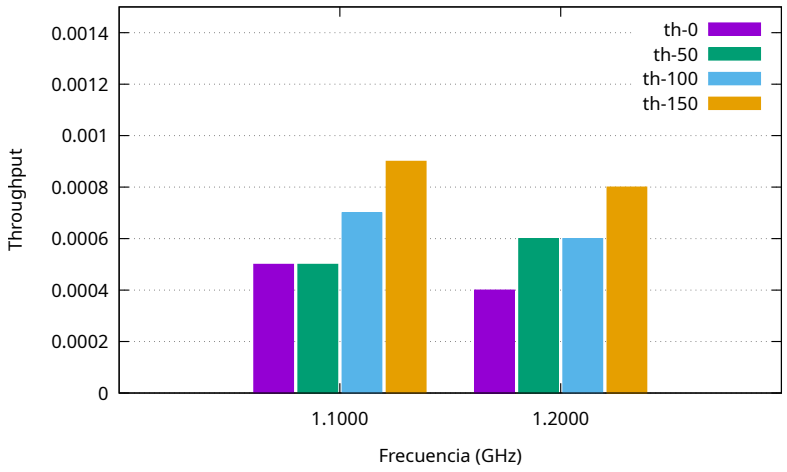


Figura 4.15: Evolución de la energía respecto a la frecuencia y threshold en una raspberry

4.3.1. Modelo Energético de la Red de Comunicación

Un modelo de consumo energético para la transmisión de datos en la red de comunicación está descrito en [25]. En él se observa, cómo la transmisión Wireless desde una *raspberry pi*, consume entre 1W y 1,4W, mientras que el consumo debido a la transmisión a través de Ethernet consume entre 0,3W y 0,4W. Sin embargo, experimentalmente pudimos comprobar cómo para transmitir los vídeos de las *raspberry pis* al centro de datos, el sistema tiene que acceder al disco de la *raspberry*, esto es, a una tarjeta de memoria. Eso hace que la potencia total necesaria para realizar la transmisión de un vídeo a través de Ethernet se eleve hasta unos 4W. Es, por tanto, importante considerar que cuando se toma la decisión de transmitir al siguiente nodo, se incurre en un coste energético que puede no ser despreciable.

4.4. Algoritmo de búsqueda

El algoritmo escogido fue el algoritmo voraz, este es una búsqueda que obtiene una solución correcta cumpliendo un conjunto de restricciones preestablecidas. Este algoritmo nos garantiza encontrar una solución en caso de que exista, pero no garantiza que sea la óptima. Además, la simplicidad del algoritmo permite no añadir un gran sobre coste, y con ello un mayor gasto indeseado como otras búsquedas que recorran todo el espacio de soluciones. Este algoritmo se describe con las instrucciones del Algoritmo 1.

```
Data: C
Result: S
Greedy( Conjunto de candidatos C): solución S;
S = ∅;
while C no sea vacio y S no sea solucion do
    x = seleccionar(C) ;
    C = C - {x};
    if x cumple restricciones then
        S={x};
    end
end
if S ≠ ∅ then
    return S;
else
    return "no hay solución";
end
```

Algorithm 1: Algoritmo Voraz que busca la configuración energética apropiada

El algoritmo parte de un conjunto de posibles soluciones C y un conjunto final de soluciones S. A partir de ahí, se realizará una iteración sobre el conjunto C, extrayendo

un elemento cada vez. Si el elemento extraído es una solución correcta (cumple las restricciones), la búsqueda finalizará y dará por solución el último elemento extraído y único elemento de S. En caso de que ningún elemento cumpla las restricciones, el algoritmo determinará que no existe solución. Se debe remarcar que una vez explorado un elemento del conjunto C, este se descarta, no pudiendo volver a ser evaluado.

En este proyecto el conjunto de soluciones es una lista de configuraciones del sistema, ordenada por *throughput*, de mayor a menor. Cada configuración esta formada por la terna ocupación, frecuencia y *threshold*, teniendo asociada además la potencia que dicha configuración genera en el sistema (nótese que a partir de la potencia y el resto de parámetros se puede calcular la energía total necesaria). La búsqueda consiste en recorrer la lista ordenada de soluciones, hasta que haya una que satisfaga todos los requisitos: ii) que se satisfagan las restricciones energéticas y ii) que se satisfagan los requisitos de la aplicación, por ejemplo, en cuanto a la precisión de los cálculos. Como salida el algoritmo proporciona la configuración del sistema que satisfaciendo los requisitos de energía y de la aplicación, ofrezca un mayor *throughput*. En caso de que ningún elemento cumpla las restricciones, el algoritmo determinará que no existe solución.

4.5. Gestor de recursos consciente del gasto energético

En cada clúster hay un gestor de recursos como se explica en la sección 3.2. Como gestor de recursos, se encarga de configurar y proporcionar los recursos adecuados para la carga de trabajo del sistema. Su ciclo de vida, en continua ejecución, sigue un esquema MAPE-K [26]: monitorización, análisis, planificación y ejecución de la gestión de recursos para poder ahorrar energía.

4.5.1. Gestor de recursos

El gestor de recursos es el núcleo de nuestra aplicación como muestra la figura 4.16 guarda el estado tanto del entorno (potencia máxima permitida o error máximo) como la situación del sistema (*througput*, ocupación, frecuencia, etc.).

Monitorización La primera fase corresponde a la monitorización, actualizando los valores de la potencia media de las máquinas, el tiempo medio de ejecución y la ocupación actual. Los valores se actualizan de forma reactiva, conforme los distintos

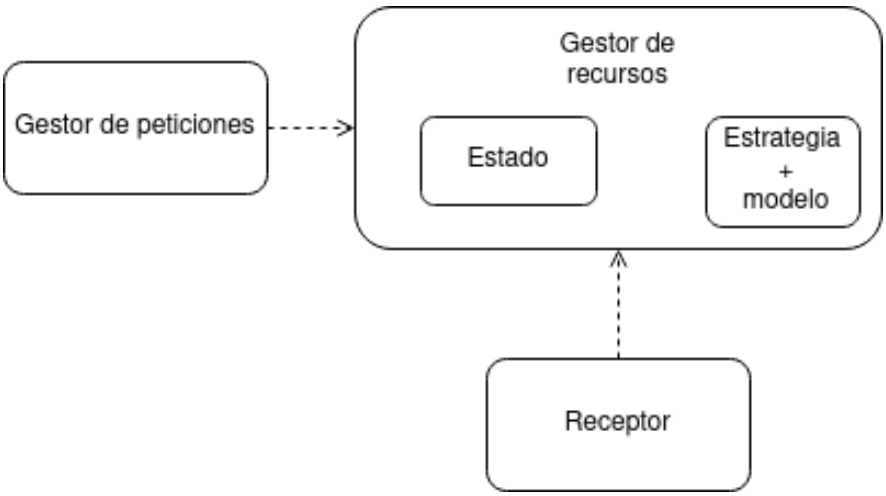


Figura 4.16: Diagrama arquitectural del gateway

receptores envían los datos: por ello, en el bucle MAPE-K se realiza un *snapshot* del estado antes de continuar a la siguiente fase, de tal forma que si un valor cambia, no será tenido en cuenta dicha modificación hasta la siguiente iteración, evitando así inconsistencias.

Análisis y planificación El análisis y la planificación los realizamos de forma conjunta y es el Algoritmo de búsqueda 1 el encargado de realizalos. Una vez encontrada una solución, el sistema obtiene la nueva configuración energética asociada a esta.

Ejecución A partir de la decisión tomada se deberán tomar una serie de acciones configurando el sistema con los parámetros devueltos por la fase anterior.

4.5.2. Receptores

Los componentes encargados de obtener métricas para actualizar el estado son los receptores, estos se encargaran de analizar el entorno y de comunicarlos al gestor de recursos los cambios que detecten. Actualmente existen dos receptores para el gestor de recursos, el que monitoriza la potencia del sistema, creando una abstracción de la herramienta porwerstat, y el encargado de monitorizar el número de peticiones y tiempo de ejecución de estas, integrado en el gestor de peticiones. Mientras que el receptor de potencia obtiene las métricas en periodos regulares, el receptor de peticiones las toma de forma reactiva.

4.5.3. Gestor de peticiones que llegan al sistema

Las peticiones de ejecución de *pymemo* llegan a los gestores de peticiones de cada clúster. Este módulo decide si se acepta una petición en función de la configuración del

clúster escogida por el gestor de recursos.

El gestor de peticiones es un servidor http donde se reciben las peticiones. Cuando llega una petición se encola y, en caso de haber disponibilidad, se trata la primera petición de la cola, pudiendo ser o no la recién llegada. Para tratarla, se realiza una llamada http al servicio de OpenFaas del clúster local. En caso de no haber disponibilidad, se envía a un clúster de nivel superior o se espera a que haya disponibilidad si no hay más niveles. Tras obtener el resultado de la ejecución se comunican métricas como tiempo de ejecución necesario para la petición y el número de petición que era. Una vez respondida la petición al cliente el gestor de peticiones comprobará de nuevo la disponibilidad del sistema y en caso de haber peticiones encoladas pasarán a ejecutarse, respetando siempre las restricciones.

Capítulo 5

Validación Experimental

5.1. Configuración de los Experimentos

Los experimentos se han realizado únicamente en el servidor dado que este permite ejecutar simultáneamente más de un *pymemo*, de esta forma en los resultados se podrá ver el efecto de la interferencia. La máquina utilizada ha sido la misma que para realiza el modelo: un procesador Intel(R) Xeon(R) Bronze 3106 CPU 1.70GHz, el cual tiene una arquitectura x86_64, 16 CPUs, un hilo por núcleo, una memoria RAM con 125 GB y 9.4 GB de memoria swap. Su rango de frecuencias es de 0.8GHz a 1.70GHz. El sistema operativo utilizado ha sido 18.04.6 LTS (GNU/Linux 5.3.0-7648-generic). En él se desplegaron cuatro máquinas virtuales mediante la herramienta Vagrant con 3 GB de memoria cada una y 3 núcleos en el procesador. En las máquinas virtuales se ejecutó Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-171-generic) como sistema operativo.

Para validar el modelo se diseñaron cuatro experimentos, teniendo todos ellos la misma entrada: una serie de peticiones de ejecución de pymemo, con un vídeo, a ritmo regular (primero tres iniciales y luego una nueva cada 40 segundos), en media $\lambda = 0{,}025$ tareas / s. Dos de los experimentos, tuvieron como restricción energética la potencia, en ellos se empezó en 61 vatios, de tal forma que todas las configuraciones cumplieran las restricciones, escogiendo así la de mayor *throughput*. Cada 4 minutos la potencia disminuyó en 3 vatios, hasta llegar que alcanzó un valor suficientemente inferior donde ninguna configuración sería posible, llevando al sistema a su paralización. Tras pasar 4 minutos a esta potencia mínima volvería a subir hasta los 61 vatios al mismo ritmo. Además, el *threshold* en cada experimento puede tomar un valor máximo de cero, en uno de los dos experimentos, y el otro de 150. El tiempo total de cada experimento, en este caso, es de 52 minutos cada uno.

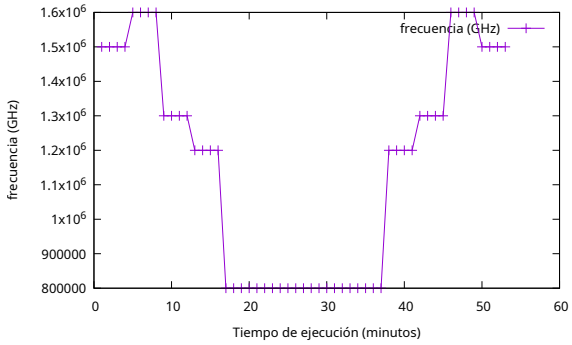
Los dos experimentos restantes, como restricción energética, tuvieron el coste energético máximo para la ejecución de un pymemo. En estos casos, el coste máximo por pymemo de 10000 J, una energía superior a todos los elementos del modelo.

Posteriormente, cada cuatro minutos, se decrementando en 1000 J. Además, el *threshold* en cada experimento puede tomar un valor máximo de cero, en uno de los dos experimentos, y el otro de 150. El tiempo total de cada experimento, en este caso, es de 35 minutos cada uno.

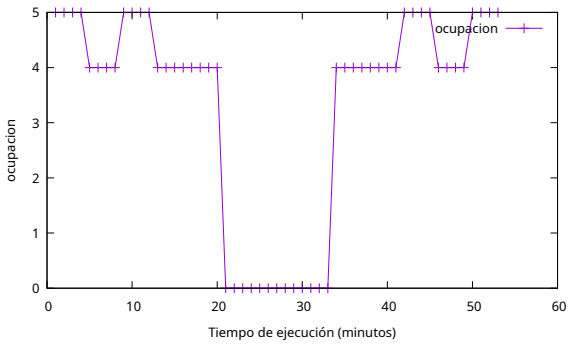
5.2. Experimentos

5.2.1. Restricción *threshold* 0 y máxima potencia

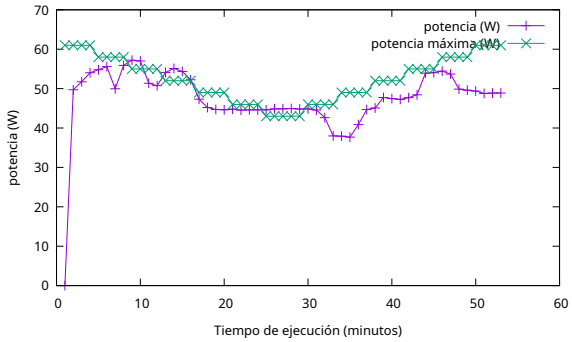
La Figura 5.1 muestra todas las métricas obtenidas durante este experimento. El *threshold* puede valer como máximo cero. El sistema cumplió la restricción del *threshold* en todo momento como muestra la Figura 5.1e. Cabe destacar la primera decisión, donde el sistema decidió disminuir la ocupación máxima a cambio de aumentar la frecuencia (Figuras 5.1a y 5.1b), mientras que en el resto de casos donde debe disminuir la potencia siempre disminuyó la frecuencia, utilizando 1,3 GHz, 1,2 GHz y 0,8 GHz, para finalmente suspender el sistema en el minuto 22, puesto que ninguna configuración podía ejecutarse con una menor potencia que la máxima permitida. Tanto la gráfica de la frecuencia (Figura 5.1a) como de la ocupación máxima (Figura 5.1b) son simétricas. La gráfica de la Figura 5.1d muestra la evolución del *throughput*, donde se aprecia que este se mantuvo alto hasta suspender el sistema, momento en el cual tuvo un gran decremento, a continuación se fue recuperando, aunque el tiempo medio de ejecución de un *pymemo* es de alrededor de 7 minutos. Finalmente, la gráfica 5.1c muestra la evolución de la *potencia medida real* (morado) y la potencia máxima (verde). Se observa claramente como el sistema fue disminuyendo su potencia conforme la máxima disminuyó: es cierto que se nota un retraso en la adaptación. Al final de esta gráfica se ve cómo decrementó la potencia, esto se debe a que se dejaron de enviar peticiones, y únicamente se estaban ejecutando las que estaban encoladas hasta acabarse.



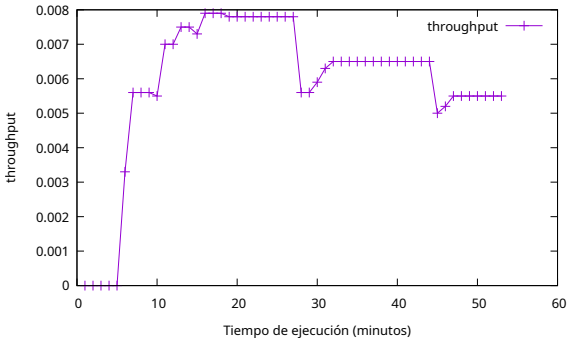
(a) Frecuencia



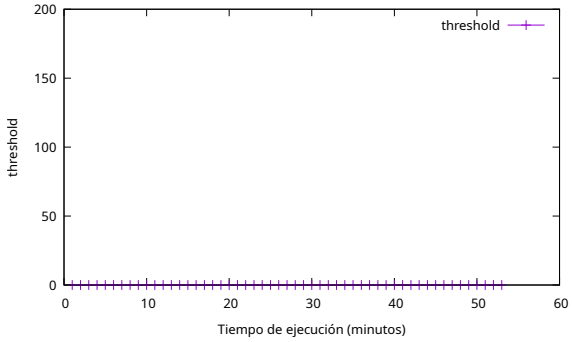
(b) Ocupación máxima (en pymemos)



(c) Potencia (W)



(d) Throughtput (pymemos / s)

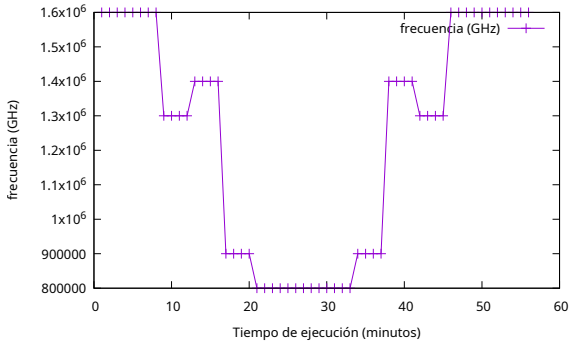


(e) Threshold

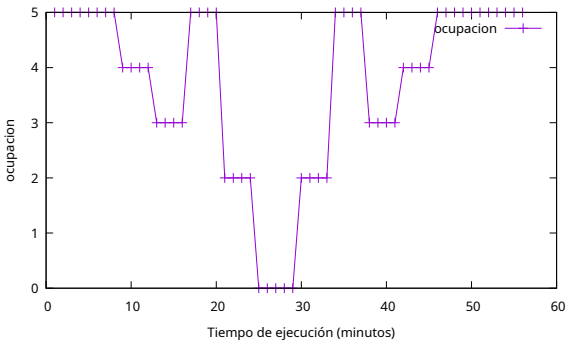
Figura 5.1: Cronograma de evolución del experimento 1 en el tiempo, threshold 0 y potencia limitada

5.2.2. Restricción *threshold* 150 y máxima potencia

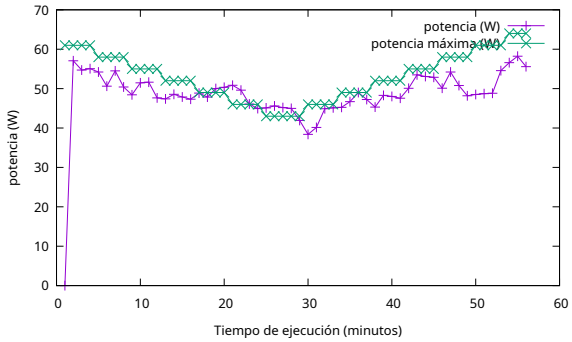
La Figura 5.2 muestra todas las métricas obtenidas durante este experimento. En este experimento se habilitaron todos los valores de *threshold* disponibles en el modelo: 0, 50, 100 y 150. En la Figura 5.2e se muestra cómo siempre priorizó el máximo *threshold*, con excepción del minuto 19, donde este lo decrementó a 100, disminuyendo al mismo tiempo la ocupación a dos pymemos simultáneos cómo máximo (Figura 5.2b). En cuanto al *throughput*, se dio una situación similar al experimento con *threshold* 0: se encuentra alto hasta que decayó fuertemente al suspender el sistema y aparecer largos tiempos de espera. En este caso, al tener un *threshold* más alto, los tiempos de ejecución de *pymemo* fueron menores respecto de un *pymemo* con *threshold* cero, por lo que las peticiones se ejecutan más rápidamente. Esto se observa claramente porque el *threshold* vuelve a aumentar poco a poco, habiéndose liberado la cola y volviendo a ejecutarse bajo demanda.



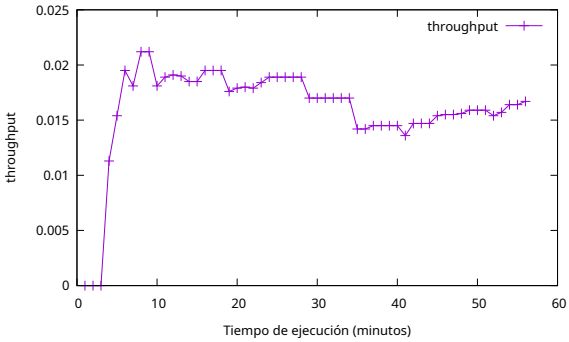
(a) Frecuencia



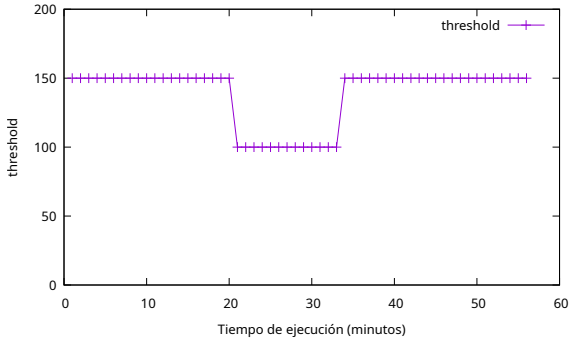
(b) Ocupación máxima (en pymemos)



(c) Potencia



(d) Throughput

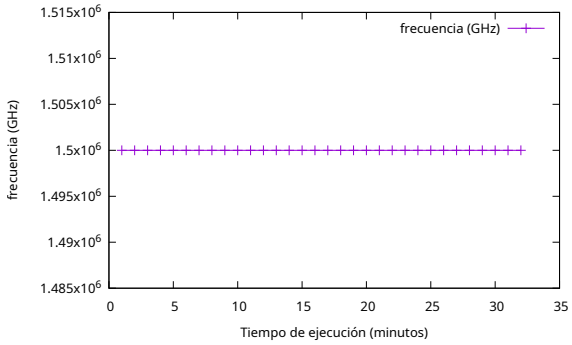


(e) Threshold

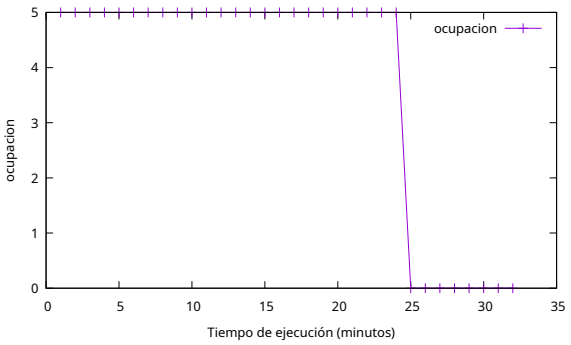
Figura 5.2: Cronograma de evolución del experimento 2 en el tiempo, threshold 150 y potencia limitada

5.2.3. Restricción *threshold* 0 y máxima energía por ejecución de *pymemo*

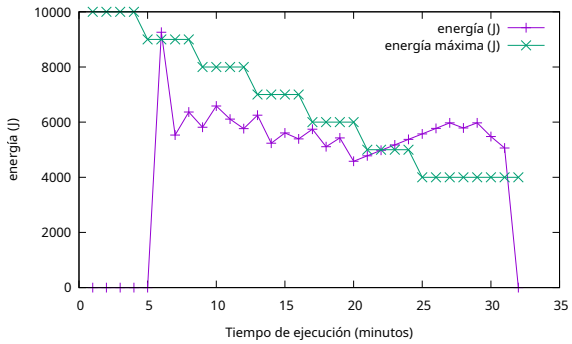
La Figura 5.3 muestra todas las métricas obtenidas durante este experimento. En la gráfica de la Figura 5.3e se demuestra que el sistema cumplió bien las restricciones por *threshold*, al no sobrepasar el valor máximo de cero. Al analizar los resultados se muestra cómo únicamente se tuvo una configuración hasta que que ninguna fue válida y, entonces, se suspendió, como muestran las Figuras 5.3a y 5.3b, escogiendo siempre 1,5 Ghz y cinco ejecuciones simultáneas. Esta configuración elegida es la que satisfaciendo la restricción energética, maximiza el *throughput*. En la gráfica de la Figura 5.3c se muestra cómo la energía por ejecución fue menor que la máxima permitida excepto al final, donde aumentó la energía. Esto es debido a que había peticiones en ejecución que hacen que aumente el coste energético (inercia de un sistema reactivo). En cuanto al *throughput*, se mantuvo a niveles estables mientras la configuración era constante, pero este disminuyó al final, al suspenderse el sistema debido a las restricciones energéticas Figura 5.3d.



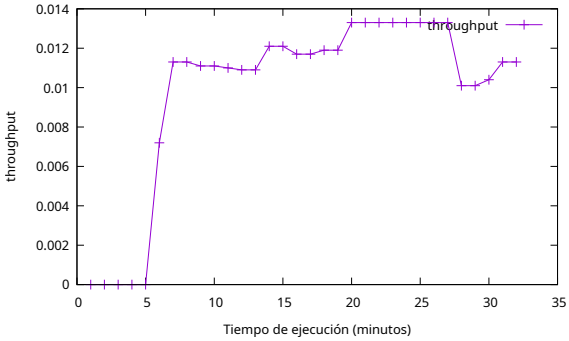
(a) Frecuencia



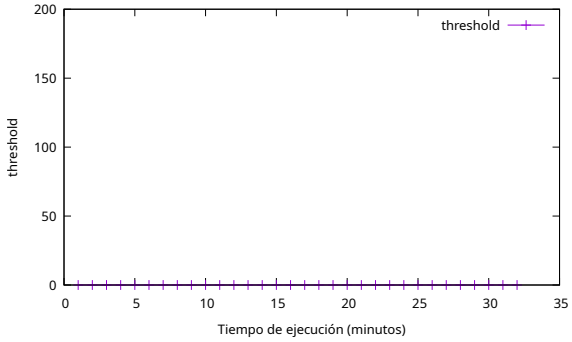
(b) Ocupación máxima (en pymemos)



(c) Energía por ejecución de pymemo



(d) Throughput

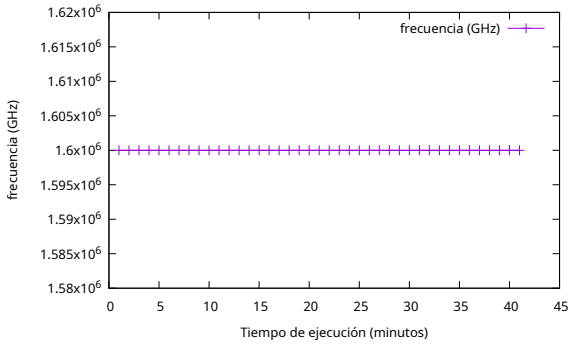


(e) Threshold

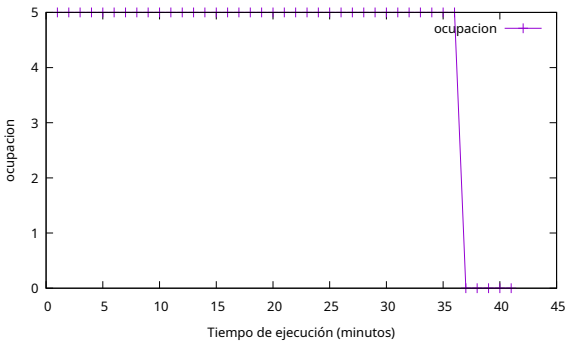
Figura 5.3: Cronograma de evolución del experimento 3 en el tiempo, *threshold* 0 y limite en energía por *pymemo*

5.2.4. Restricción *threshold* 150 y máxima energía por ejecución de *pymemo*

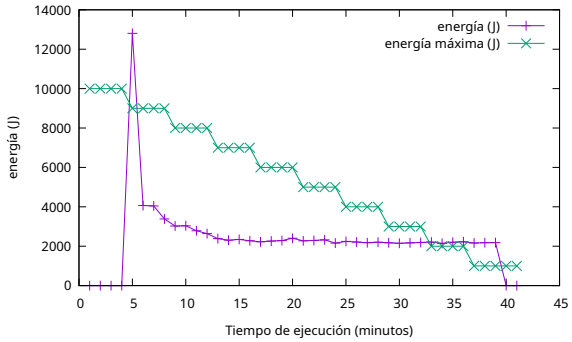
La Figura 5.4 muestra todas las métricas obtenidas durante este experimento. En este experimento la configuración no se modificó hasta que el sistema tuvo que suspenderse, como muestran las gráficas de las figuras 5.4e, 5.4b y 5.4a, siendo la configuración escogida 1,6 GHz, cinco ejecuciones máximas en paralelo y *threshold* 150. Como se muestra en la gráfica de la figura 5.4d, el *throughput* fue constante y máximo ya que era igual al valor de $\lambda = 0{,}025$. En cuanto a la energía por ejecución siempre fu menor como muestra la figura 5.4c, a excepción del final donde se mantuvieron durante un tiempo las peticiones que ya se habían aceptado en ejecución. Destacar que al habilitar ejecuciones con mayor *thresholds* que en el caso anterior, el sistema se suspendió diez minutos más tarde a pesar de tener las mismas restricciones energéticas.



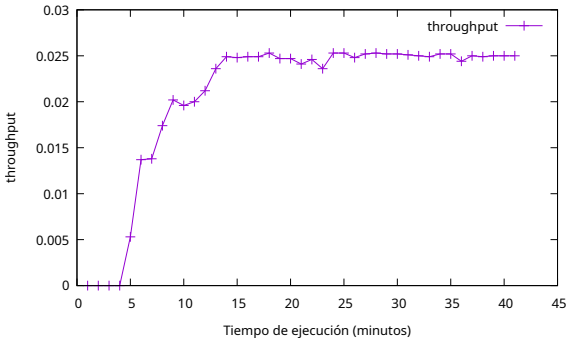
(a) Frecuencia



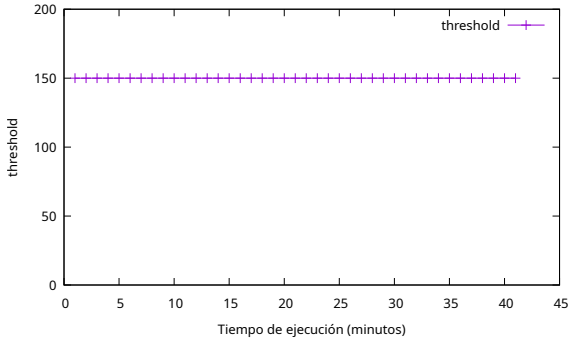
(b) Ocupación máxima (en pymemos)



(c) Energía por ejecución de pymemo



(d) Throughput



(e) Threshold

Figura 5.4: Cronograma de evolución del experimento 4 en el tiempo, threshold 150 y limite en energía por pymemo

5.3. Resumen

Como se ha mostrado en todos los experimentos, el modelo se adecúa correctamente a la situación del entorno. Sí es cierto que suele haber un intervalo de tiempo en todos los escenarios desde que se toma una decisión hasta que es realmente efectivo debido, a la inercia del sistema, por lo que al ser un modelo reactivo en la mayoría de casos hay pequeños momentos donde se incumplen las restricciones hasta que el sistema se estabiliza de nuevo. Se debe remarcar como diferencia clave que el modelo de la potencia trata de no sobrepasar el límite de esta, mientras que el de energía por *pymemo* utiliza siempre un máximo en torno a los 61 vatios. Debido a ello, la calidad del servicio para el primer modelo debería indicar cuánto es el gasto máximo energético del conjunto del sistema, pudiendo conocer a priori el gasto económico que supondrá. El otro modelo, en cambio, minimiza el gasto por ejecución, de tal forma que se podrá obtener el gasto máximo si conocemos la carga de trabajo.

Una clara diferencia entre los dos modelos es que en el de la potencia la configuración se va modificando, mientras que en el limitador de energía al comenzar encuentra la mejor y no la cambia: esto se debe a que coincide que la configuración más económica es la que tiene también mayor *throughput*.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

Tradicionalmente, los sistemas distribuidos cuentan con modelos y mecanismos para garantizar la calidad del servicio: sólo se se considera reducir el consumo energético una vez se haya garantizado las prestaciones acordadas. Sin embargo, con las fluctuaciones en el precio de la energía en el contexto actual, en muchos casos el coste económico de las ejecuciones puede ser *inasumible*. En este trabajo, se considera en todo momento un gasto energético máximo que el sistema puede permitirse, de manera que cuando el gasto sube o debemos tener menos gasto, el sistema se reconfigura para reducir el consumo, a costa de aumentar el tiempo de ejecución o reducir la precisión de los procesamientos.

Este sistema considera un tipo de aplicación que está generando datos continuamente para su procesamiento, utilizando una infraestructura *fog / cloud computing*: un centro de datos con una gran capacidad computacional y pequeños clústers de capacidad limitada, próximos a las fuentes de datos. Para garantizar la escalabilidad del sistema, el sistema sigue una estrategia de *scheduling* completamente distribuido: los datos generados en la fuente se llevan al clúster de poca capacidad más próximo donde se ejecutan si hay capacidad, de lo contrario, se mueven al siguiente nivel de la jerarquía, con mayor capacidad. Sobre esta estrategia de scheduling, el gestor de recursos de cada clúster cuenta con un modelo energético que garantiza que el coste energético no va a superar el máximo permitido. Se proponen como límites energéticos: limitar la potencia que se puede consumir o limitar el coste máximo energético de la ejecución de una tarea. El gestor de recursos de cada clúster puede actuar en cualquier nivel arquitectural: aplicación, middleware o hardware. En el nivel de la aplicación, se utilizan técnicas de computación aproximada para reducir la precisión de la ejecución con distinto grado. En el ámbito del middleware se reduce el número de máquinas activas y en el hardware se gestiona la frecuencia de los procesadores.

Para poder encontrar la configuración energética adecuada, cada controlador cuenta con un modelo obtenido experimentalmente por combinación de parámetros (*parameter sweep*), y utiliza un algoritmo voraz.

Para validar la aproximación, se ha utilizado una aplicación de análisis de vídeo y se han realizado varios experimentos en una infraestructura compuesta por dos clústers: uno con *raspberry pis* y otro, con rol de centro de datos con máquinas virtuales. Los experimentos muestran cómo el sistema es capaz de adaptarse a las restricciones de energía y consumir menos, disminuyendo el *throughput* y/o la precisión de los procesamientos. En unos experimentos se ha limitado la potencia máxima que se puede utilizar, en otros el gasto energético máximo para una aplicación.

6.2. Trabajo futuro

En el futuro este trabajo se podría mejorar de manera que el propio sistema aprendiera de la experiencia: por ejemplo, si el consumo energético basal cambiara por alguna operación del sistema operativo y esto fuera percibido, el sistema puede actualizar el modelo. Otra mejora podría ser la incorporación de un modelo predictivo del coste de la energía, haciendo que el sistema reaccionara con mayor celeridad.

Bibliografía

[1] Alcides Fonseca, Rick Kazman, and Patricia Lago. A Manifesto for Energy-Aware Software. *IEEE Software*, 36(6):79–82, November 2019. Conference Name: IEEE Software.

[2] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. Data Center Energy Consumption Modeling: A Survey. *IEEE Communications Surveys & Tutorials*, 18(1):732–794, 2016. Conference Name: IEEE Communications Surveys & Tutorials.

[3] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in computers*, 82:47–111, 2011.

[4] Congfeng Jiang, Tiantian Fan, Honghao Gao, Weisong Shi, Liangkai Liu, Christophe Cerin, and Jian Wan. Energy aware edge computing: A survey. *Computer Communications*, 151:556–580, 2020.

[5] Hancong Duan, Chao Chen, Geyong Min, and Yu Wu. Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems. *Future Generation Computer Systems*, 74:142–150, 2017.

[6] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. Approximate computing: A survey. *IEEE Design & Test*, 33(1):8–22, 2015.

[7] Rahul Yadav, Weizhe Zhang, Omprakash Kaiwartya, Prabhat Ranjan Singh, Ibrahim A Elgendy, and Yu-Chu Tian. Adaptive energy-aware algorithms for minimizing energy consumption and sla violation in cloud computing. *IEEE Access*, 6:55923–55936, 2018.

[8] Zhou Zhou, Jemal Abawajy, Morshed Chowdhury, Zhigang Hu, Keqin Li, Hongbing Cheng, Abdulhameed A Alelaiwi, and Fangmin Li. Minimizing sla violation and power consumption in cloud data centers using adaptive energy-aware algorithms. *Future Generation Computer Systems*, 86:836–850, 2018.

[9] Luiz Bittencourt, Roger Immich, Rizos Sakellariou, Nelson Fonseca, Edmundo Madeira, Marilia Curado, Leandro Villas, Luiz DaSilva, Craig Lee, and Omer Rana. The Internet of Things, Fog and Cloud continuum: Integration and challenges. *Internet of Things*, 3-4:134–155, October 2018.

[10] By:IBM Cloud Education. Faas.

[11] Kubernetes documentation.

[12] OpenFaaS Authors. Introduction.

[13] OpenFaaS Authors. Your first openfaas function with python¶.

[14] Yannick Becker and Stefan Naumann. Software based estimation of software induced energy dissipation with powerstat. *From Science to Society: The Bridge provided by Environmental Informatics*, pages 69–73, 2017.

[15] Vagrant.

[16] Giorgio C Buttazzo. Scalable applications for energy-aware processors. In *International workshop on embedded software*, pages 153–165. Springer, 2002.

[17] Chia-Ming Wu, Ruay-Shiung Chang, and Hsin-Yu Chan. A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters. *Future Generation Computer Systems*, 37:141–147, 2014.

[18] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *2009 international conference on high performance computing & simulation*, pages 1–11. IEEE, 2009.

[19] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[20] Zhou Zhou, Zhigang Hu, and Keqin Li. Virtual machine placement algorithm for both energy-awareness and sla violation reduction in cloud data centers. *Scientific Programming*, 2016, 2016.

[21] Jacob R Lorch and Alan Jay Smith. Improving dynamic voltage scaling algorithms with pace. *ACM SIGMETRICS Performance Evaluation Review*, 29(1):50–61, 2001.

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[23] Rafael Tolosana-Calasanz, Javier Diaz Montes, Omer F. Rana, and Manish Parashar. Feedback-control & queueing theory-based resource management for streaming applications. *IEEE Trans. Parallel Distributed Syst.*, 28(4):1061–1075, 2017.

[24] Rafael Tolosana-Calasanz, José Ángel Bañares, Congduc Pham, and Omer F. Rana. Resource management for bursty streams on multi-tenancy cloud environments. *Future Gener. Comput. Syst.*, 55:444–459, 2016.

[25] Fabian Kaup, Philip Gottschling, and David Hausheer. Powerpi: Measuring and modeling the power consumption of the raspberry pi. In *39th Annual IEEE Conference on Local Computer Networks*, pages 236–243. IEEE, 2014.

[26] Manish Parashar and Salim Hariri. Autonomic computing: An overview. In *International workshop on unconventional programming paradigms*, pages 257–269. Springer, 2004.

[27] Aaron Yi Ding, Ella Peltonen, Tobias Meuser, Atakan Aral, Christian Becker, Schahram Dustdar, Thomas Hiessl, Dieter Kranzlmüller, Madhusanka Liyanage, Setareh Maghsudi, Nitinder Mohan, Jörg Ott, Jan S. Rellermeyer, Stefan Schulte, Henning Schulzrinne, Gürkan Solmaz, Sasu Tarkoma, Blesson Varghese, and Lars Wolf. Roadmap for edge AI: a Dagstuhl perspective. *ACM SIGCOMM Computer Communication Review*, 52(1):28–33, March 2022.

[28] Jeretta Horn Nord, Alex Koohang, and Joanna Paliszkiewicz. The Internet of Things: Review and theoretical framework. *Expert Systems with Applications*, 133:97–108, November 2019.

[29] Praveen Kumar Reddy Maddikunta, Quoc-Viet Pham, Prabadevi B, N Deepa, Kapal Dev, Thippa Reddy Gadekallu, Rukhsana Ruby, and Madhusanka Liyanage. Industry 5.0: A survey on enabling technologies and potential applications. *Journal of Industrial Information Integration*, 26:100257, March 2022.

[30] Le Liang, Hao Ye, and Geoffrey Ye Li. Toward Intelligent Vehicular Networks: A Machine Learning Framework. *IEEE Internet of Things Journal*, 6(1):124–135, February 2019. Conference Name: IEEE Internet of Things Journal.

[31] Tainã Coleman, Henri Casanova, Loïc Pottier, Manav Kaushik, Ewa Deelman, and Rafael Ferreira da Silva. WfCommons: A framework for enabling scientific workflow research and development. *Future Generation Computer Systems*, 128:16–27, March 2022.

[32] Quoc-Viet Pham, Fang Fang, Vu Nguyen Ha, Md. Jalil Piran, Mai Le, Long Bao Le, Won-Joo Hwang, and Zhiguo Ding. A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art. *IEEE Access*, 8:116974–117017, 2020. Conference Name: IEEE Access.

[33] Ranesh Kumar Naha, Saurabh Garg, Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Longxiang Gao, Yong Xiang, and Rajiv Ranjan. Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions. *IEEE Access*, 6:47980–48009, 2018. Conference Name: IEEE Access.

[34] Khadija Akherfi, Micheal Gerndt, and Hamid Harroud. Mobile cloud computing for computation offloading: Issues and challenges. *Applied Computing and Informatics*, 14(1):1–16, January 2018.

[35] Geraldo F. Oliveira, Larissa Rozales Gonçalves, Marcelo Brandalero, Antonio Carlos S. Beck, and Luigi Carro. Employing classification-based algorithms for general-purpose approximate computing. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, San Francisco California, June 2018. ACM.

[36] Employing Classification-based Algorithms for General-Purpose Approximate Computing. pages 1–6, June 2018.

[37] Roberto Minerva, Abyi Biru, and Domenico Rotondi. Towards a definition of the internet of things (iot). *IEEE Internet Initiative*, 1(1):1–86, 2015.

[38] Cesare Pautasso and Gustavo Alonso. Parallel computing patterns for Grid workflows. pages 1–10, June 2006. ISSN: 2151-1381.

[39] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless computing: Current trends and open problems. In *Research advances in cloud computing*, pages 1–20. Springer, 2017.

[40] Luiz Bittencourt, Roger Immich, Rizos Sakellariou, Nelson Fonseca, Edmundo Madeira, Marilia Curado, Leandro Villas, Luiz DaSilva, Craig Lee, and Omer Rana. The Internet of Things, Fog and Cloud continuum: Integration and challenges. *Internet of Things*, 3-4:134–155, October 2018.

[41] By: IBM Cloud Education. Serverless.

Lista de Figuras

Anexos

Anexos A

Cronograma, actividades y objetivos

A lo largo de este proyecto se han realizado las siguientes actividades:

- Montar una infraestructura *edge*, incluyendo un clúster formado por *raspberries* y un clúster formado por máquinas virtuales en un servidor.
- Diseñar e implementar mecanismos para gestionar el gasto energético en los distintos niveles como disminuir la frecuencia o aproximación en cálculos.
- Habilitar una red virtual para la transmisión entre los distintos clústers.
- Elaborar un modelo e implementar un sistema inteligente que que se autoconfigure utilizando los mecanismos de gestión energética implementados.
- Realizar una validación experimental
- Escribir la memoria.

Todas estas actividades se organizaron en el tiempo como muestra la figura A.1

	FEBRERO				MARZO				ABRIL				MAYO				JUNIO		TOTAL
ACTIVIDADES	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	
A1 - infrastructure	10	10	10						10	10									50
A2 - machine		10	5	5	5	5	5		5	5									45
A3 - network		10	5	5															20
A4 - AI									5	5	5	5	15	15					50
A5 - experiments								5	10	10	10	10	15		15	10	10		95
A6 - writing																	10	30	40	80
TOTAL	10	30	20	10	5	5	5	5	30	30	15	15	30	30	30	30	20	20	340

Figura A.1: Cronograma - diagrama de Gaant