UNIVERSIDAD DE CANTABRIA

Facultad de Ciencias

TÉCNICAS DE MINERÍA DE DATOS APLICADO A LA MONITORIZACIÓN DE SISTEMAS

(Data Mining applied to the System Monitoring)

Trabajo de Fin de Grado

para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Aida Palacio Hoz

Director: Jose Luis Bosque Orero

Co-Director: Diego García Saiz

Febrero - 2015

Resumen

Uno de los problemas a los que se enfrentan las organizaciones es tener una gran cantidad de recursos computacionales localizados en distintos lugares de la empresa, pero que necesitan interactuar entre ellos para ejecutar y completar operaciones realizados en dichos entornos, lo que implica una baja eficiencia. En consecuencia de ello, las empresas han optado por concentrar todos los recursos en una única ubicación, para dar acceso a la información necesaria en las operaciones realizadas en el sistema, formando lo que se llama *Data Center*. El uso eficiente de los recursos computacionales de estos entornos es esencial ya que implica que se obtenga una reducción de gastos y una mejora del funcionamiento de la empresa, dando lugar a un mayor crecimiento de la misma. Es por ello que se necesita conocer y controlar el comportamiento de dichos recursos para mejorar su aprovechamiento y evitar posibles fallos en el sistema.

En vista de ese objetivo, la monitorización es el proceso que extrae y almacena información acerca del estado de los recursos de un entorno computacional. Para ello, se definen un conjunto de métricas que miden y permiten conocer, en tiempo real, aquellos recursos que más interese tener controlados, dependiendo de la funcionalidad del sistema. Que una empresa cuente con un sistema de monitorización también implica que exista un volumen de datos inmanejable, pero que se necesita analizar para una posterior toma de decisiones y una mejora del rendimiento de la empresa. Una solución a este problema, es utilizar Técnicas de Minería de Datos que exploran el conjunto de datos y obtienen información acerca de dichos datos.

Por tanto, el objetivo de este trabajo es desarrollar una herramienta de monitorización y análisis del comportamiento de los recursos disponibles, en función de una serie de métricas definidas para este proyecto.
En base al cumplimiento de este objetivo se puede comenzar a tomar decisiones que minimicen los gastos de una empresa y mejoren su crecimiento.

Palabras Clave: *Data Center*, monitorización, métrica, herramienta de monitorización, Minería de Datos, *clustering*.

Abstract

One of the problems in the enterprises is to have a huge amount of computational resources located in different places though the enterprise, but they need to interact each other to execute and complete the operations realized in these environments, that provides a little efficiency. As a consequence, the enterprises have concentrated all the resources in one place or room, giving access to the information needed in the operations of the system, creating a Data Center. The efficient use of the computational resources in these environments is important due to it provides a expenses reduction and a better performance of the enterprise. For this reason, we need to know and to control the state of these resources to improve their use and to avoid possible problems in the system.

As a result of that goal, monitoring is the process to take out and save information about the resources state in a computational environment. For that, must be defined a set of metrics that allow to measure and to know, in real time, the most interesting resources, to control, depending on system functionality. If a environment have a monitoring system, also cause a unmanageable volume data that we need analyse to act and improve the organization performance. One solution for this problem is to use Data Mining Techniques which explore the data set and obtain information about this data.

In conclusion, the target of this work is develop a monitoring and analysis tool to know the behaviour of the available resources, depending on set of metrics defined in this project.
If this target is achieved, we can make decisions to decrease the expenses and improve the evolution of the enterprise.

Keywords: Data Center, monitoring, metric, monitoring tool, Data Mining, clustering.

Agradecimientos

Quiero aprovechar estas líneas para agradecer las personas que me han ayudado y apoyado durante toda esta etapa académica, aquellas personas que son las culpables de que hoy esté aquí presentando este trabajo.

En primer lugar, agradecer a mi tutor Jose Luis y co-director, Diego, por darme la posibilidad de realizar este trabajo y ayudarme en todo lo que necesitase, desde el principio hasta el final ampliando los conocimientos adquiridos durante la carrera.

También agradecer a todos los profesores, que han contribuido a mi enseñanza durante estos cuatro años y medio que ha durado mi paso en la carrera.

Y por otra parte, dar las gracias a mi familia, sobre todo a mis padres Teresa y Jose Manuel que, aunque los primeros dos años costase, siempre me han apoyado para llegar hasta aquí. También a mis tíos, que siempre se sentían orgullosos de que siguiese adelante.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este capítulo se comienza describiendo las situaciones que nos motivan para realizar este proyecto, así como el problema y la solución planteada para resolverlo. Además, se marcan una serie de objetivos generales que se prevee alcanzar. Finalmente, se añade un nuevo apartado especificando cómo se estructura la memoria.

1.1. Motivación

Un Data Center, o "Centro de Proceso de Datos" (CPD) es aquella ubicación donde se concentran los recursos necesarios para el procesamiento de la información de una organización [1]. Dichos recursos consisten en las dependencias correctamente acondicionadas, equipos y redes de comunicación (**figura 1.1**). Con el término acondicionadas se refiere a que es un sistema que tiene instalado: climatización (aire acondicionado), alimentación eléctrica estabilizada e ininterrumpida, cableado estructurado, sistemas contra incendios, control de acceso, sistemas de cámaras de vigilancia, alarmas contra incendios, control de temperatura y humedad, etc. Gracias a estos componentes:

- Los datos son almacenados, tratados y distribuidos al personal o procesos autorizados para analizarlos o modificarlos.
- Los equipos en los que se almacenan los datos se encuentran en un entorno de funcionamiento óptimo.

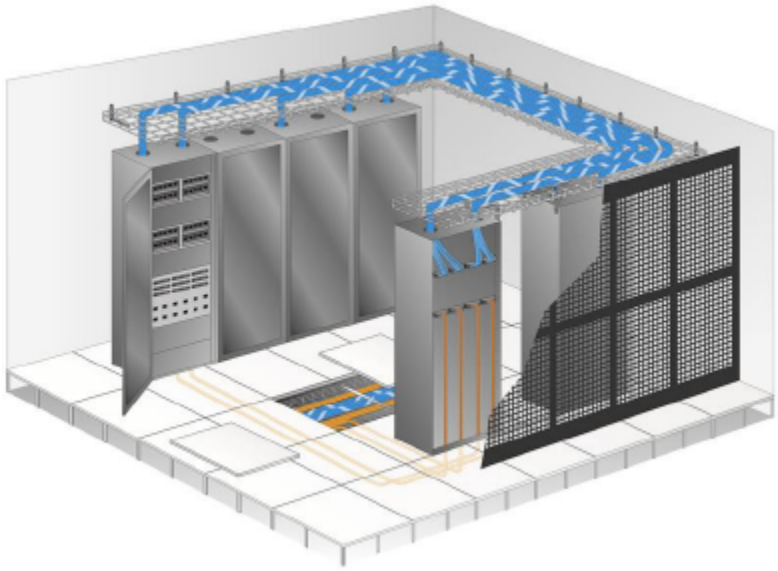


Figura 1.1: Esquema de un Data Center

La necesidad de gestión y optimización del espacio han hecho que estos sistemas estén basados en equipos cuyas dimensiones permiten aprovechar al máximo el volumen disponible, logrando una alta densidad de equipos por uni-

dad de espacio. Además, la rápida evolución de Internet y la necesidad de que los usuarios accedan a la información en todo momento, ha obligado a las empresas a requerir un alto nivel de fiabilidad y seguridad, de tal forma que se proteja la información, y esté disponible sin interrupciones. Por ello, se utilizan los CPD. Los *Data Centers* en estas empresas contribuyen significativamente a la minimización del riesgo operacional y a la continuidad del negocio.
Ejemplos de entornos que cuentan con CPDs pueden ser: empresas de telecomunicaciones como Telefónica, Bancos, compañías de seguros, etc. Por ejemplo, un banco puede tener un CPD con el propósito de almacenar todos los datos de sus clientes y las operaciones que estos realizan sobre sus cuentas. El Banco Santander cuenta con varios CPDs ubicados en distintos lugares del mundo. El mayor de ellos permite aumentar en un 35 % su capacidad de crecimiento y poder procesar más de 10.000 transacciones por segundo en todo el mundo, y más de tres billones al año. Una de las mayores empresas internacionales, como es Google, se estima que cuenta con más de 36 *Data Centers* alrededor del mundo, con unos 200.000 servidores y en constante crecimiento para permitir procesar más de 1000 millones de peticiones de búsqueda diarias. Por eso, una caída de los servidores durante un escaso período de tiempo provoca una pérdida enorme de datos y de dinero, además de dañar la imagen y la credibilidad de una empresa.
En estos entornos de cómputo es vital el uso eficiente de los recursos minimizando el impacto ambiental y maximizando su visibilidad económica. Ésto es el llamado *Green Computing* [2]. Una de las metas que persigue el Green Computing es la administración de energía. Se necesita una gran cantidad de energía eléctrica para que los equipos computacionales puedan operar, desde los servidores que se alojan en los CPDs hasta el aire acondicionado y la iluminación. Por tanto, las empresas necesitan tomar medidas para la reducción del consumo de energía en sus CPD.
Otro de los puntos claves a los que se enfrentan las empresas que cuentan con CPDs es el uso eficiente del espacio. Como ya se ha comentado, los CPD cuentan con unas dimensiones que permiten alojar la mayor cantidad de equipos por unidad de espacio. Pero las organizaciones están en constante crecimiento, lo que implica que el número de equipos se vaya incrementando para poder satisfacer las necesidades de los usuarios.
Mejorar la eficiencia de los CPD se basa en dos pasos:

1. Extracción de la información de los equipos y,
2. Actuar en base a dicha información.

Tomar decisiones en base al comportamiento de los recursos de los equipos es muy importante. Por ejemplo, se puede comenzar apagando aquellos equipos que no se están utilizando. Esta simple acción puede resultar en una reducción del 50 % del consumo por cada 100 servidores. Otro aspecto que se puede considerar es la implementación de procesadores que utilizan algoritmos capaces de reducir el consumo de energía o de incrementar el rendimiento del conjunto de equipos del sistema.

La estracción y gestión de la información de los equipos, de forma que sea útil para tomar decisiones estratégicas, paso 1, es el tema principal en el cuaĺ se centra este trabajo.

1.2. Problema y Solución

Como se ha comentado en el punto anterior, es necesario conocer el comportamiento de los recursos de un CPD para llevar a cabo estrategias que mejoren la eficiencia del entorno donde nos encontremos. Ésto implica que se obtenga una reducción de gastos y una mejora en el funcionamiento de la empresa.
Para ello, en este apartado se identifican los problemas que surgen a la hora de tener un control de los recuros y las soluciones para solventarlos. Uno de los problemas más comunes que surgen en estos entornos es no tener una clara visibilidad de las métricas que midan el estado de la infraestructura del centro de datos. Los administradores no detectan los problemas lo antes posible ni optimizan al máximo la asignación de recursos, como energía, conectividad de la red, espacio en la planta, etc. Existen una gran cantidad de formas de medir el rendimiento de los recursos debido a que cuentan con muchos tipos de datos. Por ejemplo, para conocer la utilización del procesador se puede medir el número de cambios de contexto, el número de interrupciones o los procesos que se están ejecutando. Todos ellos necesitan que la CPU tome el control para ejecutarlos, lo que implica que a un mayor número de ellos, mayor es el consumo de CPU. En consecuencia de ello, es necesario optimizar el conjunto de métricas para que nos ayuden a conocer el estado de los recursos de cada equipo, o lo que es lo mismo, monitorizar. Por tanto, se necesita:

1. definir qué parámetros interesan tener controlados dependiendo de la función del sistema donde nos encontremos, y
2. comenzar a monitorizarlos.

Para monitorizar las métricas, es necesario utilizar una herramienta o *framework* de monitorización que tenga constancia, en tiempo real, del estado de todos los recursos de cada nodo, además de poder almacenar los datos medidos para realizar un posterior análisis de ellos.
Monitorizar, en tiempo real, implica que exista una gran cantidad de datos almacenados para extraer la información que se oculta en ellos. Una solución a esta problemática es utilizar Técnicas de Minería de Datos. Estas técnicas realizan un tratamiento de la información de forma automática que puede ser útil para la toma de decisiones.

1.3. Objetivos

Una vez identificadas las soluciones que ayuden a tomar decisiones para mejorar la eficiencia en los CPD, en este trabajo, se marcan una serie de objetivos que las desarrollen. El objetivo principal de este proyecto es desarrollar una herramienta de monitorización y agrupación de los nodos de un CPD en función de una serie de métricas. Para alcanzar el propósito general es necesario dividir el trabajo en subobjetivos que permitan ir paso a paso.

- Definir las principales métricas que se monitorizan. Hay que tener en cuenta el tipo de clúster donde se trabaja y la funcion o funciones para la que se destina y escoger aquellos parámetros más interesantes que ayuden a la hora de agrupar los nodos.
- Implementar una herramienta de monitorización que permita llevar un seguimiento continuo de las métricas. Esta herramienta, *Graphite*, debe obtener las medidas del estado de los recursos, en el último intervalo de tiempo que se pueda medir, así como almacenar y actualizar los datos en tiempo real.
- Diseñar un panel de control, en la aplicación web de *Graphite*, para llevar un seguimiento, en tiempo real, de las métricas de aquellos nodos que más le interesen al administrador.
- Utilizar técnicas de Minería de Datos que exploren los valores medidos. Analizar aquellas técnicas de *clustering* que relacionen los datos más similares entre sí y quedarse con aquella que más se ajuste a nuestras necesidades.
- Realizar una experimentación exhaustiva que permita validar y verificar el comportamiento del algoritmo elegido y su correcto funcionamiento para este trabajo, además de incrementar el aprovechamiento y el rendimiento de los componentes del entorno computacional.
- Sacar conclusiones acerca de los resultados de las pruebas y analizar, otros aspectos que no se vayan a tratar en las pruebas, pero que también ayuden a evaluar si es correcto el funcionamiento del algoritmo para nuestro caso.

1.4. Estructura del documento

La memoria del proyecto se estructura en varios capítulos que se distribuyen de la siguiente manera:

1. En el capítulo 2, **Estado del Arte**, se explica en qué consisten, a grandes rangos, las herramientas de monitorización como *Ganglia* y *Graphite*, y las técnicas de Minería de datos. Será en esta última sección donde se describan algunos de los algoritmos más importantes de *clustering*. En este apartado se pretende dejar claro cuáles son los conceptos clave del proyecto.
2. Es en el capítulo de **Diseño** donde se describe detalladamente los conceptos más importantes de los que consta el trabajo. En esta sección, se definen las métricas elegidas para la monitorización, además del porqué de su elección, se describe en profundidad qué es *Graphite* y en qué consisten sus componentes, así como el algoritmo de *clustering* elegido, *k-means*.
3. Una vez explicado de manera teórica los distintos temas que se tratan a lo largo del documento, en el capítulo 4, se detalla la **Implementación** de cada uno de ellos. Este apartado consta de 3 puntos:
   - En la primera sección se detalla cómo se instalan y configuran los componentes de Graphite.
   - En esta siguiente parte, se explica cómo se monitorizan las métricas elegidas. Para ello se describe el directorio /proc, que está compuesto de ficheros de donde se obtienen los valores de las métricas, y los scripts ejecutados para medir dichos parámetros.

- En la última parte se describen los datos del fichero de entrada, cómo se implementa el algoritmo y los resultados que se obtienen en función del número de agrupaciones en las que se quiera dividir el conjunto de nodos.

4. El siguiente capítulo 5, **Evaluación Experimental**, consiste en explicar la realización de las pruebas en base a los datos obtenidos. Se divide en 2 apartados:
   - La primera parte detalla las pruebas realizadas y los resultados obtenidos en función de los datos medidos durante la monitorización.
   - Y la segunda parte consiste en sacar las conclusiones después de obtener las anteriores pruebas, así como añadir algunos aspectos que no se hayan incluido en los resultados.
5. Por último, en el capítulo 6, se resumen las **conclusiones** alcanzadas en el proyecto y los posibles trabajos futuros que se pueden realizar en base a este proyecto.

También se incluye un **apartado bibliográfico** citando las referencias a las que se ha accedido a lo largo del documento.

Capítulo 2

Estado del Arte

2.1. Monitorización

Hoy en día, debido a la fuerte expansión de los equipos tecnológicos en las empresas e instituciones, es clave el uso de herramientas que permitan monitorizar y supervisar el correcto funcionamiento y la eficiencia de los recursos y servicios de los que constan dichos equipos. Toda organización debería contar con su propio sistema de monitorización para prevenir de incidencias y conocer el aprovechamiento de los recursos disponibles.
Así, lo primero que se debe realizar es un análisis del sistema que se va a monitorizar para, entre otras cosas, detectar aquellos sistemas críticos que imposibilitan el buen funcionamiento de los sistemas, además de crear políticas de actuación frente a incidencias. Por eso es clave el papel del administrador en estos sistemas, al cual se avisará automáticamente cuando se produce una emergencia. Otro de los puntos clave en la monitorización es garantizar la disponibilidad de los recursos y servicios en el sistema para minimizar el número de fallos que puede ocasionar y sacar el mejor partido de ellos mejorando el rendimiento de todo el sistema.
Dado el crecimiento de los equipos en la empresas existen una serie de herramientas que permiten administrar la monitorización de forma remota y centralizada. Estas herramientas pueden monitorizar una cantidad muy alta de parámetros, por cada equipo: carga de cpu, memoria total o libre en disco, cantidad de memoria swap disponible, el uso de red (número de bytes enviados y recibidos), temperatura, humedad, etc. Ésto, sumado al incremento constante de los equipos en las organizaciones es un problema para la monitorización porque se hace demasiado pesado y hay que escoger aquellos recursos clave que se quieran controlar en cada equipo o en un momento dado para un problema en cuestión.

2.1.1. Ganglia

Ganglia es un sistema de monitorización distribuido y escalable para sistemas de alto rendimiento nacido en la Universida de *Berkeley (California, EEUU)* [3],[4]. Administradores e ingenieros que diseñaron esta herramienta vieron como solución el poder monitorizar desde un equipo la enorme cantidad de aplicaciones ejecutadas en paralelo en miles de nodos. En la actualidad, se usa en sistemas distribuidos como *clusters* o *Grids*. Los *clusters* son sistemas caracterizados por un conjunto de nodos con similitudes en cuanto a *hardware* y sistema operativo, por lo que solo es gestionado a partir de una única entidad de administración. En cambio, *Grids* son un conjunto de sistemas heterogéneos unidos por una red de comunicación de área extensa *(WAN)* conocida mundialmente como Internet. Así, son requeridas múltiples entidades de administración. Respecto a la arquitectura, Ganglia está basado en el prótocolo *multicast listen/announce* para monitorizar el estado de los *clusters*, es decir, cada nodo monitoriza sus propios recursos y envía paquetes *multicast* conteniendo la información de monitorización a la dirección *multicast* compartida por el *clúster*. La implementación de Ganglia se basa en tres componentes:

- **Gmond** es un demonio instalado en cada uno de los nodos del *cluster* en donde se van a medir las diferentes métricas. Gestiona la monitorización en un único *nodo*, implementando el protocolo *listen/announce*, y responde a las peticiones de estos determinados nodos-cliente enviando una representación XML de los datos monitorizados.
- **Gmetad** es un demonio instalado sólo en el nodo que se va a encargar de monitorizar los recursos recogiendo la información en diferentes ficheros de los nodos donde está instalado el *gmond*.

- **Gweb** es el demonio encargado de mostrar gráficamente, a partir de una interfaz web,el estado de los recursos que se están monitorizando en cada uno de los nodos mediante una interfaz de usuario configurable.

En la **figura 2.1** se observa cómo es la interfaz web de usuario de Ganglia en el servidor *www.ojito.atc.unican.es* de la Universidad de Cantabria. En la parte superior, se puede escoger el *clúster* o nodos que se quiere mostrar, las métricas que se quieren medir o el periodo de tiempo que se quiere representar. En el ejemplo de la **figura 2.1**, se observa un *clúster* que cuenta con 24 nodos en total, de los cuales solo 2 están en funcionamiento. En las gráficas están representados los valores que corresponden a la última hora con respecto al número de nodos, de procesos, carga de cpu del sistema, memoria en uso, compartida o de espacio *swap*, el consumo de red de entrada y de salida, etc.

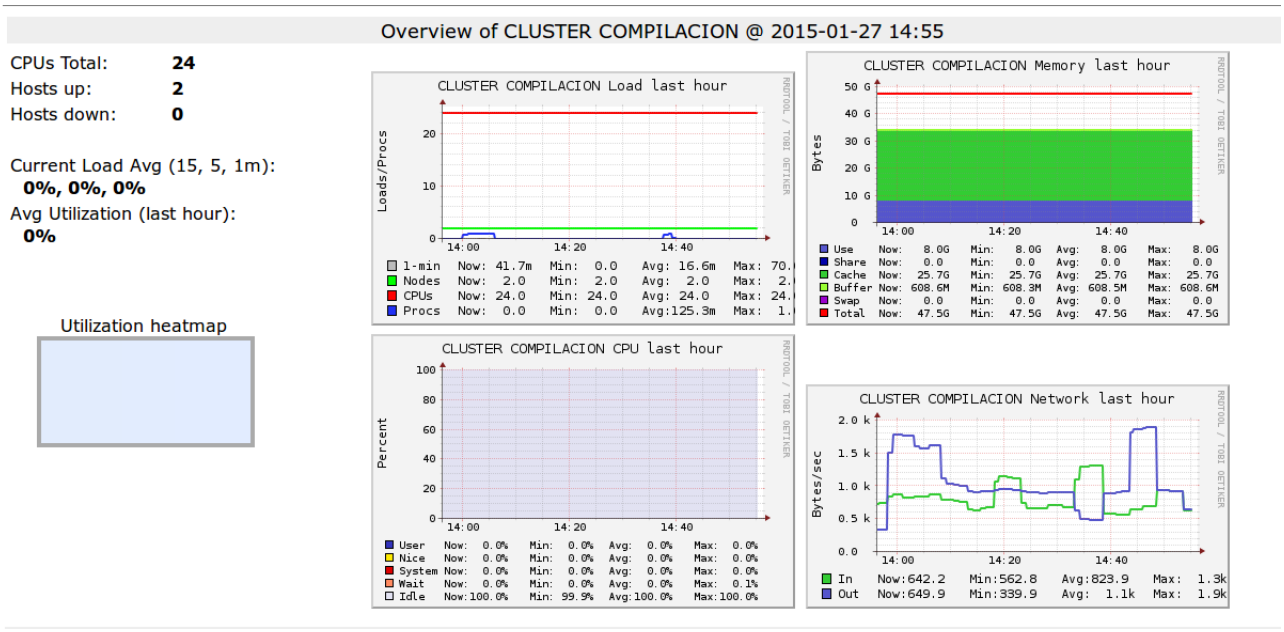


Figura 2.1: Ejemplo de representación gráfica de Ganglia en clúster ojito.atc.unican.es

2.1.2. Graphite

Graphite es una herramienta de monitorización originalmente escrita y diseñada por *Chris Davis* en la empresa *Orbitz* [5] en 2006 como un proyecto que en la actualidad está siendo desarrollado por un equipo de voluntarios en la organización *Graphite-Project GitHub* [6]. Graphite es un sistema que permite mostrar gráficamente la monitorización de los recursos del sistema en tiempo real, además de contar con una fuerte escalabilidad de forma que se eleve la carga de trabajo sin perder calidad de servicio.

Graphite está caracterizado por ser un sistema en *tiempo real* y *escalable*.

- **Tiempo real:** Incluso cuando la carga de trabajo es muy pesada, de forma que el número de datos en cada intervalo de tiempo supere la capacidad del sistema de almacenamiento, puede ejecutar operaciones de entrada/salida y muchos datos se están almacenándo en la caché de disco, *Graphite* es capaz de seguir actualizando los gráficos en tiempo real. Para implementar esta característica correctamente, cuando *Graphite webapp* recibe una petición para dibujar un grafo, simultáneamente recupera datos desde un disco y desde la caché de almacenamiento, así, combina las dos fuentes de datos para crear un grafo en tiempo real. Además, esto permite que se pueda crear un histórico de valores de los recursos a lo largo del tiempo y posteriormente puedan ser representados, sobre todo cuando estos valores cambian potencialmente en cada instante de tiempo. Por ejemplo, cuando se monitorizan parámetros de una gran cantidad de nodos que componen un clúster, como pueden ser la memoria libre o la carga de cpu, están modificándose continuamente.

- **Escalabilidad:** A medida que se añaden más equipos se obtiene más cantidad de información que se quiere monitorizar. Si en un instante dado, el *back-end* deja de funcionar, causará una mínima pérdida de datos

ya que el sistema está capacitado para manejar una determinada cantidad de carga en los equipos. Desde la perspectiva de entrada/salida, *Graphite* ejecuta una gran cantidad de pequeñas operaciones de entrada/salida en muchos archivos diferentes muy rápidamente, sin tener en cuenta cuál es la carga del sistema. Esto es debido a que cada métrica (operación) enviada a Graphite es almacenada en su propio fichero de datos. El gran volumen de métricas actualizándose cada periodo de tiempo requiere de un buen sistema *RAID* o *SSD* de almacenamiento.
Graphite está escrito en *Python*, al igual que el *back-end, carbon*, y la base de datos whisper. En cambio Graphite webapp está programada en *Django web framework* y usa el conjunto de herramientas *ExtJs javascript GUI*. La representación de gráficos se realiza mediante la librería *Cairo*.

El funcionamiento de *Graphite* es el siguiente: el usuario escribe una aplicación donde se encargará de definir los recursos que se van a representar, seguidamente al ejecutar dicha aplicación, los datos serán enviados al *back-end* de procesado, **carbon**, que será el encargado de guardar dichos datos en la base de datos, **whisper**. A partir de aquí, los datos pueden ser visualizados a través de las interfaces de **graphite webapp**. Graphite web tiene un entorno muy amigable para el usuario. El administrador puede situar en un mismo gráfico (**Figuras 2.2 y 2.3**) varias métricas de un mismo nodo o tener un gráfico donde se situaría una misma métrica perteneciente a distintos nodos para así comparar cada uno de ellos.

2.1.3. Ventajas de Graphite frente a Ganglia

Una de las principales ventajas es que Graphite permite representar y actualizar en tiempo real la monitorización de los datos mientras que Ganglia necesita de una previa edición del fichero de datos a monitorizar y una configuración del servidor web para poder obtener una nueva vista de la información. Por tanto, Ganglia tiene mayor dificultad a la hora de obtener una rápida administración de los servicios y recursos de un equipo.
Además, Graphite permite desplegar uno por uno información de un recurso en un equipo o varios equipos dados, mientras que Ganglia crea una vista de todos los recursos de manera simultanea. Por tanto, Graphite tiene una interfaz con una mayor interactividad construyendo una vista de grafos según la necesidad del administrador, permite guardar esa vista para que pueda estar disponible para otros usuarios, combinar diferentes grafos, etc.
Por último, una de las grandes ventajas de Graphite es su fuerte escalabilidad explicada anteriormente. Esta herramienta permite manejar gran cantidad de información causando mínimas pérdidas de datos.

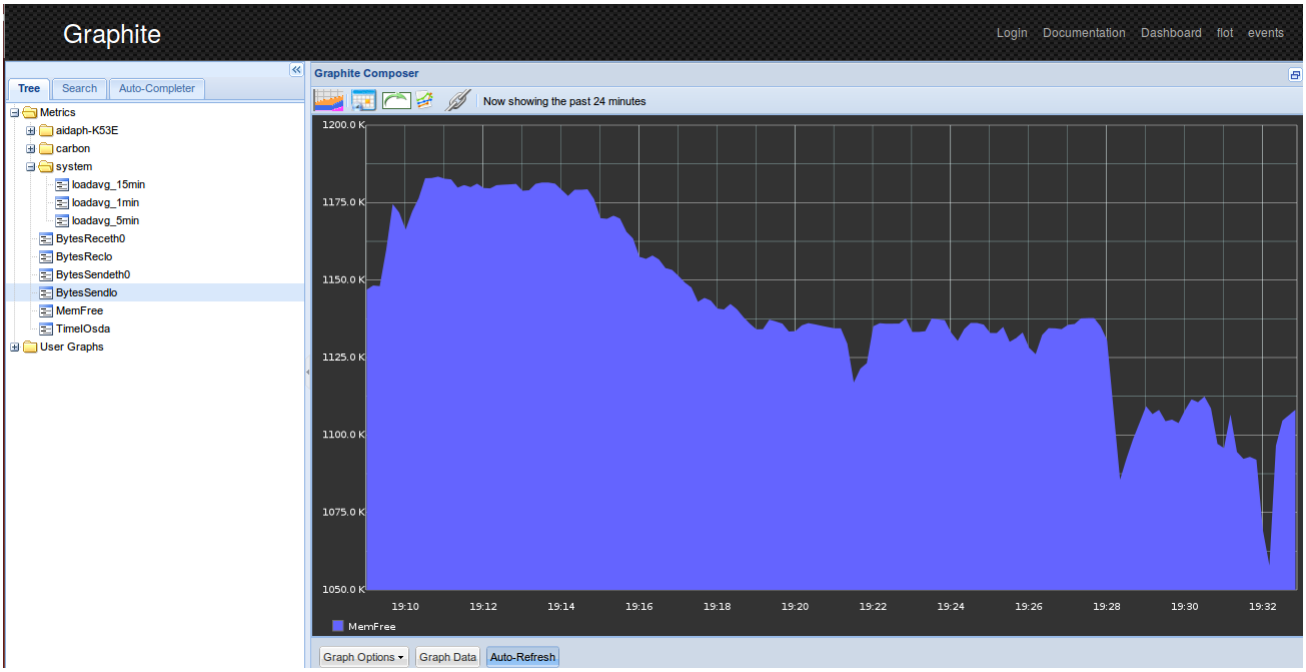


Figura 2.2: Monitorizar una métrica en un solo gráfico.

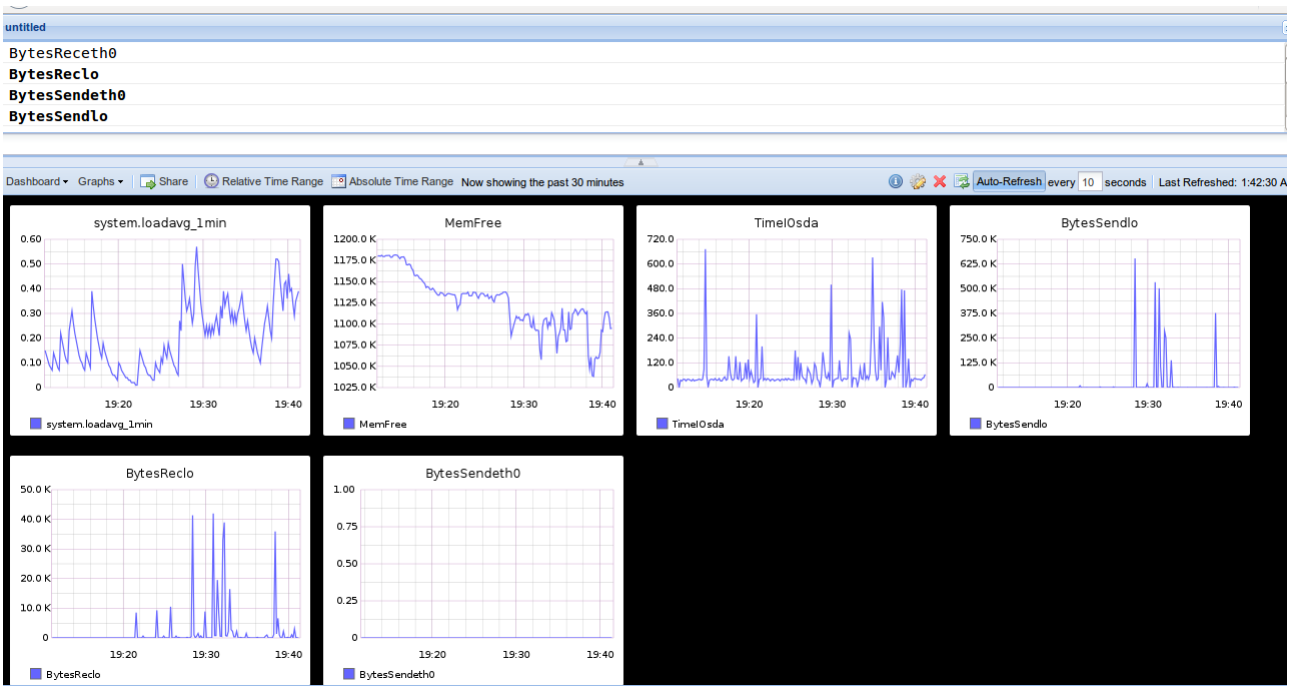


Figura 2.3: Dashboard representando varias métricas del mismo nodo.

2.2. Técnicas de Data Mining

La monitorización permite supervisar el correcto funcionamiento de los equipos en cada instante de tiempo. Para ello, es necesario controlar cada uno de los recursos o aquellos más importantes para las tareas que se tengan que realizar, lo que conlleva tener una cantidad muy alta de parámetros que gestionar. Se informa del estado de cada uno de los recursos en tiempo real, o después de haber pasado un periodo de tiempo. Todo ello provoca que el número de datos que se almacenan es muy alto. Además de utilizar herramientas gráficas para medir el estado de nuestro equipo, se necesitan técnicas que permitan explorar los datos, con el objetivo de encontrar información, tendencias o patrones que expliquen el comportamiento de los datos en un contexto determinado. Ésto es el llamado **Data Mining**[13].

Dentro de la Minería de datos, un tipo de técnicas ampliamente utilizado es el ***Clustering***. Consiste en la exploración de los datos para particionarlos en grupos donde el conjunto de datos sea similar entre ellos y lo más diferente posible a los de otros grupos. El proceso llevado a cabo en el *clustering* está dividido en dos fases principales[7]:

1. *Segmentación o Disección*, cuyo objetivo es particionar los datos de manera que los grupos obtenidos sean lo más prácticos para los fines del usuario, o administrador en este caso. Por ejemplo, a la hora de monitorizar un equipo hay que tener en cuenta, por un lado, la carga de cpu, por otro lado, la memoria disponible, también el uso de la red, etc.

2. Y ver si los datos que han caído dentro de un grupo son similares a los miembros de ese mismo grupo pero diferentes a los miembros de los demás grupos.

Esisten numerosos algoritmos de *clustering* publicados, que se pueden distinguir en varios tipos [8]. A continuación se definen aquellos que pueden resultar más prácticos. El **Clustering jerárquico** agrupa datos, para formar un grupo, basándose en la distancia. Cada agrupación está caracterizada por la máxima distancia necesaria para conectar partes de él. Se representan mediante un dendograma (representación gráfica en forma de árbol), donde, a medida, que la distancia se incrementa se forman diferentes grupos. Estos algoritmos no forman un único particionamiento sino que, proporcionan una extensa jerarquía de agrupaciones que se mezclan con otras dependiendo de la distancia. A modo de ejemplo se muestra la **figura 2.4** para entender mejor el funcionamiento del algoritmo. Suponiendo los 6 elementos de la primera imagen de la figura 2.4 y usando como distancia la Euclidea, si se corta el dendograma después de la segunda fila se obtienen los grupos a b c d e y f. Si, en cambio, se corta el dendograma

después de la tercera fila se obtienen las particiones a b c d e f. Por tanto, dependiendo la distancia que se escoga se obtendrán unos grupos u otros. Cuanto mayor sea la distancia, menor es el número de particiones. Los algoritmos de **Clustering basados en *centroides*** están representados por un *centroide*, que no suele pertenecer al conjunto de datos, y es el punto central de cada uno de los "k" grupos que se definen. Estos algoritmos definen aleatoriamente los puntos centrales y asigna cada uno de los datos al *centroide* más cercano. Este tipo de *clustering* es considerado uno de los métodos más usados y efectivos en las aplicaciones científicas, industriales o tecnológicas. Existen varios algoritmos de este tipo como puede ser *k-means, k-medoids, k-medians, fuzzy c-means*, etc. Por ejemplo, *k-means* es usado en el análisis de clústers para agrupar una gran cantidad de datos en *"k"* particiones, en el procesamiento de imágenes dividiendo el conjunto de colores de una imagen en k colores fijos, etc. Por eso, es el algoritmo elegido en este trabajo y se explica más claramente en la siguiente sección y en el capítulo 3. El **Clustering basado en la distribución** se fundamenta en modelos de distribución en estadística. Los grupos se basan en la probabilidad de que los objetos que los incluyan pertenezcan a la misma distribución. Y el **Clustering basado en densidad** define los grupos como areas de mayor densidad que el conjunto de los datos donde se agrupan los puntos que están más cercanos entre sí. Los puntos que se localizan en areas menos 'pobladas' se consideran como ruido o frontera. El algoritmo más popular de este tipo es *DBSCAN*, que consiste en conectar los puntos que se encuentran dentro de ciertos umbrales de distancia. Además, se debe satisfacer un criterio de densidad, debe existir un mínimo número de puntos dentro del radio de un determinado punto. En los ejemplos de la **figura 2.5** se ve este funcionamiento donde se distinguen los grupos en los cuales se concentran la mayoría de puntos.

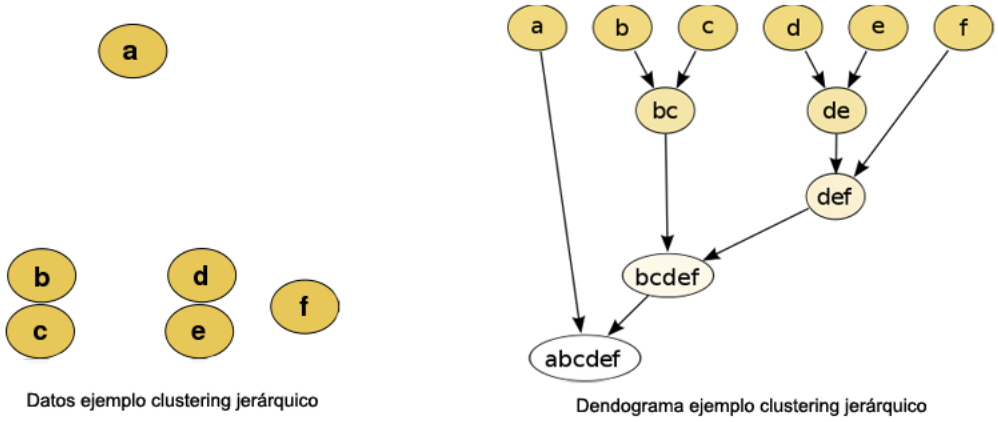


Figura 2.4: Ejemplo Clustering Jerárquico. Fuente: *Wikimedia Commons*

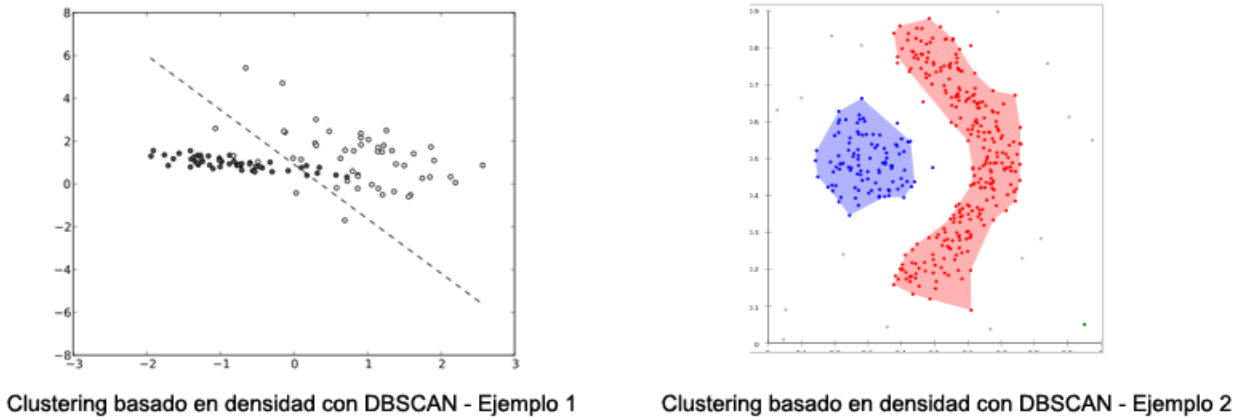


Figura 2.5: Ejemplo clutering basado en Densidad con DBSCAN. Fuente: *Wikimedia Commons*

Capítulo 3

Diseño

En este capítulo se define, el entorno donde se monitoriza y se realizan las pruebas así como las métricas que se monitorizan. Además, se explica en qué consisten la herramienta de monitorización, *Graphite* y el algoritmo de *clustering* escogido, *k-means*, que se utilizará para agrupar los nodos según las métricas seleccionadas.

3.1. Definición del entorno

La monitorización, con sus posteriores pruebas, se realizan en el entorno computacional de cálculo en el Centro de procesamiento de Datos 3MARES, construído y ensamblado en la Facultad de Ciencias de la Universidad de Cantabria. El elemento de cálculo de este entorno, en el cuál se ejecutan dichas pruebas y se obtienen los datos, es el ***clúster* HPC calderon.atc.unican.es** [17]. Un *clúster HPC* esun sistema multicomputador compuesto por un elevado número de computadores completamente independientes, se encuentran disponibles más de 800 procesadores que cada vez más se irán ampliando, dada su naturaleza de alta escalabilidad, interconectados entre sí por una red de altas prestaciones, en la que, uno o varios de los computadores mantiene el rol de *frontend*, y el resto son nodos de cálculo. Lo mismo ocurre con los usuarios de los sistemas de cálculo del grupo ATC, que es superior a 60, actualmente, pero que, seguramente, seguirá creciendo. Más detalladamente, el *clúster* donde se trabaja está compuesto por:

- **2 nodos frontend**: Estos nodos son los encargados de la comunicación entre el usuario y el *clúster*. Por eso, el frontend tiene la función de gestión y de control: gestión de colas, gestionar las cuentas de usuarios pudiendo dar acceso a otros usuarios, añadir trabajos (por medio de *scripts*) al sistema de colas de Calderon, etc. En el último punto mencionado, hay que destacar que los frontends también realizan la comunicación con los nodos de cómputo, encargados de ejecutar los scripts antes mencionados. En estos nodos no es recomendable ejecutar directamente los programas ni compilar código ya que, supone una carga añadida que retrasa y obstaculiza las tareas principales del clúster.
- 2 **nodos de compilación**: Estos dos nodos cumplen con las mismas tareas que cualquiera de los nodos de cálculo, pero están destinados exclusivamente para compilación y depuración de código. Se podría monitorizar un proceso de depuración de código, pero para el objetivo de este trabajo no es necesario.
- **Nodos de cómputo**: En estos nodos se ejecutan los procesos de comunicación con el frontend, el control de ejecución de los trabajos y además, son los encargados de controlar el estado de los recursos del sistema. También se encargan de ejecutar los *scripts* de trabajo enviados por el usuario. Debido a diferentes características hardware de cada uno de los nodos, como puedan ser el nº de procesadores, el tamaño de memoria principal, la frecuencia del procesador, etc. el clúster se divide en 10 grupos. En este trabajo, solo se va a detallar aquellos grupos en los que se ha estudiado, aunque hay que tener en cuenta, que no todos los nodos de cada conjunto han formado parte en la ejecución de los *scripts*:
  1. **Grupo 2 (g2):** formado por 21 nodos con 2 procesadores (4 cores) AMD Optaron 276/275 a 1.8/2.2 GHz y 8 GB de memoria principal cada uno. En este grupo se emplearán los nodos *compute-0-10,compute-0-11, compute-0-14, compute-0-15*.
  2. **Grupo 3 (g3):** formado por 16 nodos con 2 procesadores (8 cores) Intel Xeon X5472 a 3 GHz y 16 GB de memoria principal cada uno. Los nodos que ejecutan los scripts de trabajo en este grupo son

compute-2-35, compute-2-36, compute-2-37, compute-2-38, compute-2-39, compute-2-40, compute-2-41, compute-2-42, compute-2-43 y compute-2-44.

3. **Grupo 4 (g4):** formado por 12 nodos con 2 procesadores (8 cores) Intel Xeon X5472 a 3 GHz y 16/32 GB de memoria principal cada uno.Los nodos de este grupo que se emplearán son: *compute-3-48, compute-3-50, compute-3-51, compute-3-52, compute-3-53, compute-3-54, compute-3-55 y compute-3-56.*
4. **Grupo 5 (g5):** formado por 2 nodos con 2 procesadores (8 cores) Intel Xeon X5550 a 2.67 GHz y 32 GB de memoria principal cada uno. Aquel nodo del grupo empleado es el siguiente: *compute-3-63.*
5. **Grupo 6 (g6):** formado por 20 nodos con procesadores (12 cores) Intel Xeon X650 a 2.67 GHz y 24/48 GB de memoria principal cada uno. En este grupo, todos los nodos ejecutarán el trabajo enviado: *compute-4-64, compute-4-65, compute-4-66, compute-4-67, compute-4-68, compute-4-69, compute-4-70, compute-4-71, compute-4-72, compute-4-73, compute-4-74, compute-4-75, compute-4-76, compute-4-77, compute-4-78, compute-4-79, compute-4-80, compute-4-81, compute-4-82 y compute-4-83.*
6. **Grupo 7 (g7):** formado por 20 nodos con procesadores (12 cores) Intel Xeon E645 a 2.4 GHz y 54 GB de memoria principal cada uno. Igual que en el grupo anterior, todos sus nodos ejecutarán el script de trabajo: *compute-5-84, compute-5-85, compute-5-86, compute-5-87, compute-5-88, compute-5-89, compute-5-90, compute-5-91, compute-5-92, compute-5-93, compute-5-94, compute-5-95, compute-5-96, compute-5-97, compute-5-98, compute-5-99, compute-5-100, compute-5-101, compute-5-102 y compute-5-103 .*

Como se ha visto al enumerar los nodos de cada grupo, no todos ellos han ejecutado el *script* o trabajo enviado por el usuario. Ésto es debido a que el Sistema de Colas, con el que cuenta Calderon, es el que controlcentroidesa qué nodos ejecutan el trabajo. En total, el **número de nodos** en los cuales se realizan las pruebas es **63**.

Sistema de Colas

Antes de explicar de qué trata este mecanismo hay que situarse en por qué surgió. El clúster HPC tiene la función de soportar grandes cantidades de trabajo, pero llega a un punto en el que los usuarios pretenden ejecutar más trabajos que procesadores existentes, y todo ello de forma simultánea. Si ésto ocurre no se saca partido al uso de estos supercomputadores, por dos razones principales:

- *Cambio de contexto*: Trabajos, que están ejecutándose, deben abandonar el procesador durante un periodo de tiempo para que así, otros trabajos que se encuentran parados puedan obtener un procesador disponible. Al tener varios trabajos sin haber finalizado se ve, claramente, como este proceso es muy costoso en tiempo.
- Ya se ha detallado anteriormente el *tamaño limitado de la memoria principal* de los procesadores. Así, puede ocurrir que el total de memoria que quede disponible, no sea suficiente para la ejecución de todos los trabajos provocando que el sistema comience a utilizar la memoria virtual. Debido a que la memoria virtual está en los discos duros, tiene una velocidad menor de transferencia a la de la memoria física. Por tanto, aquellos trabajos que necesiten de memoria principal tardan más tiempo que los que solo requieren memoria física.

En conclusión, el no usar las prestaciones que ofrece el clúster, como es el Sistema de Colas, implica que los usuarios se vean afectados, sobre todo, por un **coste de tiempo**.

Debido a toda esta problemática surgieron los sistemas de trabajos en *batch*, conocidos como **sistemas de colas**. Este mecanismo gestiona los recursos del sistema para que se encuentren en sus puntos óptimos. Configurando un fichero, como se detalla seguidamente, es posible que el usuario se despreocupe de controlar cómo se encuentra el sistema y sea el gestor de colas el que lance los trabajos de manera que no se superen los valores admisibles, evitando que el usuario envíe los trabajos a cada nodo de forma directa. Es por eso que no todos los nodos del sistema han sido utilizados, solo aquellos que se encuentran disponibles, sin revasar sus limitaciones en el momento en el que el usuario pide al gestor de colas que lance su *script.*

Antes de explicar cómo configurar el sistema de colas, para que el usuario trabaje, hay que **loguearse para entrar al clúster** usando el protocolo *Secure Shell (SSH).*

Tras pedir acceso al *cluster*, se **introduce la contraseña de usuario**. A partir de aquí, el usuario ya se encuentra en su correspondiente directorio de trabajo en uno de los dos frontends. Para acceder a los demás nodos,

únicamente en algún caso exclusivo, basta con conectarse mediante el protocolo *SSH* introduciendo, además, la dirección IP del nodo al que se quiere acceder o su *hostname*.

El directorio de trabajo del usuario está 'replicado' en cada computador. Es decir, el directorio que ve desde el *frontend* es el mismo visto desde uno de los nodos. Por ejemplo, si se crea un *script* de trabajo en una nueva carpeta del *frontend*, cada uno de los nodos puede modificar ese *script* permitiendo que el cambio permanezca para los demás computadores. Como conclusión, el *frontend* es considerado el clúster para el usuario, mientras que el resto de computadores son accesibles (envío de trabajo) mediante el sistema de colas.

Para las pruebas que se van a realizar se van a ejecutar *scripts* de trabajo secuenciales (es decir, los nodos no necesitan comunicarse entre sí sino que cada uno ejecuta su tarea de forma independiente) y de corta duración (40 min como máximo tarda cada equipo en ejecutar su trabajo). Por tanto, para la configuración del gestor de colas crea un nuevo fichero de configuración en el *frontend*:

```
#!/bin/sh
# Script de lanzamiento trabajos secuenciales/multiples
# PARAMETROS GLOBALES # PARAMETROS SGE-EE
# Shell que usará el sistema de colas para ejecutar el trabajo en el servidor.
shell_name= "/bin/bash"
# Nombre del trabajo que se ejecutará en los nodos.
job_name=" Graphite"
#Dirección de email donde el gestor puede enviar el estado del trabajo
email=" user@alumnos.unican.es"
# Path en el cual se almacenan los ficheros de salida (-o) y error (e) del trabajo.
output_path="/afs/atc.unican.es/u/a/aida/output/"
error_path="/afs/atc.unican.es/u/a/aida/output/"
# ENVIAR el TRABAJO AL SISTEMA DE COLAS
qsub « eor
# TICKETS Policy
# PERMISOS SGEEE-AFS
kinit -k -t /home/Keytab/krb5.keytab.$USER $USER 2>/tmp/kinit aklog -c ATC.UNICAN.ES
-k ATC.UNICAN.ES 2»/tmp/kinit
# Encapsulado del trabajo
if [ true ]; then
-----------------------------------------
# Trabajo (Job) por nodo ...
        cd $HOME
        cd scripts/
        python total.py
        sleep 2
-----------------------------------------
fi 1 > $output_path$REQUEST.o$JOB_ID 2 > $error_path$REQUEST.e$JOB_ID
kdestroy
-fr /tmp/kinit
eor
```

Además de cada una de las opciones que vienen especificadas en el fichero, cabe destacar, la parte en la que está encapsulado el trabajo que se va a enviar a uno de los nodos cuando se ejecuta dicho *script*. Lo que se coloca entre las dos líneas dentro del fichero, ***job***, va a ser lo que se envie al nodo elegido por el gestor. Para este trabajo, se ejecuta el *script* **total.py** encargado de enviar a las bases de datos de *whisper*, creadas en el *frontend* de *Graphite*, cada uno de los parámetros que se están midiendo en el nodo donde se esté ejecutando. Como se necesita más de un nodo para realizar las pruebas, se crea otro fichero que tiene la misión de ejecutar el primer *script* una cantidad iterativa de veces(número de equipos que se quiera monitorizar). Es decir enviará ese *job*, tantas veces como se haya indicado, en este caso, se realizarán 90 iteraciones. Este gestor de colas envía el *job* a aquellos nodos que se

encuentren disponibles, o mejor dicho, aquellos que no hayan superado sus valores admisibles. Los *jobs* lanzados que no encuentren nodos disponibles pasarán a estado *wait* hasta que algún otro *job* haya finalizado.

```
#!/bin/sh
cd $HOME
count=92
for i in  `seq 0 $count`
do
        ./run.sh
        sleep 3
done
```

Figura 3.1: Ejecución del trabajo en cada nodo

Tanto en este fichero de la **figura 3.1** como en el *job* del fichero de configuración se coloca, al final del *loop*, un *sleep* para no saturar al sistema de colas cada vez que envía el *job*.

3.2. Monitorización

En los CPD se concentran todos los recursos necesarios para el procesamiento de la información de una organización. Es por ello que se necesita tener un sistema de monitorización que supervise el estado de todos los recursos de un entorno, para anticipar problemas que puedan surgir. Monitorizar un sistema es el proceso, dentro de un sistema distribuido, que extrae y almacena información de estado de los recursos. Un sistema distribuido se define como una colección de computadores separados fisicamente y conectados entre sí por una red de comunicaciones. Cada equipo contiene sus componentes hardware y hordware que el administrador percibe como un único sistema.

Cuanto se monitoriza un sistema que está ejecutando aplicaciones o servicios, un equipo tiene la función de monitor. El monitor realiza dos pasos para decidir cómo monitorizar:

1. Definir qué datos va a monitorizar, y
2. Configurar su herramienta de monitorización para obtener dichos datos y representarlos por pantalla.

Para definir los parámetros que se van a monitorizar es necesario conocer los recursos que se monitorizan, definidos en el siguiente apartado, y el período de tiempo de monitorización. Cuando se monitoriza se suele obtener información en tiempo real y de forma continua, por lo que es necesario crear un historico con los datos que se miden en función del tiempo que requieran ser almacenados. En este trabajo, un objetivo principal es analizar el los patrones e información de los datos obtenidos por lo que se seleccionan los datos medidos en el último momento. Ésto se conoce como **sampling**. *Sampling* es la técnica para la selección de una muestra a partir de una población. Al elegir una única muestra se espera conseguir que los resultados sean parecidos a los de todo el conjunto. Este proceso permite ahorrar recursos, como el tiempo, y a la vez, obtener resultados similares a si se analizase toda la población.

El rendimiento en un sistema de monitorización se basa en dos aspectos:

- Impacto en el equipo donde está instalado: Es decir, cualquier elemento del sistema de monitorización que impida el correcto funcionamiento de los procesos principales del equipo. Idealmente, monitorizar es un servicio que requiere simplicidad ya que debe permitir que el impacto en el sistema sea mínimo.
- La monitorización debe ser eficiente: La aplicación debe ser capaz de monitorizar los parámetros dentro del período de tiempo deseado. Ésto está relacionado con la escalabilidad.

Hoy en día, tanto pequeñas empresas como organizaciones internacionales pueden estar compuestas de Centros de Datos. La diferencia es que las grandes empresas cuentan con centros de datos compuestos por una gran cantidad de equipos que hay que supervisar continuamente. El problema de escalabilidad que surge aquí provoca que la cantidad de recursos que se controlan es enorme en comparación con entornos más pequeños y la monitorización se hace muy pesada.

Además, los sistemas de monitorización son tolerantes a fallos, es decir, tienen la capacidad de acceder a la información del estado de los recursos, aun en caso de producirse algún fallo o anomalía del sistema. La implementación de la tolerancia de fallos requiere que el sistema de almacenamiento guarde la misma información en más de un dispositivo físico, deSSH esta forma, si se produce algún fallo que pueda ocasionar pérdida de datos, el sistema es capaz de restablecer la información. Que las aplicaciones de monitorización tengan esta característica es muy importante, sobre todo, si son entornos donde se trabaja con información crítica como las entidades financieras, gobierno, coorporaciones, etc.

3.3. Definición de Métricas

La definición de métricas es uno de los puntos claves de este trabajo ya que, para realizar la monitorización de los equipos se necesita hacer un análisis de aquellas medidas más interesantes que reflejen, cuál es el grado de eficiencia de un único equipo o un clúster completo. Ésto es muy dependiente de las necesidades concretas de la aplicación o cliente para el que se desarrolle. En este trabajo, se va a monitorizar un clúster HPC descrito en la anterios sección. En consecuencia de ello, los factores que más interesan medir están relacionados con el rendimiento de sus distintos componentes.
Existe un conjunto de recursos que se pueden medir en los equipos, pero es indispensable elegir correctamente aquellas métricas que recojan más información útil, y no saturen con datos innecesarios.

Realizando un análisis de los recursos que componen un equipo, desde el directorio */proc*, que se describe en el capítulo 4 de Implementación, se van a monitorizar cuatro: cpu, memoria RAM, almacenamiento de disco y red, ya que son los componentes más importantes en la arquitectura de este sistema y para el análisis y experimentación de los datos que más nos interesan según la problemática inicial. Analizar el rendimiento de estos componentes no es tan simple ya que hay muchos tipos de datos para cada recurso. Por ejemplo, para conocer el rendimiento de la CPU se puede medir el número de interrupciones, de cambios de contexto o de procesos que se están ejecutando. Para cada componente mostrado en la **figura 3.2** se exploran las métricas más útiles para nuestro objetivo:

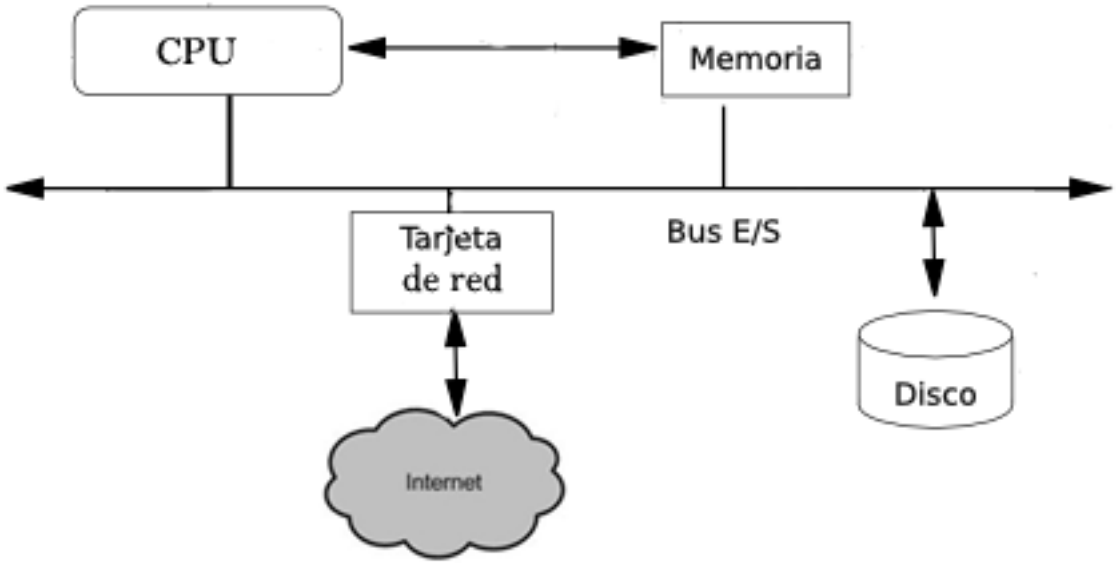


Figura 3.2: Esquema Componentes de Monitorización

- **Carga de CPU**: Esta métrica indica el % de uso del procesador, lo que permite comprobar cómo de ocupado está el procesador. Si se encuentra ejecutando continuamente procesos, habrá otros procesos esperando su turno para ser atendidos lo que implica un mayor tiempo de respuesta en la atención de dichos procesos. Por tanto, a mayor uso de CPU, mayor es el tiempo de respuesta a todos los procesos enviados por el usuario.
- **Memoria libre**: Este parámetro indica el espacio libre en RAM para los procesos del sistema. Es uno de los parámetros más importantes en la monitorización ya que, es utilizada como memoria de trabajo donde se cargan todas las instrucciones que ejecuta la CPU, además de los diversos datos con el objetivo de acceder

a ellos de manera rápida. Por ejemplo, al iniciar una aplicación la información para funcionar se encuentra en la memoria RAM además de necesitar un espacio para guardar los datos que la propia aplicación ejecute mientras está iniciada. Por tanto, la memoria RAM y el procesador están relacionados, cuanto mayor sea el espacio libre en la RAM, mejor será el rendimiento del equipo ya que puede ejecutar mayor número de procesos.

- **Tiempo de entrada/salida** : Este valor indica cuál es el tiempo destinado a realizar operaciones de entrada y salida en el último intervalo de tiempo. Cuando se realiza una operación de este tipo es que existe una transferencia de datos entre el procesador y el dispositivo, además de una posterior sincronización entre ambos (seguramente, el dispositivo y la CPU tengan una velocidad de transferencia diferente). Es interesante conocer cuánto tiempo destina el procesador en realizar una operación de entrada salida en el mismo instante en que está monitorizándose. Si el valor es muy alto puede indicar que el procesador tiene un bajo rendimiento y por tanto, pueda destinarse a atender otras aplicaciones que requierán CPU.
- **Red**: Esta métrica indica la cantidad de tráfico en la red de comunicaciones, es decir, el número de bytes enviados y recibidos en la interfaz eth0, dónde los equipos están conectados a la red. Es interesante, porque sirve para conocer si la red está sobrecargada, de forma que el rendimiento del servicio no sea el deseado.

En este trabajo se van a monitorizar únicamente esos parámetros, pero, en un instante dado, puede surgirnos querer conocer el estado de otro componente o métrica. Modificar o introducir nuevos parámetros es un proceso muy sencillo utilizando Graphite, y habiendo monitorizado anterioremente otras métricas. Este proceso se explica en el capítulo 4.

3.4. Graphite

Para la monitorización de los recursos es necesario utilizar una herramienta de monitorización. **Graphite** es una herramienta de monitorización que permite mostrar gráficamente la monitorización de los recursos del sistema en tiempo real, además de contar con una fuerte escalabilidad, de forma que se eleve la carga de trabajo sin perder calidad de servicio.

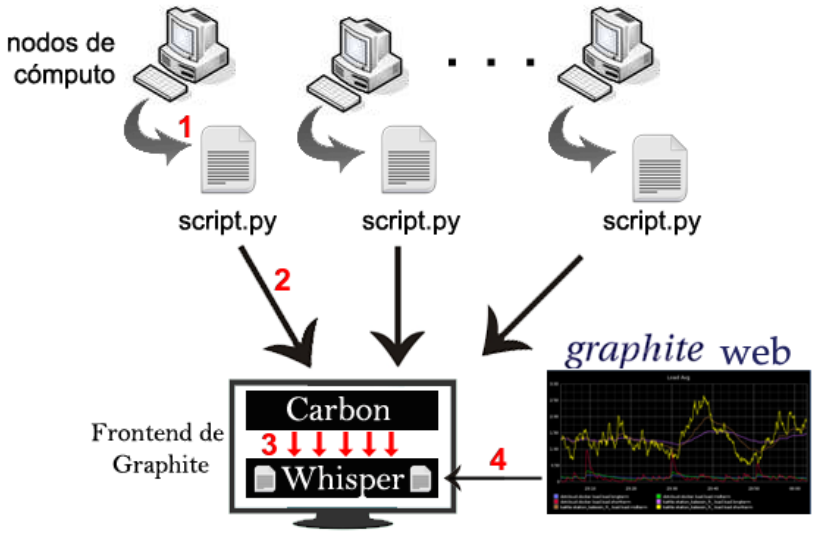


Figura 3.3: Esquema funcionamiento de Graphite

El funcionamiento de Graphite es el siguiente (**Figura 3.3**): el usuario escribe una aplicación donde se encargará de definir los recursos que se van a representar, seguidamente al ejecutar dicha aplicación **(1)**, los datos serán enviados al *back-end* de procesado **(2)**, *carbon*, que será el encargado de guardar dichos datos en la base de datos, *whisper* **(3)**. A partir de aquí, los datos pueden ser visualizados a través de las interfaces de *graphite webapp* **(4)**.

Graphite está compuesto de tres componentes:

1. **Carbon:** Demonio programado en Python que se mantiene a la escucha de datos de los agentes conectados a él y cuyo trabajo almacenar esos datos en la base de datos *whisper* para que puedan representarse inmediatamente. Originariamente, Carbon era un única aplicación ejecutada por *hilos*. Pero debido a problemas de concurrencia en *Python*, actualmente, está compuesto de tres procesos: **carbon-agent.py**, **carbon-cache.py** y **carbon-persister.py**. El primero se encarga de comenzar a ejecutar los otros dos procesos. Además, acepta conexiones mediante *sockets* y recibe el conjunto de datos en el formato apropiado. A partir de ahí envía los datos a *carbon-cache*, que guarda dichos datos en la *cache* donde son agrupados según las métricas elegidas. *Carbon-cache* intenta enviar la mayor suma de datos hacia *carbon-persister*. Este último lee todos los valores y los escribe en disco usando la base de datos *whisper*. Uno de los problemas en los que Graphite, en su conjunto, es muy potente es cuando existe demasiada carga en el sistema y *carbon* no tiene tiempo para enviar los datos desde cache a disco, y aun así, *graphite web* sigue actualizando en tiempo real todos sus gráficos. Esto se consigue gracias a que *Graphite Webapp* recupera los datos a mostrar desde la cache de carbon y disco simultáneamente. Graphite entonces combina las dos fuentes de datos en un solo valor para mostrarlos gráficamente.

2. **Whisper**: Es una base de datos de tamaño fijo, similar en diseño a RRD [11] *(Round-Robin-Database)* que almacena los datos de forma rápida en todo instante de tiempo. Los valores enviados a *Whisper* son guardados en disco. Cada valor es almacenado con su correspondiente contador de tiempo en segundos desde *Unix Epoch* (01-01-1970). La base de datos *Whisper* está compuesta de uno o más ficheros (según el conjunto de métricas que se quiera medir) con una retención y resolución específica. Retención puede estar definido como el conjunto máximo de valores que se puedan guardar o el intervalo de tiempo que transcurre entre cada valor guardado. Por ejemplo, podemos tener un archivo con una retención de 60 segundos, lo que significaría que cada valor será guardado cada 60 segundos de tiempo. El tiempo de retención total será determinado por el archivo con mayor periodo de retención, es decir, si tenemos los archivos, uno con una mayor retención que el otro, la retención total no será la suma de los dos valores sino que será el máximo valor de los dos valores.
   Como se ha dicho anteriormente, carbon tenía la función de leer datos y enviarlos a disco a través de whisper. En este caso, *whisper* es ineficiente en su uso de disco debido a lo siguiente:

   - Cada dato es almacenado con su correspondiente *timestamp* para que en el proceso de recuperación de datos se pueda confirmar la validez de cada uno de los datos. Pero en el caso en el que el *timestamp* no corresponda con el valor esperado, se devolverá un valor nulojunto con un *timestamp "out of date"*.
   - Cuando Whisper escribe en la base de datos que contiene múltiples archivos, conlleva que los datos sean escritos en todos los archivos al mismo tiempo en cada uno de los periodos de retención. Esto provoca que se solapen los periodos de retención.
   - Todos los periodos de tiempo ocupan un espacio en disco aunque en algunos de ellos no existan valores guardados.

   Inicialmente Graphite usaba RRD para el almacenamiento. Pero existen principalmente dos razones por las que Graphite acabó creando su propia base de datos whisper.

   - La primera razón es que aunque ambas tengan un diseño similar, RRD inserta datos de forma regular, mientras que Graphite facilita la visualización de las métricas que no siempre ocurren regularmente. Por ejemplo, cuando de repente se lanza un evento e inmediatamente se maneja y se mide su latencia para ser enviado a Graphite.
   - La segunda razón es el rendimiento. Aunque RRD sea mucho más rápido que whisper, solo te permite insertar un único valor al tiempo,mientras que whisper puede insertar múltiples datos mediante una única operación de escritura.

3. **Graphite Webapp:** Es un sistema de representación web de gráficas en tiempo real. *Graphite Webapp* es una aplicación *Django* que se ejecuta bajo el módulo *mod_wgsi* de apache [12]. *Django* es un *framework* de desarrollo web de código abierto que tiene como función principal facilitar la creación de sitios webs complejos. *Graphite Webapp* tiene como función crear una URL para la representación gráfica, en cada instante de tiempo si así se configura, de los datos recuperados desde whisper. Además compone una interfaz web donde se representarán las métricas definidas y permite al usuario construir *dashboards* (**Figura 1.3**) que, posteriormente, pueden ser guardados. La interfaz *Dashboard* es una herramienta que permite al usuario desplegar varios gráficos simultáneamente donde todos ellos muestran los mismos valores de tiempo. Una vez construído el panel permite guardarlo para posteriores monitorizaciones.

3.5. K-Means

K-means es uno de los algoritmos más usados en *Clustering* [9]. Esta técnica analiza los datos agrupando n puntos en k *clusters* de la siguiente manera:

- Se colocan aleatoriamente k puntos iniciales llamados *centroides*.
- Se asigna cada punto a su *centroide* más cercano.
- Y por último se recalcula el punto de los centroides hasta el centro del conjunto de puntos que se le han asignado. El proceso de asignación de puntos y cálculo de *centroides* es repetido hasta el momento en que el punto de los k *clusters* y cada conjunto de datos se estabilizan.

A la hora de utilizar k-means hay que tener clara la idea de *distancia*. Sobre todo, cuando se necesite decidir cómo dividir el conjunto de datos en grupos y en cuántos grupos se van a particionar, teniendo en cuenta que los datos estarán más cerca de los miembros de su propio grupo que de los miembros de los demás grupos. La distancia usada por k-means,en nuestro caso es la Eucídea [14] ya que para otros casos prácticos pueden ser más útiles otras. La distancia Euclídea entre dos puntos P_1 y P_2, con coordenadas cartesianas (x_1,y_1) y (x_2,y_2) es d_E (P_1, P_2)= $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Por tanto, la distancia se va a tener como punto importante a la hora de elegir el conjunto de métricas en el análisis de *clusters*, tanto que va a ser más fundamental que el valor de los propios puntos.

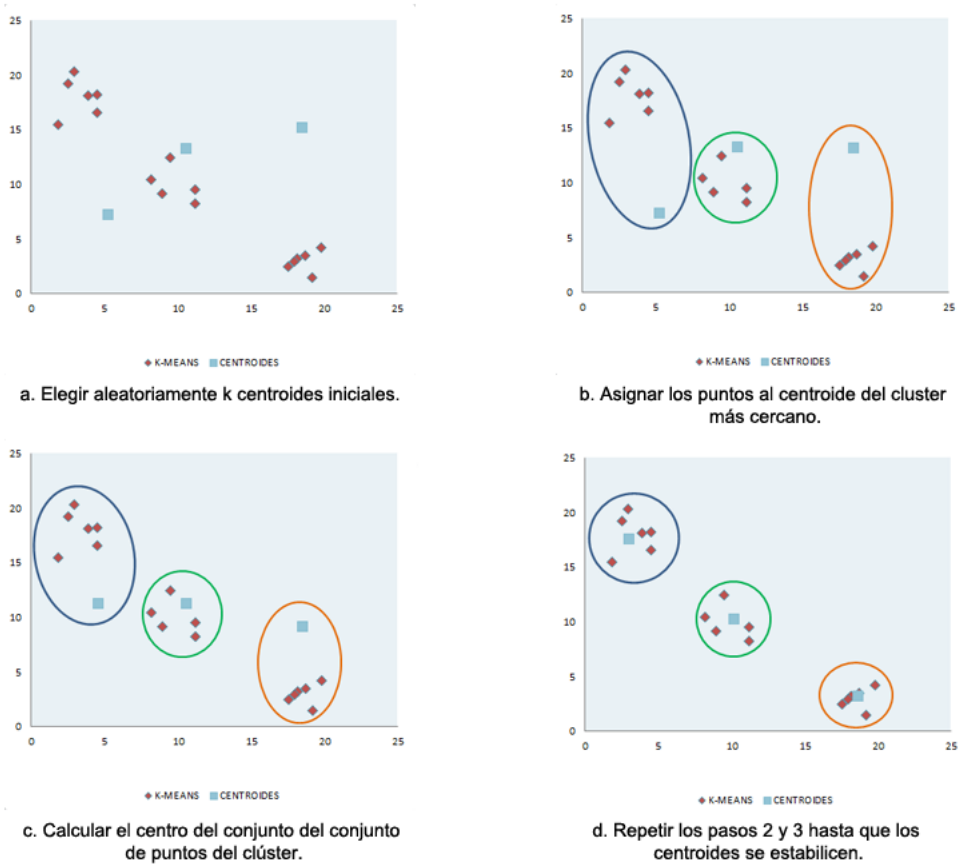


Figura 3.4: Funcionamiento del algoritmo K-MEANS

En la **figura 3.4**, los puntos que se van a agrupar (en este trabajo, los equipos que se monitorizan) tienen forma de rombo mientras que los centroides de los clústers son los cuadrados.

El funcionamiento del algoritmo es el siguiente:

1. Se eligen aleatoriamente k *centroides* iniciales $c_1, c_2, ..., c_k$ (**Figura 3.4.a**).

2. Asignar cada uno de los $1 \leq i \leq k$ *centroides* al conjunto de los n puntos en función de que el *centroide* c_i sea un punto más cercano que c_j cuando $j \neq i$(**Figura 3.4.b**).

3. Para cada $1 \leq i \leq k$ *centroides*, calcular el centro del conjunto de puntos del grupo C_i en función de $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$(**Figura 3.4.c**).

4. Repetir los pasos 2 y 3 hasta que los *clusters* C_i y sus *centroides* c_i se estabilicen. Por tanto la partición de X es el conjunto de *clusters* $C_1, C_2, ..., C_3$ (**Figura 3.4.d**).

Este algoritmo puede provocar dos situaciones no deseadas: La primera es que un *centroide* no tenga asignado puntos, por lo que terminaría eliminándose. De ahí, se obtendría un conjunto de puntos con menos de los k *clusters* iniciales posibles. El segundo caso, es cuando un punto tiene la misma distancia hacia un *centroide* y otro. Esto haría que el punto sea asignado arbitrariamente a uno de los *clusters*.

Capítulo 4

Implementación

Este capítulo se estructura en tres secciones donde se explica la instalación y contrucción de todo el software, tanto la configuración de los componentes de *Graphite* como la estructuración de los *scripts* de métricas y la implementación del algoritmo *k-means*. Para reflejar el proceso de Implementación se muestra el diagrama de flujo de la **figura 4.1** que se explica a lo largo del capítulo.

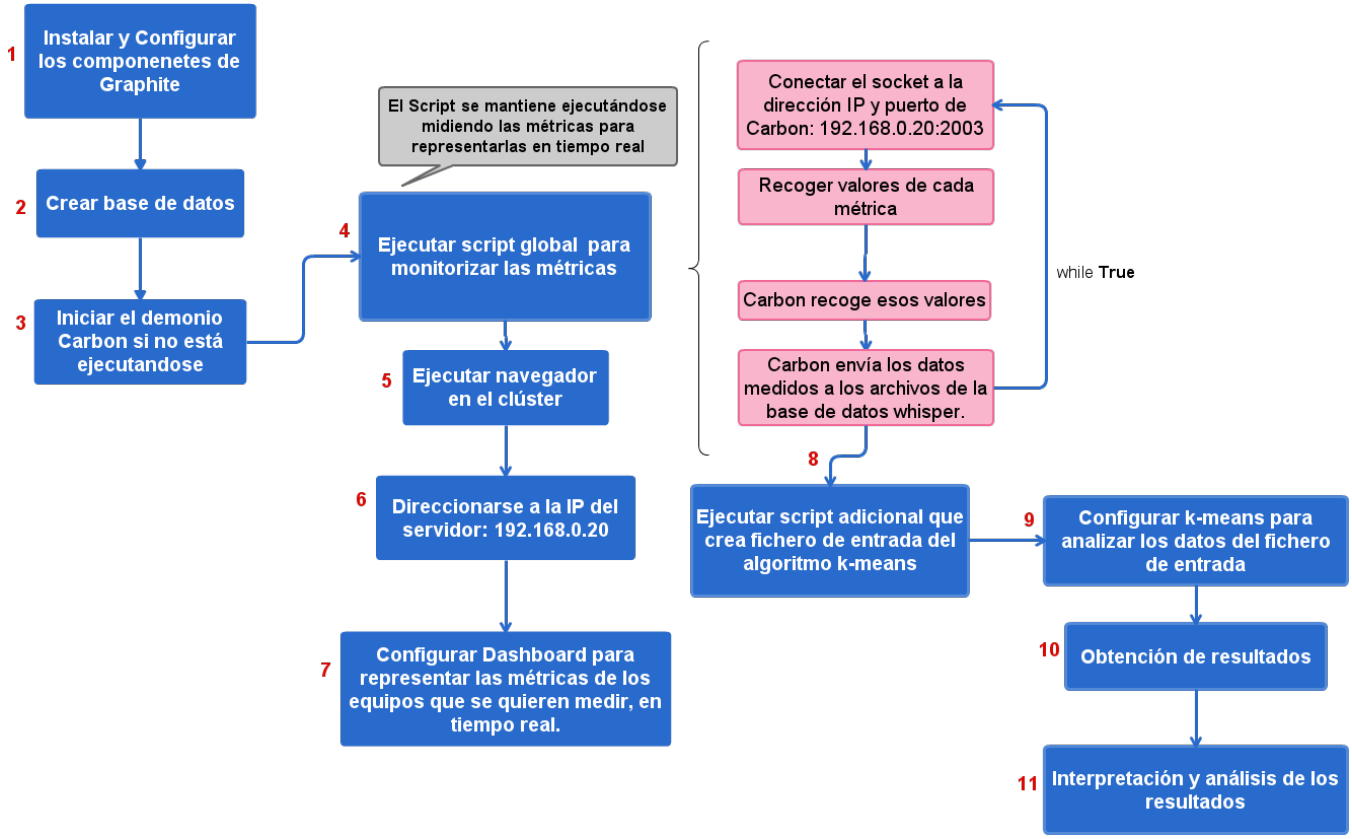


Figura 4.1: Diagrama de flujo - Implementación

4.1. Instalación y Configuración de Graphite

En esta primera sección, se explica cómo configurar **Graphite** e integrar sus componentes, *carbon*, *whisper* y *graphite webapp* para obtener la información de estado de las métricas y representarla gráficamente en tiempo real.

Antes de comenzar la configuración de Graphite se deben descargar e instalar sus tres componentes **(1)**. Como ya se definió en el capítulo 3 anterior:

- **Carbon** es un demonio ejecutándose en *background* que recibe datos de los agentes conectados a él y los envía a la base de datos.
- **Whisper** es la base de datos que almacena los datos cuando los recibe de carbon.
- Y **Graphite webapp** representa gráficamente, en tiempo real, los datos recogidos en la base de datos.

Además, graphite depende de otros dos componentes para funcionar correctamente: **Apache** y **Diango** que, igualmente, deben instalarse y configurarse. Apache es el servidor web dónde se implementa la aplicación web de graphite, y Django es una aplicación web que programa graphite webapp para guardar los datos.

Configurar tiempo de retención de Carbon

La base de datos de whisper está compuesta por archivos correspondientes a las métricas que se miden. Estos archivos almacenan dos valores: un contador de tiempo y el dato obtenido de la medición. Pero, ¿cada cuánto periodo de tiempo se almacenan esos datos? A este periodo de tiempo se le denomina **retención**. Para determinar la retención se configura el fichero **/opt/graphite/conf/storage-schemas.conf** (**Figura 4.2**).

```
[carbon]
pattern = ^carbon\.
retentions = 60:90d

[default_1min_for_1day]
pattern = .*
retentions = 60s:1d
```

Figura 4.2: Configuración tiempo de retención Carbon

La primera sección decide qué hacer con los datos que le llegan a carbon, mientras que la segunda se designa a aquellas métricas que no se recogen en cualquier otra sección. Las palabras entre los corchetes son las cabeceras de cada entrada donde se define un patrón, ***'pattern'***, y la política de retención, ***'retentions'***. *Pattern* es una expresión que se compara con información que llega a Carbon, incluyendo el nombre de la métrica medida. En la primera entrada, pattern comprueba si la métrica comienza con "carbon. ". La retención está definida por un conjunto de números seguido de dos puntos y otro conjunto de números. El primer conjunto consiste en el intervalo de tiempo que transcurre entre cada almacenamiento de datos, mientras que el segundo conjunto define el tiempo que se mantienen guardados dichos datos. Así, en la segunda entrada, los datos se almacenan cada 60 segundos y se mantienen guardados en la base de datos durante 1 día. Hay métricas que se guardan en varios archivos pero con una retención distinta. Como ejemplo, para ilustrarlo, nosotros mismos podemos crear una nueva sección en el fichero (**Figura 4.3**):

```
[stats]
pattern = ^stats.*
retentions = 10:2160,60:10080,600:262974
```

Figura 4.3: Configuracion tiempo de retención nueva métrica stats

Aquellas métricas que comienzan con 'stats' se guardan en tres ficheros distintos. El primer archivo tiene como retención *10:2160*, así que guarda datos cada 10 segundos y se mantienen durante 36 minutos (2160s). El siguiente archivo reune los datos del último minuto almacenándolos durante casi tres horas. Si se quieren representar datos de los últimos 30 minutos se obtienen los valores del primer fichero. Si se representan los datos a partir de 1 hora se obtienen del segundo.
Por defecto, Carbon hace la media de los datos que le llegan para crear el valor de la medición en ese periodo de tiempo. También se pueden crear métodos que suman, restan o devuelven el último valor de ese periodo de retención en el fichero */opt/graphite/conf/storage-aggregation.conf*.

En el fichero de configuración de Carbon */opt/graphite/conf/carbon.conf*, se pueden modificar parámetros correspondientes a: directorio donde se instala graphite, directorio de almacenamiento de los archivos de la base de datos donde se guarda cada valor de las distintas métricas, el máximo número de archivos creados por minuto, habilitar interfaz y puerto por conexión UDP, etc. Para nuestro trabajo se dejan los parámetros por defecto.

Ahora, se configura **la aplicación web de Graphite**. Para ello, se pueden modificar algunos parámetros del fichero de configuración */opt/graphite/local_settings.py*. Por defecto, todo el fichero está comentado, aunque es recomendable que el parámetro *SECRET_KEY* no esté vacío, sino definido por una cadena larga de carácteres.

Después de configurar Carbon y Graphite Web **se crea la estructura de la base de datos (2)**.

```
$ cd /opt/graphite/webapp/graphite/
$ sudo python manage.py syncdb
```

Antes de finalizar es necesario **ejecutar Carbon (3)** para que se mantenga a la escucha de los datos que le llegan de los agentes conectados a él.

```
$ sudo cd /opt/graphite/bin/
$ ./carbon-cache.py start
```

Si al ejecutar el navegador y direccionarse a la IP del dominio donde se encuentra graphite en el clúster, 192.168.0.20, no encuentra dicho servidor, es que el no se encuentra en funcionamiento. Para ejecutarlo:

```
$ cd /opt/graphite/webapp/graphite/
$ sudo python manage.py runserver
```

Una vez que Graphite ya está listo para monitorizar se ejecuta el script, explicado en la sección 4.2.2, que mide los parámetros. Mientras este script se ejecuta y se almacenan los datos en los archivos de whisper, la aplicación web de Graphite puede mostrarlos gráficamente. Para ello, se debe ejecutar el navegador dentro del clúster **(5)** y direccionarse a la IP donde se encuentra instalado el servidor de Graphite **(6)** en nuestro clúster, 192.168.0.20, donde aparece la interfaz de graphite donde configurar qué gráficas se quieren representar. Para ello, hay que configurar el panel de control **(7)**, *dashboard*, según las métricas de los equipos que se quieran supervisar.

4.2. Monitorización de las métricas

Después de haber definido las métricas que se monitorizan en el capítulo de Diseño hay que conocer de qué ficheros se van a obtener los datos. Además, teniendo la base de datos creada y *Carbon* en funcionamiento, es necesario construir un script o scripts para que *carbon* obtenga los datos de las medidas y los envíe a *whisper*.

4.2.1. Análisis del directorio /proc

El **kernel** de Linux tiene dos funciones principales: controlar el acceso a los dispositivos físicos de un equipo y establecer, cuándo y cómo, los procesos interactuarán con estos dispositivos. El directorio /**proc**/ — también llamado el sistema de archivos proc — contiene una jerarquía de archivos especiales que representan el estado actual del *kernel* — permitiendo a las aplicaciones y usuarios ver detalladamente cómo está el *kernel* del sistema [15].
Dado que *proc* no es un sistema de ficheros real, sino virtual, no consume ningún espacio de almacenamiento en disco, y solo ocupa una limitada cantidad de memoria, que es donde el kernel crea este sistemas de ficheros. Que este sistema se encuentra en memoria facilita su acceso ya que el tiempo requerido es menor que si se almacenase en disco.
La mayoría de estos archivos virtuales tienen un tamaño de 0 bytes. Sin embargo, cuando se visualiza el archivo puede contener una gran cantidad de información. Además, la mayoría de configuraciones del tiempo y las fechas reflejan el tiempo y fecha real, lo que es un indicativo de que están siendo constantemente modificados.

Dentro del directorio /proc/, se puede encontrar una gran cantidad de información con detalles sobre el hardware del sistema y cualquier proceso que se esté ejecutando actualmente. Gracias a este sistema de ficheros, nosotros somos capaces de extraer datos sobre características del kernel en un momento dato y controlar el sistema. Por

tanto, para obtener las medidas de los cuatro parámetros definidos es necesario acceder al directorio proc en cada uno de los nodos del clúster.

- **Carga de CPU**: Para obtener información acerca de este parámetro, se necesita acceder al fichero /**proc**/**loadavg**. Dicho archivo contiene tres primeros valores: carga promedio durante el último minuto, los cinco últimos minutos y los diez últimos minutos. Estos tres valores no indican el procentaje de uso de la CPU sino el número de procesos que están ejecutándose en esos momentos. De esos valores solo se recoge el primero de ellos ya que es el más proximo al valor en tiempo real.
- **Memoria libre**: Para obtener datos acerca de este parámetro se necesita acceder al archivo /**proc**/**meminfo** que contiene información de la memoria libre y usada en el sistema, así como la memoria compartida y los *buffers* utilizados por el núcleo. Sólo interesa el valor de la segunda línea, correspondiente a la memoria disponible en RAM para el sistema.
- **Tiempo de entrada/salida**: Este valor se obtiene del fichero /**proc**/**diskstats**. Este archivo contiene estadísticas de entrada y salida para cada uno de los dispositivos de disco. Cada una de las líneas corresponde a valores de un sólo dispositivo, como su nombre, las lecturas o escrituras completadas con éxito, etc. El valor 13 de cada línea indica el tiempo destinado a realizar operaciones de entrada salida en ms. En este trabajo solo se va a requerir el valor de tiempo en la partición **sda** o **xvda1**, según el nodo del clúster donde se esté monitorizando. En esta partición está instalado el sistema en los equipos.
- **Uso de Red**:Este valor es obtenido del fichero /**proc**/**net**/**dev** que contiene información acerca del estado de cada una de las interfaces de red de las que se compone un equipo, como el número de paquetes o de bytes enviados y recibidos, el número de errores y colisiones, etc. Cada línea representa una interfaz, por tanto se escogerán los valores referidos a los bytes recibidos/enviados a través de la interfaz eth0.

4.2.2. Scripts de medición de métricas

Conociendo de qué archivos obtener información de cada métrica, es necesario realizar un fichero escrito en Python, debido a que es el lenguaje usado por Graphite, que obtenga datos de cada una. Inicialmente se puede crear un *script* por cada una de las métricas, sobre todo por flexibilidad ya que puede ocurrir que haya medidas que se quieran monitorizar más frecuentemente que otras. Para simplificar el proceso y debido a que las pruebas realizadas vienen dadas por la medición de las cuatro métricas al mismo tiempo, se crea un único fichero que contenga todo lo referente a la obtención y el envío a *whisper* de los valores de los cuatro parámetros, simultáneamente.
En esta sección, se detalla qué partes debe contener el fichero para obtener los valores de cada parámetro en cada instante de tiempo, y posteriormente guardarlos en la base de datos *whisper*. El *script* tiene la siguiente estructura, aunque la función que obtiene las medidas y la creación del array que se envía a whisper es independiente para cada métrica (estas partes vienen ejemplificadas con la carga de cpu, para entender mejor el funcionamiento):

1. Importar una serie de librerías como *sys*, *time* o *socket* de las que dependen algunas funciones del *script* para su correcto funcionamiento (**Figura 4.4**).

```
#!/usr/bin/python
import sys
import time
import os
import platform
import subprocess
from socket import socket
```

Figura 4.4: Importar librerías

2. Definir la IP y el puerto *carbon* dónde va a estar escuchando peticiones de los agentes que están monitorizandose y, posteriormente, almacena los datos que le envían en la base de datos (**Figura 4.5**).

```
CARBON_SERVER = '192.168.0.20'
CARBON_PORT = 2003
```

Figura 4.5: Definir IP Carbon

3. Definir el período de tiempo entre cada ejecución del *script* (**Figura 4.6**).

```
delay = 3
if len(sys.argv) > 1:
  delay = int( sys.argv[1] )
```

Figura 4.6: Definir periodo de tiempo

4. Definir la función que obtiene los valores de cada métrica desde su fichero correspondiente en *proc* (**Figura 4.7**).

```
def get_loadavg():
    return open('/proc/loadavg').read().strip().split()[:3]
```

Figura 4.7: Definir función que obtiene datos

5. Definir la función que obtiene el *hostname* del equipo que ejecuta el script (**Figura 4.8**).

```
def get_hostname():
        with open ("/etc/hostname", "r") as myfile:
              data=myfile.read().replace('\n', '')
        return data
```

Figura 4.8: Definir función que obtine hostname

6. Inicializar y conectar el socket que comunica carbon y whisper. Este socket acepta las peticiones enviadas a *carbon* y permite el envío de los valores a la base de datos (**Figura 4.9**). Es necesario que el demonio *carbon-agent.py* esté ejecutándose para que carbon escuche las peticiones que se le envían.

```
sock = socket()
try:
  sock.connect( (CARBON_SERVER,CARBON_PORT) )
except:
  print "Couldn't connect to %(server)s on port %(port)d, is
  carbon-agent.py running?" % { 'server':CARBON_SERVER,
  'port':CARBON_PORT }
  sys.exit(1)
```

Figura 4.9: Conexión socket

7. Obtener los datos y enviarlos a la base de datos *whisper* cada periodo de 3s. En este apartado, se ejemplifica el envío con la carga de cpu (**Figura 4.10**). Primero, se crea el array de valores, *lines*, que componen el mensaje que se envía a *whisper*. A continuación, se hace una llamada a la función que obtiene las medidas y el *hostname* del equipo. Colocar el *hostname* permite guardar cada valor en diferentes archivos dentro de un directorio diferente para cada equipo cuyo nombre será el *hostname* de los nodos monitorizados. Si no se colocara, cada nodo que vaya ejecutando el script iría sobreescribiendo los archivos. Al final, se crea el mensaje compuesto por el array *lines*. El socket se encarga de enviar el mensaje al archivo de la base de datos.

```
while count<200:
  now = int( time.time() )
  lines = []
  #We're gonna report all three loadavg values
  loadavg = get_loadavg()
  hostname=get_hostname()
  lines.append("%s.loadavg_1min %s %d" % (hostname,loadavg[0],now))
  message = '\n'.join(lines) + '\n' #all lines must end in a newline
  print "sending message\n"
  print '-' * 80
  print message
  print
  sock.sendall(message)
  time.sleep(delay)
  count=count+1
```

Figura 4.10: Obtener datos y enviarlos a la base de datos

Como se ha visto en la estructura, existen dos apartados independientes para cada métrica, pero al simplificar la monitorización en un único script, se definen las cuatro funciones y se envían los datos de cada métrica en ese mismo fichero.

- La **carga de cpu** que se mide es la total de todos los procesadores que contiene el equipo monitorizado. Dependiendo del conjunto de nodos que se esté monitorizando en un instante dado, hay que dividir la carga entre el número de cores de los que se compone dicho computador, debido a que existen grupos de equipos con mayor número de procesadores que otros y no sería una medición equitativa entre todos los equipos. Ésto se va a tener en cuenta cuando se recogan los datos de la base de datos, al crear el fichero de entrada del algoritmo.
- Para la memoria libre se añade lo correspondiente a la definición de la función que obtiene el valor de la memoria libre desde el fichero */proc/meminfo* (**Figura 4.11 y 4.12**):

```
def get_memfreeabs():
          file = open('/proc/meminfo', 'r')
          for num, line in enumerate(file):
             if num==1:
                print line
                memabs= ([int(x) for x in line.split() if x.isdigit()])
          return memabs
          file.close()
```

Figura 4.11: Función memoria

```
memfree = get_memfreeabs()
lines.append("%s.MemFree %s %d " % (hostname,memfree[0],now))
```

Figura 4.12: Recoger datos y enviar a whisper - Mem

Los valores de carga de CPU y memoria libre son absolutos, pero el tiempo de entrada y salida y el uso de red son relativos a cada intervalo de tiempo de monitorización.

- En el caso del **tiempo** de entrada/salida hay que tener en cuenta que el valor que devuelve el fichero *diskstats* de proc es el correspondiente al conjunto de tiempos sumados desde una fecha de referencia (**Figura 4.13 y 4.14**). El valor que se quiere medir es el tiempo desde la última vez que se ejecutó el script. Por tanto se crea una nueva variable, *tiempo*, que contenga la diferencia entre el dato de tiempo de esta nueva ejecución, obtenido del fichero, y el valor de tiempo de la anterior ejecución. Este resultado es el número que se guarda en la base de datos.

```
def get_tiempoiosda():
        file = open('/proc/diskstats', 'r')
        tiempo=[]
        for line in file:
                array=line.split()
                if array[2]=="sda" or array[2]=="xvda1":
                        print [float(x) for x in line.split() if
                         x.isdigit()]
                        tiempo.append(([float(x) for x in
                        line.split() if x.isdigit()])[11])
        print tiempo
        return tiempo
        file.close()
```

Figura 4.13: Función tiempo

```
tiemposda = get_tiempoiosda()
tiempo=tiemposda[0]-tiempoant
lines.append("%s.TimeIOsda %s %d " % (hostname,tiempo,now))
tiempoant=tiemposda[0]
```

Figura 4.14: Recoger datos y enviar a whisper - Tiempo

- Para la medición de **red** es necesario tener una nueva variable que contenga la diferencia entre los bytes enviados y recibidos en la última ejecución del *script* y la anterior ya que los bytes son sumados desde una fecha de referencia (**Figura 4.15 y 4.16**). De momento se crean dos variables independiantes para los bytes que se envían y los bytes que se reciben a través de la interfaz, *bytesseneth0* y *bytesreceth0*.

```
def get_paquetes():
        file = open('/proc/net/dev', 'r')
        array=[]
        for line in file:
           array2=line.split()
           if "eth0" in array2[0]:
                columns= line.strip('    eth0:')
                columns=columns.split()
                array.append(([float(x) for x in columns])[0])
                array.append(([float(x) for x in columns])[8])
        return array
        file.close()
```

Figura 4.15: Función red

```
paquetes = get_paquetes()
bytessendeth0=paquetes[0]-bytessendeth0
bytesreceth0=paquetes[1]-bytesreceth0
lines.append("%s.BytesSendeth0 %s %d " % (hostname,bytessendeth0,
             now))
lines.append("%s.BytesReceth0 %s %d " % (hostname,bytesreceth0,now))
bytessendeth0=paquetes[0]
bytesreceth0=paquetes[1]
```

Figura 4.16: Recoger datos y enviar a whisper - Red

Con Graphite configurado y el script list, ya se puede ejecutar dicho script para ir monitorizando las métricas**(4)**.

4.2.3. Añadir nuevas métricas

En este trabajo nos hemos limitado a explicar cómo monitorizar las cuatro métricas definidas. Pero si en trabajos posteriores, se requiere conocer el estado de otro parámetro, ya sea porque, en un momento dado, ese recurso imposibilita el buen funcionamiento del sistema o por las mismas razones que se eligieron las cuatro métricas es necesario saber cómo extraer su información. Conociendo cómo monitorizar una métrica, este proceso es trivial ya que los pasos seguidos son los mismos. Como se comentó anteriormente, sólo depende de la frecuencia de monitorización del parámetros. Si solo se necesita conocer su estado cada hora se crea un nuevo script donde se ajusta la variable *delay* a ese período, y se construye el nuevo script siguiente los pasos explicados en el apartado anterior. Ya se conocen aquellos pasos comunes a cada métrica por lo que solo es necesario modificar, nuevamente, la función que extrae su información y el envío de los datos a su archivo en whisper.

Por ejemplo, se requiere conocer cuál es el número de procesos que se ha ejecutado en cada uno de los procesadores, que forman un equipo, desde si primer arranque. Es una medida que puede ser útil para comprobar si hay *cores* que ejecutan más cantidad de trabajo que el resto, aunque sean procesos que requiran menos complejidad del sistema. En base a ello, se comienza analizando los ficheros del *proc* que devuelvan estadísticas acerca de la actividad del kernel, más concretamente, de la cpu. Los datos buscados se encuentra en el fichero */proc/stat* donde se agrega información general acerca del sistema, como procesos ejecutados en total, o divididos entre cada procesador, desde el primer arranque, procesos ejecutados y bloqueados actualmente, etc. La primera línea corresponde a valores totales de la cpu, es decir, sumando los números de todos los cores. Esos números corresponcen a los procesos que se han ejecutado desde su primer arranque, aquellos que se encuentran esperando para que la entrada/salida se complete, el número de interrupciones, etc. Las dos siguientes líneas corresponden a los valores concretos de cada procesador. Por tanto, hay que acceder a la primera columna, de cada una, para conocer el número de procesos ejecutados. El número de esa columna, por cada procesador, es el extraido por la función en el script que es enviado al nuevo archivo creado en la base de datos para esta nueva métrica.

4.3. Proceso de extracción de la información

En esta sección se define el proceso de extracción de la información de los datos monitorizados. En general, *k-means* divide el conjunto de nodos en varios grupos según si las características medidas de un subconjunto de equipos tienen una gran correlación.

1. **Creación del fichero de datos:**
   Llegados a este punto, la base de datos whisper, de la que se compone graphite, va a contener un conjunto de archivos independientes para cada métrica. Además, se ha creado un directorio para cada nodo monitorizado, por lo que el conjunto de archivos es independiente en cada equipo. K-means, necesita un fichero de entrada que contenga las medidas de los parámetros monitorizados en cada nodo. Para obtener los datos de los archivos de whisper y construir el fichero de entrada es necesario construir un *script* adicional **(8)**. Este fichero se estructura de la siguiente forma:

   - Recoger el último dato almacenado en los archivos de la base de datos correspondientes a las cuatro métricas de los directorios de cada nodo. Para visualizar los datos almacenados en whisper, es necesario utilizar uno de los scripts incluidos en Graphite, *whisper-fetch.py*. En la figura solo se refleja la extracción de datos de la carga de cpu pero hay que realizar lo mismo para el resto de métricas(**Figura 4.17**).

```
whisper-fetch.py loadavg_1min.wsp | grep -v None | tail -n1
        | awk '{print $2}' > $HOME/results/resultsload.txt
```

Figura 4.17: Obtener último dato del fichero de whisper - Ejemplo carga CPU

   - Crear las variables correspondientes a cada una de las medidas. La carga de cpu está condicionada por el grupo del clúster al que el nodo pertenece ya que, lo que se quiere medir es el uso de cpu por core en cada equipo (**Figura 4.18**). Por tanto, hay que dividir ese valor entre cuatro, ocho o doce según el nodo que está ejecutándose. La variable de red es la suma de los bytes enviados y recibidos.

```
if [ $count2 -lt 21 ]
then
      load=`cat $HOME/results/resultsload.txt`
      result=$(awk "BEGIN {printf \"%.4f\",${load}/${cuatro}}")
fi
```

Figura 4.18: Crear variables de cada métrica - Ejemplo carga CPU

- Escribir los datos en el fichero (**Figura 4.19**). Cada línea del nuevo fichero contiene como primera columna el nombre de cada nodo, seguido por los cuatro valores medidos que corresponden a las cuatro columnas contiguas.

```
echo "$i $result $mem $time2 $bytestot" >> $HOME/resultstotal
```

Figura 4.19: Escribir datos en el fichero

Ejecutando el *script* adicional, en el *frontend* de *Graphite*, se crea el nuevo fichero, que se pasa como entrada a k-means **(9)**.

2. **Introducir a k-means el fichero de datos como fichero de entrada (9)**:
Una vez construido el fichero de entrada de *k-means*, hay que definir algunos parámetros adicionales para poder extraer los datos del fichero y particionarlos en varios grupos, antes de ejecutar el algoritmo.

   - Primero se deben inicializar los cuatro arrays que corresponden a los datos del fichero de entrada, es decir, el primer array son los valores de la carga de cpu de todos los nodos, el segundo array son los datos de la memoria libre, y así los dos últimos respecto al tiempo de entrada/salida y el tráfico de red.

   - El siguiente paso es definir el conjunto de puntos que corresponden a cada uno de los nodos monitorizados. Cada punto está formado sus cuatro métricas correspondientes del fichero de entrada, normalizados. Antes de definir el punto hay que normalizar cada valor. Recorriendo el array de carga de cpu, por ejemplo, se sustituye el mayor de todos los nodos, *max*, por 1. El resto de medidas de cpu y se sustituyen por *result*, de forma que $result = y * max$. Los arrays del resto de métricas se normalizan de la misma manera.
   - Por último, se definen las dimensiones de cada punto correspondiente al número de parámetros del nodo y el número *"k"* de agrupaciones en las que se divide el conjunto. Las dimensiones y el número de clusters se van modificando a medida que se realiza cada una de las pruebas, según si los nodos se miden a partir de dos, tres o cuatro métricas, o si se divide el conjunto entre dos, tres y cuatro grupos.

3. **Configurar el valor de "k"para el algoritmo k-means**:

   Antes de ejecutar k-means hay que definir el número de agrupaciones óptimo para el conjunto de nodos que se monitorizan. Cuando se ejecuta k-means, puede darse la situación de que el número de nodos en los grupos sea equitativo o, por el contrario, alguno de los grupos contenga una pequeña cantidad de puntos. Ésto son resultados normales, pero suelen ocurrir cuando se añaden más clusters, sobre todo si se escoge un número de clústers inadecuado para una baja cantidad de computadores monitorizados. Se observa, de manera más clara, en el apartado 5.1 de Pruebas.
   En CPD de empresas que se componen de gran cantidad de equipos, determinar el número k de clusters es un problema. La correcta elección de k es, a menudo, ambigua ya que puede depender de la forma de la distribución de los puntos en un conjunto de datos o de cómo desee el usuario que se distribuyan las agrupaciones. Entonces, para lograr la cantidad óptima de grupos debe existir un equilibrio entre la máxima compresión de los datos utilizando un solo grupo, y la máxima precisión mediante la asignación de cada punto de datos a su propio agrupamiento. Si un valor adecuado de k no es evidente a partir de un conocimiento previo de las propiedades del conjunto de datos, debe ser elegido de alguna manera [16]. Existen una serie de algoritmos que ayudan con la determinación de k, pero quedan fuera del presente trabajo dado que es sencillo determinar el número "k"según la baja cantidad de nodos monitorizados. Es por ello, que después de observar el número de equipos monitorizados, en el capítulo 3, se ha llegado a la conclusión de que el número óptimo de grupos para realizar el *clustering* sea 2, 3 y 4. Así, en el siguiente capítulo se comprueba cuales son los resultados y si es correcta dicha elección.

4. **Ejecutar el algoritmo (10)**:

   Llegados a este punto, y con todos los parámetros definidos, ya estamos listos para ejecutar el algoritmo. Una vez ejecutado, los resultados obtenidos son como los que se muestran en la **figura 4.20**. Como se ve en la figura 4.20, la ejecución devuelve el número de iteraciones en las que converge el algoritmo, el valor del *centroide* para cada agrupación y el conjunto de nodos, con sus respectivos valores de los 2, 3 o 4 parámetros, que se encuentran en cada grupo. Recordar que, cada uno de los parámetros están normalizados, es decir, se encuentran en un rango entre 0 y 1.

```
Converged after 1 iterations
Cluster  0  :  [0.483, 0.702, 0.045, 0.064]
 Cluster:  0 compute-4-64        : [0.0833, 18205844.0, 0.0, 1043.0]
 Cluster:  0 compute-4-65        : [0.1675, 16903548.0, 0.0, 3316.0]
 Cluster:  0 compute-4-66        : [0.0833, 18858352.0, 0.0, 1135.0]
 Cluster:  0 compute-4-67        : [0.0833, 18963144.0, 0.0, 861.0]
 Cluster:  0 compute-4-68        : [0.34, 28784680.0, 0.0, 902.0]
 Cluster:  0 compute-4-69        : [0.3333, 22241720.0, 0.0, 3843.0]
 Cluster:  0 compute-4-70        : [0.5, 27091580.0, 16.0, 6031.0]
 Cluster:  0 compute-4-71        : [0.25, 29612744.0, 4.0, 4540.0]
 Cluster:  0 compute-4-72        : [0.3333, 26195628.0, 0.0, 1798.0]
 Cluster:  0 compute-4-73        : [0.25, 26265668.0, 0.0, 1718.0]
 Cluster:  0 compute-4-74        : [0.5, 21633768.0, 19.0, 1326.0]
 Cluster:  0 compute-5-100       : [0.3392, 31626152.0, 0.0, 2430.0]
 Cluster:  0 compute-5-101       : [0.4308, 25208300.0, 0.0, 1195.0]
 Cluster:  0 compute-5-102       : [0.25, 36064724.0, 0.0, 7656.0]
 Cluster:  0 compute-5-103       : [0.4017, 18351760.0, 0.0, 4715.0]
 Cluster:  0 compute-5-84        : [0.5, 22114440.0, 0.0, 1525.0]
 Cluster:  0 compute-5-85        : [0.3333, 19910824.0, 12.0, 5299.0]
 Cluster:  0 compute-5-86        : [0.255, 18148020.0, 0.0, 857.0]
 Cluster:  0 compute-5-87        : [0.2533, 26769052.0, 0.0, 1609.0]
 Cluster:  0 compute-5-88        : [0.25, 26746672.0, 1.0, 4292.0]
 Cluster:  0 compute-5-90        : [0.7333, 33879124.0, 1.0, 3598.0]
 Cluster:  0 compute-5-91        : [0.335, 36252152.0, 24.0, 933.0]
 Cluster:  0 compute-5-92        : [0.3333, 27626940.0, 324.0, 84118.0]
 Cluster:  0 compute-5-93        : [0.4258, 28373680.0, 2.0, 1251.0]
 Cluster:  0 compute-5-94        : [0.5, 26247268.0, 0.0, 362.0]
 Cluster:  0 compute-5-95        : [0.3533, 22688092.0, 3.0, 5761.0]
 Cluster:  0 compute-5-96        : [0.645, 34859272.0, 0.0, 2383.0]
 Cluster:  0 compute-5-97        : [0.4208, 28628628.0, 3.0, 1342.0]
 Cluster:  0 compute-5-98        : [0.51, 25491476.0, 27.0, 5275.0]
```

Figura 4.20: Fichero resultados ejecución de algoritmo k-means

5. **Interpretación y análisis de los resultados (11)**:

   Después de obtener los resultados se deben realizar una serie de pruebas para interpretar los datos obtenidos, así como el correcto funcionamiento del algoritmo escogido. Dichas pruebas se realizan en el capítulo 5.

Capítulo 5

Evaluación Experimental

En este capítulo, se realizarán una serie de pruebas en función de los datos monitorización para comprobar el correcto funcionamiento al algoritmo *k-means*. Finalmente, se discutirán los resultados obtenidos por las pruebas.

5.1. Descripción de los experimentos

En este apartado se describen las distintas pruebas que se han realizado. Usando algoritmo k-means, como se explica en el apartado 4.3, se dividen los nodos del cluster, entre aquellos que tengan los valores de métricas, en su conjunto, más próximos. Además, se valoran, aquellas métricas que tengan más relevancia a la hora de dividir cada una de las agrupaciones. Por último, se comprueba si variando una de las métricas en uno de los nodos, repercute a su colocación en los grupos.

Situándose el usuario dentro del clúster, para realizar estas pruebas se deben ir modificando el número de métricas que se quieren introducir en ese momento. Además, a medida que se van introduciendo parámetros se modifica la cantidad de agrupaciones, que varía entre 2, 3 y 4. Variar el número de medidas, crecientemente, permite comprobar, de forma más sencilla, cuáles son aquellas que tienen mayor relevancia.

5.1.1. Monitorización y *clustering*

En este apartado se va a explicar una de las ejecuciones realizadas, paso a paso. Es decir, se van a ir analizando los resultados a medida que se incrementan el número de métricas. Para una mejor visualización y explicación de los resultados se combinan las métricas por pares. Es por ello, que se van a ir mostrando graficamente los resultados en función de las métricas escogidas y así comprobar cómo influyen en la ejecución del algoritmo. De esta forma, se observa cuál o cuáles son las medidas que tienen mayor relevancia.

En cada una de las gráficas que se muestran a continuación, se van a representar los puntos correspondientes a los nodos, con coordenadas x e y dependiendo de los parámetros que se quieran analizar en ese momento, y los *centroides* de cada agrupamiento según indique la leyenda.

Junto a cada gráfica se muestran, en las tablas, los valores de los *centroides* con el valor sin normalizar. Es decir, el valor real o, lo que es lo mismo, el dato normalizado multiplicado por el máximo valor de la métrica correspondiente a uno de los nodos de cómputo.

Como primera gráfica en cada una de las pruebas posteriores, se van a representar las métricas de uso de cpu(X) y de memoria libre (Y). A continuación, se incluyen más gráficas donde se combinan dichas métricas con el tiempo y la red, sobre todo en los casos en los que estas dos últimas cobren más peso que las dos primeras. En la mayoría de los casos, no se incluyen todas las combinaciones porque hay representaciones donde los datos no reflejan información clara.

Primero, los nodos se dividen entre 2, 3 y 4 agrupaciones para 2 únicos parámetros. Más adelante, se ejecuta el algoritmo con el fichero de entrada donde estos nodos se incrementan a 3 y 4 métricas, respectivamente. Ésto se realiza para comprobar cómo se crean los conjuntos de nodos en base al número de atributos que se introducen en el fichero de datos. Antes de observar los gráficos de cada caso, se podría decir que, al realizar graficas con dos coordenadas, aquellas ejecuciones con solo dos parámetros obtienen grupos muy diferenciados que con 3 y 4 parámetros. Cuando se añaden atributos a la ejecución, la 3ª o 4ª metrica influye en la ejecución, lo que implica que la agrupación de los nodos no se visualice de forma tan clara. Por eso es clave combinar las métricas, para crear varias

representaciones donde se coloquen como coordenadas algunos de los parámetros que puedan tener menor relevancia.

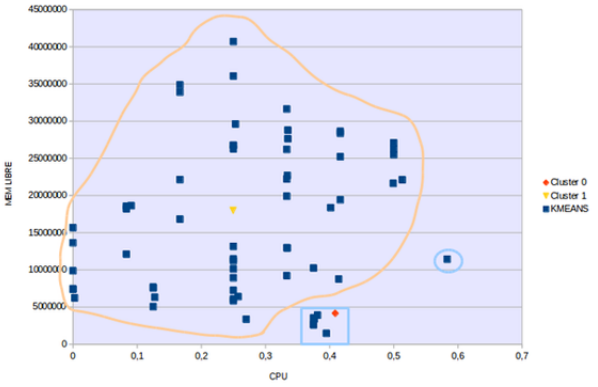
Prueba 1.1: 2 métricas

El fichero de entrada del algoritmo va a contener todos los nodos unicamente con dos métricas, eliminando los valores del resto de medidas. Para dicha realización es necesario eliminar las métricas que no se van a medir, en el fichero adicional antes de ejecutarlo y crear el fichero de entrada del algoritmo.

Se va a llevar una secuencia de pasos en cada una de las pruebas: primero, observando los valores de las tablas se ve cómo difieren cada una de las métricas entre los grupos, y a continuación, se escogen aquella o aquellas que más se difirencien entre los grupos para representarlas gráficamente, combinándolas entre sí o escogiendo el resto de parámetros de menor peso.

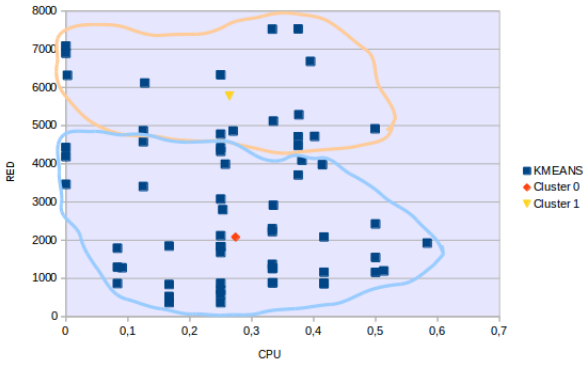
Estos son los resultados en base a ejecutar el algoritmo, únicamente con 2 grupos:

CLUSTER	CPU	MEM LIBRE
Cluster 0	0,4088933	4151256,792
Cluster 1	0,2496524	17988779,432



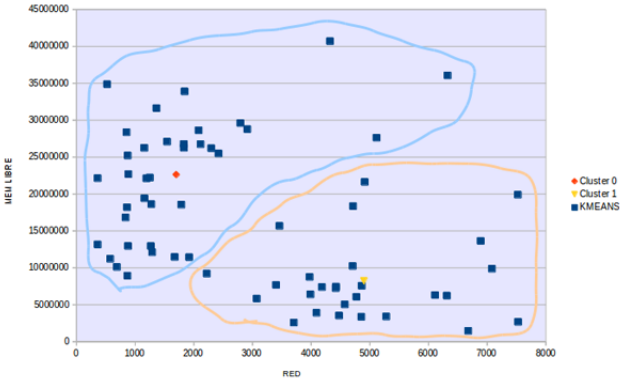
a. Prueba1 / 2 Param - Gráfica 2 clusters CPU - MEM

CLUSTER	CPU	RED
Cluster 0	0,274151	2078,004
Cluster 1	0,2642349	5774,743



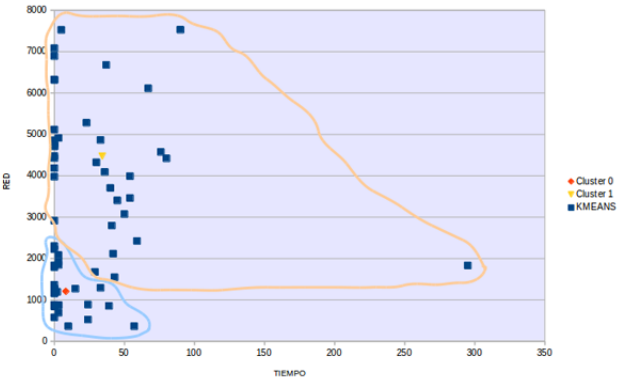
b. Prueba1 / 2 Param - Gráfica 2 clusters CPU - RED

CLUSTER	MEM	RED
Cluster 0	22628419,376	1701,554
Cluster 1	8221116,392	4901,379



c. Prueba 1 / 2 Param - Gráfica 2 clusters RED - MEM

CLUSTER	TIEMPO	RED
Cluster 0	8,26	1204,64
Cluster 1	34,22	4472,226



c. Prueba 1 / 2 Param - Gráfica 2 clusters TIEMPO - RED

Figura 5.1: Prueba 1.1: Gráfica 2 grupos - 2 Parámetros

En la gráfica **5.1.a**, correspondiente a la CPU(X) y a memoria libre(Y), junto a su tabla de valores de *centroides* se observan dos grupos bien diferenciados donde los valores de cada una de las métricas difiere bastante entre sí entre los dos grupos. El primer grupo que cuenta con, únicamente, 5 computadores corresponde a aquellos con carga de cpu alta y memoria baja, mientras que el resto está en un grupo con un promedio en cpu más bajo y memoria bastante más alta (Casi el punto medio del rango de memoria libre [0 - $\simeq$ 4 GB]). Lo que ocurre es que el número de nodos que cae en cada partición no es equitativo, además de que alguno de los valores de cpu o de memoria de los nodos de un grupo son similares a los del otro grupo. Ésto es algo normal cuando existen nodos con características

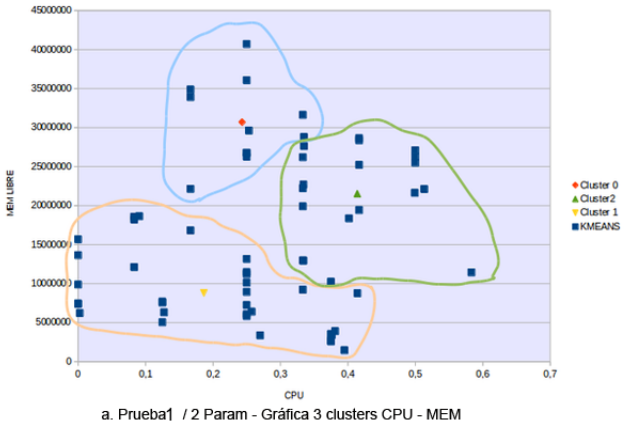
muy distintas al resto. Al observar dos grupos cuyos *centroides* están bien diferenciados, si se introduce una nueva aplicación que necesita mucha memoria libre se ejecutará en aquellos nodos del grupo 1.

En los dos siguientes casos (**Figura 5.1.b**) se sustituye la memoria por el parámetro de red, y la red por la CPU (**Figura 5.1.c**). Con estas dos últimas gráficas se comprueba cómo el parámetro de red tiene más peso que en la prueba 1. En la **figura 5.1.b** se reflejan dos grupos perfectamente diferenciados con, más o menos, similar número de nodos, mientras que la carga de CPU es similar en los dos. Es por ello, que como los nodos de grupo 1 tienen un tamaño de memoria libre muy alto, se pueden lanzar en ellos aplicaciones que requieran almacenar datos de grna tamaño. En la siguiente gráfrica tanto la memoria como la red tienen unos valores que se diferencian entre un grupo y otro. Así, en el grupo 0 se pueden ejecutar procesos que requieran del uso del procesador debido a que la carga en la mayoría de los nodos es baja. También se podría sobrecargar los nodos del grupo 1 para utilizar los del grupo 0 para otro tipo de trabajos que no requieran el procesador. Añadiendo la última métrica que no se ha comprobado, tiempo de I/O(X), combinada con el parámetro de red(Y), por ejemplo. En este último caso **(5.1.d)**, ocurre algo similar a la primera gráfica ya que, el segundo grupo cuenta con más nodos que el primero. Pero se ve claramente como es la red la que distingue, perfectamente, los nodos de un conjunto y de otro, mientras que el tiempo es similar en nodos de los dos conjuntos.

En conclusión, observando las tres últimas gráficas, en esta prueba es el valor de red el que tiene más peso en las pruebas. Además, al representar dicha medida con la memoria libre se obtiene un caso claro de que al combinar estos dos atributos los resultados obtenidos son bastante claros. También, en base al primer grafo, la carga de cpu sitúa un conjunto con nodos con carga más alta diferenciando otro grupo de nodos con menor cpu. Por tanto, aunque el atributo de red sea el que más diferencie los grupos, existen otros parámetros, memoria y cpu, que son útiles para la experimentación. Si se introdujese un nuevo nodo o se modificasen los valores medidos de uno de los nodos de la prueba, serían los tres atributos principales, sobre todo el de red, en los que el algoritmo se basaría para colocarlo. Es por ello que si se introdujese un nuevo nodo, sería el atributo de red el que tendría mayor relevancia para colocarse. Por ejemplo, si se requiere un nuevo computador para que tenga la función de servidor de datos en la red, se excogería un nodo de un grupo cuyo parámetro de red sea el más bajo, para no sobrecargar su tráfico de red.
Por el contrario, si el algoritmo no se está ejecutando con el atributo de red, habría que combinar el valor de cpu y de memoria para particionar los nodos.

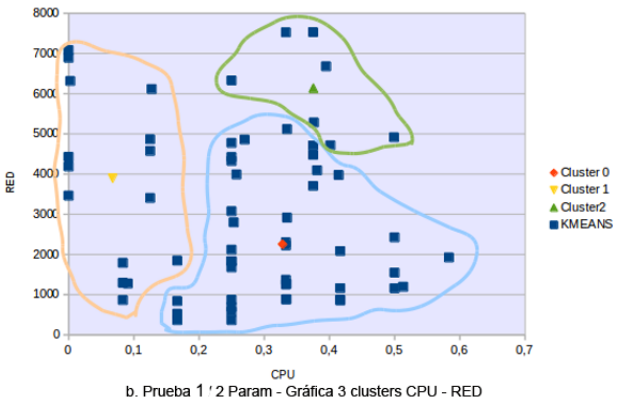
En la siguiente prueba se comprueba si con 3 grupos sigue ocurriendo lo mismo. Así, igual que en el caso anterior se ejecuta el algoritmo con los parámetros de cpu y memoria, y cpu y red. Se excluye el valor del tiempo porque es irrelevante tanto en el caso anterior como en éste.

CLUSTER	CPU	MEM LIBRE
Cluster 0	0,2432361	30686741,384
Cluster 1	0,186656	8790896,736
Cluster 2	0,414143	21529557,284



a. Prueba1 / 2 Param - Gráfica 3 clusters CPU - MEM

CLUSTER	CPU	RED
Cluster 0	0,3283979	2251,171
Cluster 1	0,0676628	3892,493
Cluster 2	0,3756452	6136,135



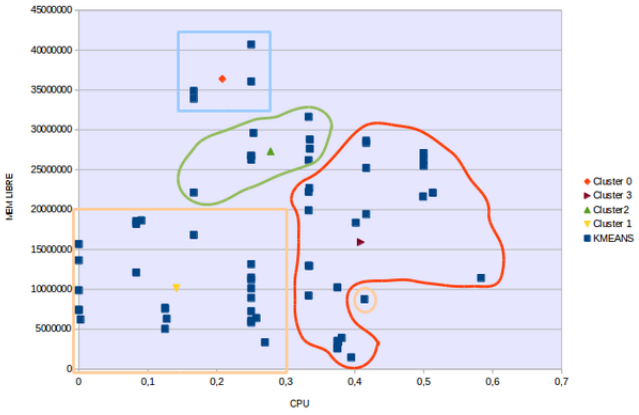
b. Prueba 1 ' 2 Param - Gráfica 3 clusters CPU - RED

Figura 5.2: Prueba 1.1: Gráfica 3 grupos - 2 Parámetros

En la **Figura 5.2.a**, cpu(X) y memoria(Y), se distinguen tres grupos: el primero con una memoria libre bastante alta mientras que la cpu es media, el siguiente grupo tiene una memoria más baja al igual que su cpu, y el último dónde la cpu es alta mientras que la memoria libre tiene un valor medio. Por tanto, la selección hecha por el grupo viene dada no solo por uno de los parámetros sino por los dos conjuntamente, debido a que si solo se escogería uno de ellos habría nodos con mismo valor de parámetro en un conjunto u otro. Es por ello, que si se requiere un nodo para ejecutar una aplicación que requiera de mucha memoria libre se elegiría los nodos del grupo 0. Además, el grupo 0 es más ideal para ejecutar procesos que necesiten tener el control del procesador.
Seguidamente, se representa la red (Y) junto a la carga de cpu (X), por ejemplo. En la **Figura 5.2.b** al igual que en el caso anterior son los dos parámetros los que provocan unos resultados claros. Es por ello, que los nodos del grupo 1 serían ideales para lanzar procesos que requieran el uso del procesador. En la última partición no sería recomendable tener un nodo que actúe como servidor de red ya que son nodos con mayor tráfico en red.
Entonces, se concluye que, los tres parámetros cobran gran importancia cuando se tienen que dividir en tres subconjuntos. Además, hay que combinarlos entre ellos para que los resultados sean los que se quieren esperar. En base a esta conclusión, si uno de los nodos, en una siguiente ejecución cambiase dos de estos tres parámetros, sería muy probable que se moviese a otro grupo. Por ejemplo, si en la figura 5.2.b se introduce una aplicación que aumente la carga de un nodo del grupo 1 para que tenga un valor similar a la carga de los otros dos agrupamientos, habría que fijarse en su atributo de red para colocar el punto en uno de los dos grupos.

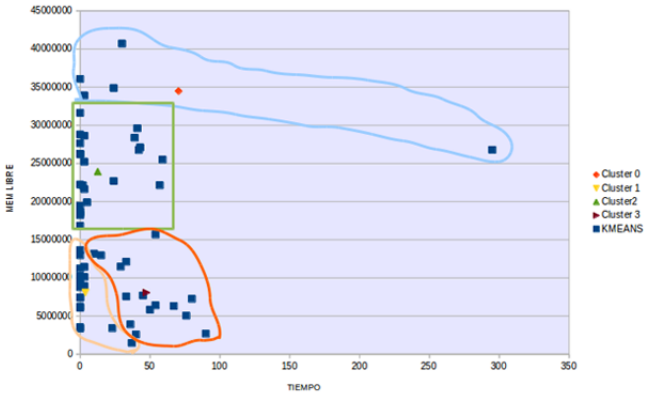
En el siguiente caso, se divide el conjunto de nodos en 4 grupos. Se comienza como anteriormente, con las métricas de cpu y memoria.

CLUSTER	CPU	MEM LIBRE
Cluster 0	0,2082381	36384544,824
Cluster 1	0,1417419	10133950,404
Cluster 2	0,2782341	27308757,916
Cluster 3	0,4088933	15913151,036



a. Prueba 1 / 2 Param - Gráfica 4 clusters CPU - MEM

CLUSTER	MEM LIBRE	TIEMPO
Cluster 0	34471710,812	70,505
Cluster 1	8058322,008	3,835
Cluster 2	23930774,448	12,685
Cluster 3	8058322,008	47,495



b. Prueba 1 / 2 Param - Gráfica 4 clusters TIEMPO - MEM

Figura 5.3: Prueba 1.1: Gráfica 4 grupos - 2 Parámetros

En las pruebas con 3 subconjuntos con estas mismas coordenadas cpu(X)- memoria libre (Y), (**Figura 5.3.a**) había una situación ideal en la que se dividían los grupos sin ningún nodo que, a simple vista, debiera pertenecer a otro grupo. En esta figura, el **nodo 37** con una cpu=0.4141 y memoria_libre=8756160 que pertenece al grupo 1, parece raro que no se incluya en el 3. Ésto es debido a que aunque su cpu es similar a la de éste último grupo, su memoria se asemeja más a la mayoría de nodos del grupo 1 y no a la mayoría del grupo 3.
De este último caso se puede concluir que con dos parámetros es más fácil distinguir grupos bastante diferenciados ya sea por una, otra o ambas variables. Más adelante se comprobará que a medida que se añaden parámetros esos grupos no se diferencian tan claramente y puede ocurrir en gran medida, con más nodos, lo que acaba de suceder con ese único nodo. Posteriormente se detallará lo que ocurre y por qué. Además, el tener los grupos 0 y 2 donde la memoria libre es mayor, se pueden lanzar aplicaciones en nodos de este grupo que requieran mayor memoria libre para almacenar sus datos mientras se esté ejecutando.
Ahora bien, finalizando con la ejecución del algoritmo con dos parámetros en estos 4 grupos se representan los nodos con los parámetros de tiempo(X)-memoria libre(Y), por ejemplo, y así se comprueba si el tiempo se tiene en cuenta

en alguna de las medidas.
Como se ve en la **figura 5.3.b**, la memoria tiene más peso para dividir el conjunto. Se podría discutir la división entre los grupos 2 y 3 que se realiza en base al tiempo. Pero en ello también hay que tener en cuenta que los dos podrían unirse en 1 si se hablaría de unicamente 3 clusters. Este es uno de los casos en los que, tal vez, realizar el agrupamiento con tantos clusters provoca que haya una división en dos que no debería de existir.
En el caso de cpu y memoria de la figura 5.3, si se introdujese un nuevo proceso que necesitase el procesador para ejecutarse, seguramente escogería algún nodo del grupo 1 para ejecutarse donde la carga es más baja. Si escogería cualquier otro grupo, sobre todo el 3, además de que podría regargar el equipo, tal vez esté ejecutando otros procesos y lo ponga en la cola. También puede ser interesante, utilizar nodos como los del grupo 3 que estén muy cargados para ejecutar los procesos nuevos que se introduzcan y así tener otros grupos que realicen otro tipo de trabajos, o en último caso, si no están sobrecargados, ahorrar energía apagándolos.

En todas los casos realizados con 2 parámetros, la división de los grupos es bastante clara debido a que en los grafos sólo se tienen en cuenta las medidas que se ejecutan. Las métricas de mayor peso han sido, sobre todo, la carga de cpu, la memoria libre y la red. Es interesante comprobar cómo a medida que se incrementa el número de agrupaciones, se necesita combinar dos de los atributos para ver las agrupaciones bien diferenciadas, en cambio, con dos únicos grupos es el parámetro de red el que más influye. Por eso, si un nuevo nodo es monitorizado, si se dividiese el conjunto en dos particiones, influiría, en mayor medida e parámetro de red para colocarlo, mientras que si se particiona en 3 o 4, se necesitarían combinar dos de los tres atributos para situarlo dentro de un grupo. En la siguiente prueba 2.2, se observa cómo influye ese tercer parámetro.

Prueba 1.2: 3 métricas

Esta sección, se realiza de la misma manera que la anterior, pero se varía la ejecución del algoritmo pasándole, a cada nodo del fichero, 3 parámetros y no dos. El problema que va a surgir en las siguientes representaciones no se ha visto hasta ahora, o no en gran medida. Y es que haya nodos, que por sus dos únicos valores representados, se visualicen dentro de otro grupo al que no pertenecen. Esos equipos tienen similares valores en cuanto a las dos medidas que se representen pero el tercer parámetro provoca que se coloque en un grupo que, tiene un valor conjunto más equivalente al de este nodo. Por ello, es clave mostrar nuevos grafos con esa tercera medida como coordenada y comprobar si es eso lo que ocurre.

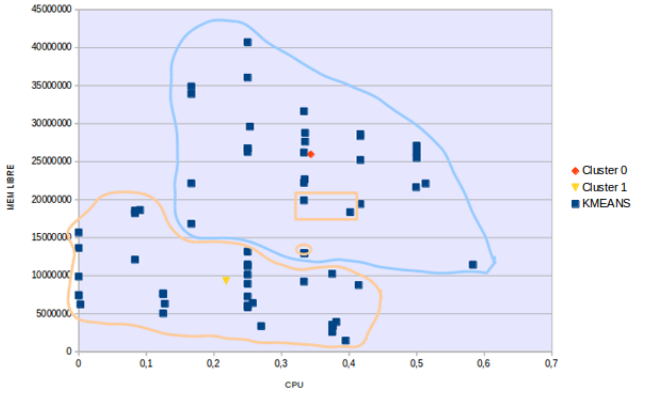
Se comienza por la ejecución de k-means con respecto a CPU-MEM-RED ya que en el apartado anterior se vio que eran los tres más importantes. Aún así, posteriormente, se sustituye la métrica de RED por la del tiempo de entrada y salida de disco.

Lo que ocurre en la **Figura 5.4.a** es lo que se suponía. Aunque, en general, hay dos subconjuntos más grandes, diferenciados sobre todo por el atributo de memoria, hay nodos dentro de ellos que, seguramente, debido a sus valores de red, pertenecen al otro conjunto cuyo centroide cpu-mem está más alejado. Para aclararlo, se muestra la gráfica de red (X) - Memoria libre (Y), seguidamente. En esta primera figura eciste un grupo 0 donde todos sus nodos tienen mayor memoria libre, por lo que es en ellos donde se lanzarían aplicaciones que necesiten mayor espacio para ejecutarse.

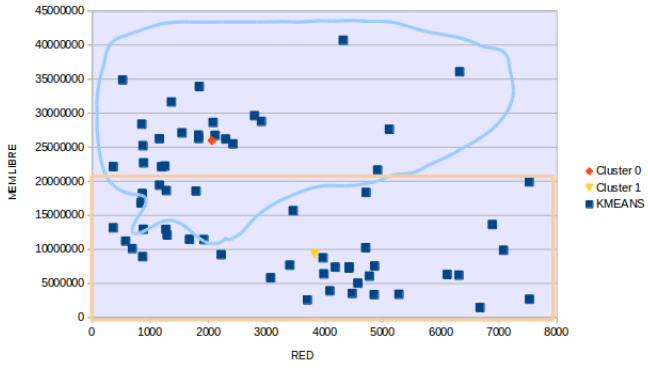
Aunque en la **Figura 5.4.b** no se percibe qué nodos son los que antes creaban confusión, se ha comprobado que son aquellos que están claramente definidos en su grupo puesto que están más cercanos en red a la mayoría de nodos de su conjunto. En consecuencia de la visualización de los grupos, al igual que en la figura anterior, los nodos del grupo 0 son más ideales para lanzar aplicaciones que necesiten mayor espacio de ejecución.

Variando la ejecución y sustituyendo el parámetro de red por el del tiempo (ya que en la primera gráfica cpu-mem los grupos están bastante diferenciados) se comprueba si la métrica de tiempo tiene la misma repercusión que la red. En la **Figura 5.4.c**, cpu(X)-memoria libre(Y), ejecutando el algoritmo no cambia la disposición de los subconjuntos, ya que es la memoria la que mantiene la distinción entre ambas agrupaciones, semejante a la división de la figura 5.4.a. Así se ve comparando las dos tablas, donde solo varía el parámetro de red-tiempo, que los valores de cpu y memoria, entre ellas, son bastante similares al igual que la división de grupos obtenida. Es por ello, que al lanzar una nueva aplicación serían los nodos del grupo 0 los más propensos a ejecutarla por su mayor memoria libre.
Si se ejecuta una aplicación que necesitara un gran espacio de memoria libre se escogería aquel grupo de nodos con mayor valor de carga libre. En el caso de la figura 5.4.a, por ejemplo, el grupo 0.

CLUSTER	CPU	MEM LIBRE	RED
Cluster 0	0,3435637	25965704,248	2062,946
Cluster 1	0,2181542	9319978,484	3832,261

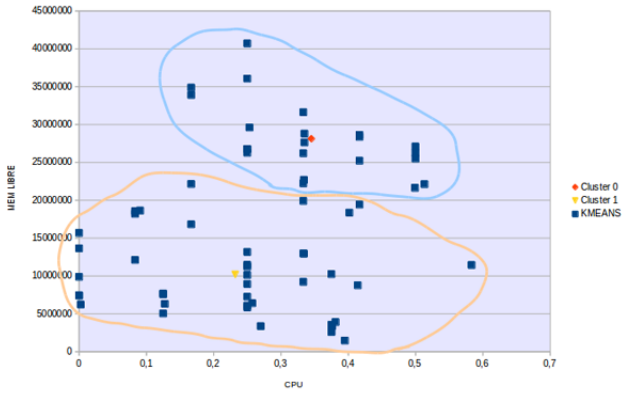


a. Prueba 1 / 3 Param - Gráfica 2 clusters CPU - MEM /RED



b. Prueba 1 / 3 Param - Gráfica 2 clusters RED - MEM / RED

CLUSTER	CPU	MEM LIBRE	TIEMPO
Cluster 0	0,3453136	28122729,836	27,73
Cluster 1	0,2321534	10174649	20,65



c. Prueba 1 / 3 Param - Gráfica 2 clusters CPU - MEM / TIEMPO

Figura 5.4: Prueba 1: Gráfica 2 grupos - 3 Parámetros

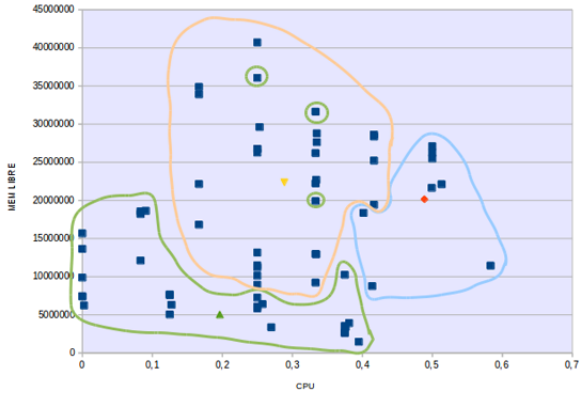
En este siguiente apartado se define un nuevo grupo. De la misma manera, se comienza ejecutando con cpu-mem-red, que suelen ser los de mayo peso, y se finaliza sustituyendo red por tiempo.

De la misma forma que ocurría anteriormente, en la **figura 5.5.a** hay nodos que se visualizan en el grupo 1 pero pertenecen al 2. Aun así, los tres grupos en general están bastante bien diferenciados. El primero por una alta CPU y una memoria libre media, el segundo por tener una cpu y memoria libre media y el tercero por tener una memoria baja. Además, en la tabla, se comprueba que la red en la tercera agrupación es bastante superior a los otros dos grupos. Lo que ocurre en la gráfica anterior con los nodos que pertenecían al 2 en vez de al 1, se ve bien identificado en la **figura 5.5.b** con la diferencia de nodos gracias a la medida de red. Es por ello que procesos que requieren el uso del procesador se intentarán ejecutar en nodos del grupo 2 o 3, o si un computador funciona como servidor que realiza transacciones web con los clientes de una empresa pertenece al grupo 2 de la figura 5.5.b.

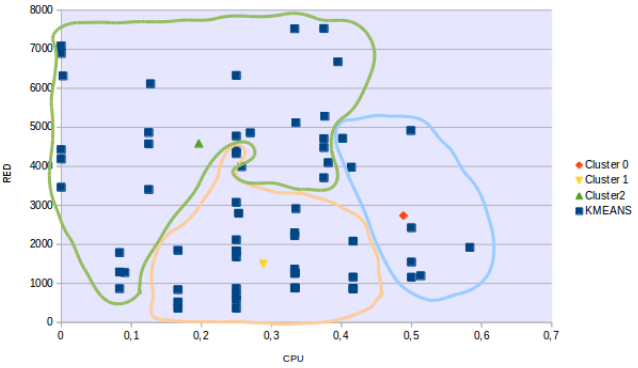
Se finaliza comprobando si el tiempo es relevante, o bien ocurre lo mismo que anteriormente. En la **Figura 5.5.c** se refleja cómo el parámetro de tiempo afecta a uno de los nodos que se visualiza dentro del grupo 0 pero pertenece al 1. Este es el nodo con mayor tiempo de todo el conjunto. En la siguiente **figura 5.5.d** se muestra como ese nodo tiene mayor cercanía en tiempo a los nodos de su grupo.

Igual que ocurría con la ejecución con dos parámetros, se necesitaría que un nodo modificase dos de los tres parámetros de mayor peso para cambiar de grupo.

CLUSTER	CPU	MEM LIBRE	RED
Cluster 0	0,4888054	20145805,02	2733,027
Cluster 1	0,2887335	22343529,204	1490,742
Cluster 2	0,1965721	5036451,255	4585,161

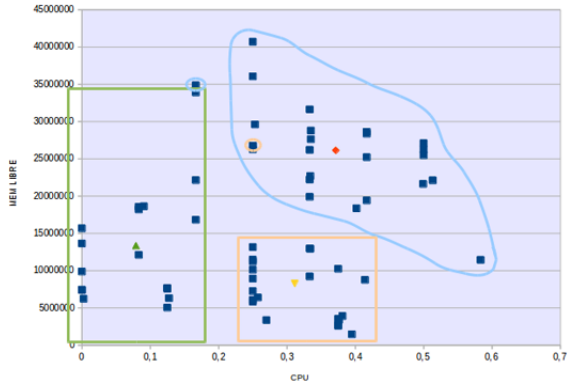


a. Prueba 1 / 3 Param - Gráfica 3 clusters CPU - MEM /RED

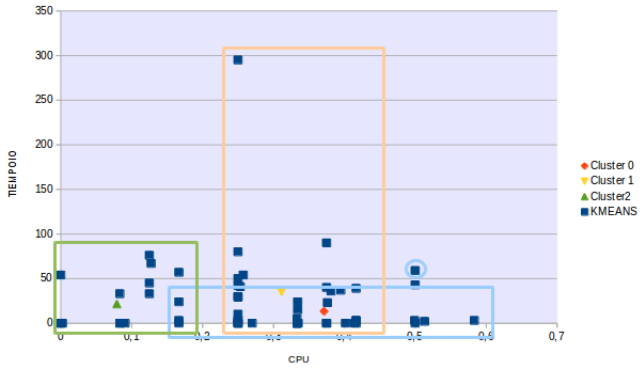


b. Prueba 1 / 3 Param - Gráfica 3 clusters CPU - RED / RED

CLUSTER	CPU	MEM LIBRE	TIEMPO
Cluster 0	0,3715621	26128498,632	13,275
Cluster 1	0,3114822	8261814,988	34,81
Cluster 2	0,0793288	13349139,488	21,535



c. Prueba 1 / 3 Param - Gráfica 3 clusters CPU - MEM / TIEMPO



d. Prueba 1 / 3 Param - Gráfica 3 clusters CPU - TIEMPO / TIEMPO

Figura 5.5: Prueba 1.2: Gráfica 3 grupos - 3 Parámetros

En el último caso de este apartado, se define un nuevo grupo en la ejecución y se sigue la misma secuencia anterior. Se empieza ejecutando el algoritmo con el fichero compuesto por solo las métricas CPU-MEM-RED. El primer gráfico (**Figura 5.6.a**) correspondiente a CPU-MEM queda de la siguiente forma:

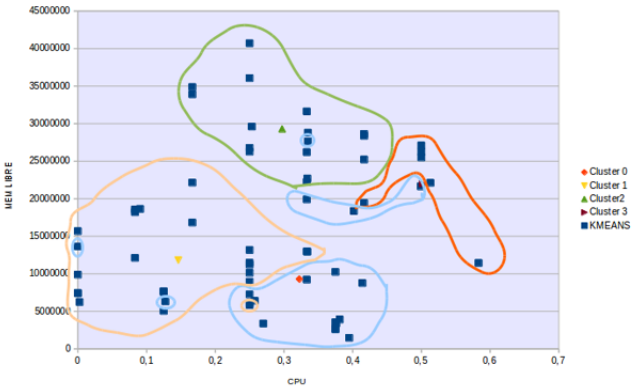
En la primera gráfica se ven 4 particiones bien diferenciadas salvo algunos nodos que pueden crear confusión al visualizarse en grupos a los que no pertenecen. Se va a evitar reflejar gráficas con una de las coordenadas de red que lo explique porque puede resultar, aún, más confuso. La explicación es la misma que en ocasiones anteriores. Por ejemplo, si una aplicación requiere ser ejecutada en algún equipo, se escogería uno de los nodos del grupo 2, donde existe mayor memoria libre.

Así mismo, se comprueba lo que ocurre con el tiempo, en la **figura 5.6.b**. En esta gráfica se visualiza lo mismo que con la red, salvo que las 4 agrupaciones están mucho mejor diferenciadas. Ésto quiere decir que la red tiene mayor repercusión a la hora de dividir los grupos. En consecuencia de ello, si un proceso nuevo se introduce es necesario un nodo no sobrecargado del grupo 1 para ejecutarlo. Así mismo, aplicaciones que requieren memoria libre para ir guardando sus datos se ejecutarían en nodos de la partición 0 que cuentan con mayor memoria libre.

Al ejecutar k-means con tres parámetros y representar gráficamente solo 2 de ellos se ve cuánta es la influencia del tercer parámetro no representado. En los grafos donde solo se muestran CPU y memoria libre se dividen las particiones de manera clara salvo algún nodo que está condicionado por el nuevo parámetro introducido. Cuando se representa el nuevo parámetro como coordenada se observa cómo esos nodos están mas cerca de la mayoría de nodos del grupo al que pertenece. Al sustituir el atributo de red por el tiempo, hay casos en las que solo la cpu y la memoria son relevantes, y otras en la que la relevancia del tiempo es mínima. Por tanto, aunque en la mayoría de los equipos son la cpu y la memoria las que cobren el mayor peso, el parámetro de red también influye, notablemente, en

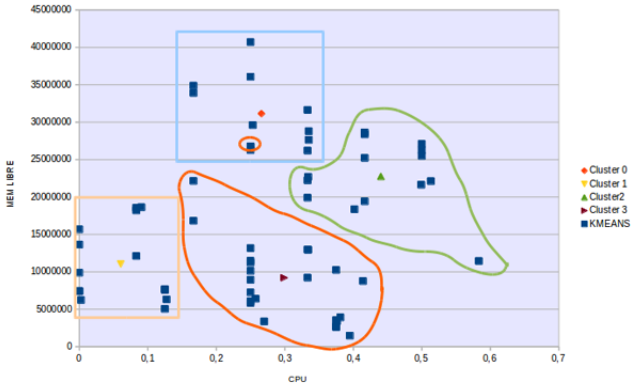
alguno de los nodos. Es por eso, que si se modifica algún valor medido de un nodo, si es solo el tiempo, seguramente el nodo se se movería de grupo. En cambio, si se aumenta o disminuye la medida de cpu o memoria, la mayoría de nodos se colocaría en otro grupo. Solo algunos equipos se basarían en la red para agruparse.

CLUSTER	CPU	MEM LIBRE	RED
Cluster 0	0,3225649	9279279,888	5051,959
Cluster 1	0,1464083	11802592,84	2461,983
Cluster 2	0,297483	6681081,51936	2130,707
Cluster 3	0,5022213	21977241,84	1566,032



a. Prueba 1 / 3 Param - Gráfica 4 clusters CPU - MEM / RED

CLUSTER	CPU	MEM LIBRE	TIEMPO
Cluster 0	0,2659848	31134425,94	12,685
Cluster 1	0,0606632	11029319,516	22,125
Cluster 2	0,4403915	22750515,164	12,98
Cluster 3	0,2992329	9197882,696	34,22



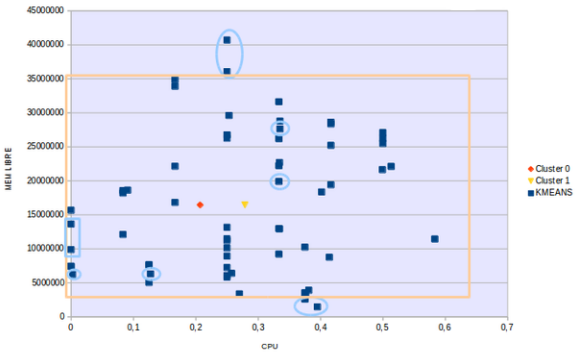
b. Prueba 1 ' 3 Param - Gráfica 4 clusters CPU - MEM / TIEMPO

Figura 5.6: Prueba 1.2: Gráfica 4 grupos - 3 Parámetros

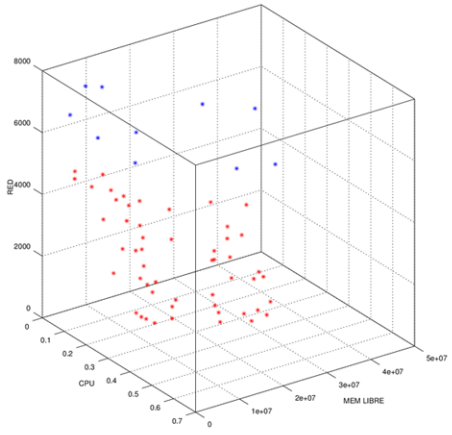
Prueba 1.3: 4 métricas

En este apartado se va a ejecutar el algoritmo en dos instantes de tiempo distintos pasando los 4 parámetros en cada nodo del fichero de entrada. Así, se comprueba cómo varía la relevancia de las métricas cuando transcurre un período de tiempo entre cada prueba. Con este apartado se ve cómo afectan las 4 métricas a la hora de seleccionar los grupos, sobre todo cómo afectan la red y el tiempo a los parámetros de cpu y memoria que suelen ser de mayor peso. Se comienza dicha prueba de la misma manera que las anteriores situando en un gráfico CPU(X) - MEM(Y) para tener en cuenta como influyen los otros dos o solo uno de ellos.

CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,2070715	16442232.781	23.01	6392.121
Cluster 1	0,2788174	16442232.784	23.01	2446.925



a. Prueba 1 / 4 Param - Gráfica 2 clusters CPU - MEM



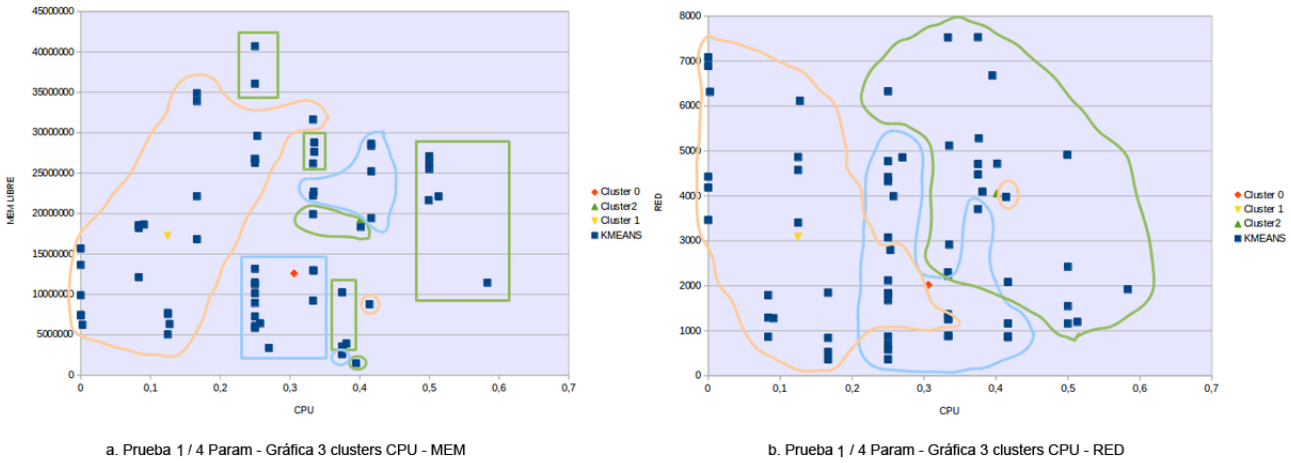
b. Prueba 1/ 4 Param - Gráfica 2 clusters CPU - MEM - RED / 3D

Figura 5.7: Prueba 1.3: Gráfica 2 grupos - 4 Parámetros

Como era de esperar, en el grafo **5.7.a** ya no existen dos particiones perfectamente ubicadas, sino un segundo grupo que cuenta con la mayor parte de nodos y otro grupo en el que los valores de cpu-mem no son tan similares. Si se observa la tabla, la medida de memoria libre y tiempo es paralela entre los dos mientras que los valores de cpu tampoco se distinguen enormemente. El único del que habría que hacer mención es el parámetro de red. Es por eso que es necesario representar dicha métrica de forma más explícita. Para ello, la **Figura 5.7.b** en 3D, se ha realizado para reflejar las 3 variables más destacadas (cpu - mem - red) en un mismo gráfico. Se ven claramente dos grupos diferenciados. El 0, que antes contaba con nodos más dispares en cuanto a cpu y memoria, ahora se ve como es la red la que los diferencia tal como aparecía en la tabla, dividiendo los equipos con más uso de red del resto.
Por tanto, si se requiere un nodo como servidor de aplicaciones web que constantemente ejecutan transacciones con los usuarios es necesario escoger nodos del grupo 1 cuyo tráfico de red es más bajo.

En el siguiente caso, se aumenta el número de agrupaciones y se comprueba si afecta algo a lo reflejado, anteriormente, con dos.

CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,3062325	12575866.164	18,029	2017.772
Cluster 1	0,1248262	17174807.512	32.155	3086.89
Cluster 2	0,4013104	19291134.504	16.815	4058.131



a. Prueba 1 / 4 Param - Gráfica 3 clusters CPU - MEM

b. Prueba 1 / 4 Param - Gráfica 3 clusters CPU - RED

Figura 5.8: Prueba 1.3: Gráfica 3 grupos - 4 Parámetros

En la **Figura 5.8.a** ocurre algo parecido a lo anterior pero son las agrupaciones 0 y 2 las que cuentan con nodos más dispersos en la gráfica de cpu(X) y memoria libre (Y). Tampoco la tabla es de gran ayuda para entender qué parámetro tiene más infuencia, puesto que hace bastante distinción entre cada una de las métricas de los grupos, sobre todo en el caso del 2 (en verde) con una CPU, MEM y RED altas. Para ello está bien, reflejar alguna de las demás gráficas sustituyendo alguno de los parámetros. Aun así, queda claro que si un proceso requiere un equipo para ejecutarlo, nodos del grupo 1 serían los más ideales. Si se quiere ahorra energía apagando algún equipo que no se está utilizando, se ejecutará en algún nodo del grupo 2 para sobrecargarlos al 100 %.
Se ha escogido CPU-RED donde se puede comprobar, más o menos, por qué los grupos están tan dispersos. En la **Figura 5.8.b** se observa cómo aún hay nodos que pueden estar superpuestos a otros nodos de otros grupos ya sea por CPU o por RED. Pero muchos de los nodos que antes se encontraban dispersos se concentran más cerca de la mayoría de puntos de su grupo. Por ejemplo, los nodos del grupo 2 (verde). En vista de la figura, los nodos del grupo 1, son mejores para ejecutar procesos que requieran el uso del procesador, tal como se dijo en la anterior figura.

Si observando cualquiera de las dos gráficas existe algún nodo que sigue creando confusión porque se visualiza en algun grupo que no pertenece seguramente sea cuestión del atributo de red. Pero como los casos son mínimos se intenta evitar colocar otra siguiente gráfica que pueda crear una mayor confusión.

El último caso de esta prueba va a considerarse con 4 agrupaciones y 4 parámetros, siendo la más cercana a las prácticas que se puedan realizar en un futuro en clústers, monitorizando más equipos.

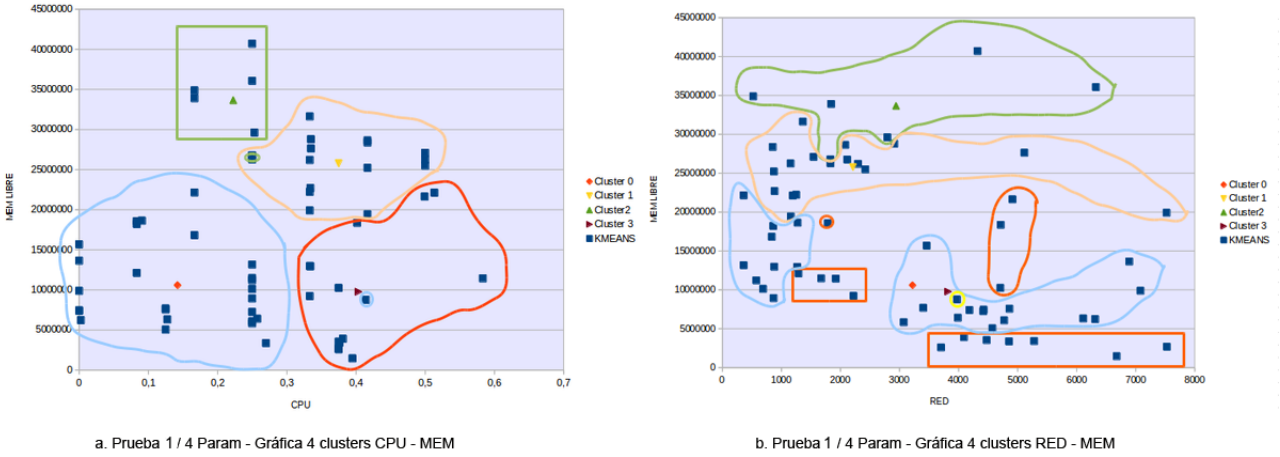
En la **figura 5.9.a** se muestran 4 particiones muy bien diferenciadas lo que hace seguir creyendo que cpu y memoria son los parámetros más relevantes. Aún así, hay dos nodos que se visualizan dentro de los agrupamientos 1 y 4 pero que pertenecen a grupos distintos. Seguidamente, se van a analizar detalladamente, aunque nunca antes se hubiese realizado. Así, se comprueba perfectamente cómo cualquiera de las métricas que no están representadas puede variar la posición de esos nodos.

Además se podría recordar la explicación de cómo funciona el algoritmo. Kmeans busca agrupar nodos que, en función de los valores de las cuatro métricas, al hacer la media entre los cuatro número, pertenezca a grupos cuyo centroide sea cercano al valor de esa media. Es decir, si el valor de un atributo, en un equipo, pueda parecer que deba pertenecer a un grupo por su similitud, existe otro parámetro que aumenta o decrece el promedio del nodo para que se una a otro agrupamiento con centroide cercano a su valor promedio. Para ejemplificarlo, eso es lo que ocurre con el computador 87 cuya memoria es similar a la del grupo 1, pero el valor del tiempo es más similar al del 2. Con el nodo 37 existe más confusión puesto que sus valores, en conjunto son más semejantes a la agrupación 3. Por tanto, se va a mostrar, el nodo en una gráfica de red(x) memoria libre(y). Aunque la **figura 5.9.b** parece que puede crear más confusión solo hay que fijarse en el nodo 37 rodeado en amarillo. Ahí se ve como en este caso debido a razones que combinan los valores de memoria y de red está más cercano a los nodos del grupo 0 y no al 3.

Como conclusión a esta última prueba ejecutada con los 4 parámetros siguen siendo cpu, memoria libre y red los que tienen más relevancia. En muy pocos casos se ha visto influencia del tiempo de entrada y salida. Si un nuevo nodo se monitorizase, el algoritmo no tendría en cuenta el tiempo para colocarlo dentro de un grupo. Viauslaizando la figura 5.9.a, lanzar un nuevo proceso que necesita ejecutarse rapidamente, necesitaría un nodo del grupo 0 o 2 cuya carga de cpu es más baja. Si se ejecuta una aplicación que necesita espacio libre para guardar sus datos sería ideal que fuese un nodo del grupo 2, o en su defecto, del 1 el que la ejecutara.

CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,1423252	110581634.96	22.125	3222.412
Cluster 1	0,3750619	25762211.268	13.57	2213.526
Cluster 2	0,42228206	33657738.892	65.49	2943.839
Cluster 3	0,4036436	9767663.04	17.7	3824.732

NODO	CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Nodo 87	2	0,25	26768056	295	1831
Nodo 37	0	0,0.4142	8756160	10	3975



a. Prueba 1 / 4 Param - Gráfica 4 clusters CPU - MEM

b. Prueba 1 / 4 Param - Gráfica 4 clusters RED - MEM

Figura 5.9: Prueba 1.3: Gráfica 4 grupos - 4 Parámetros

Prueba 1.4 - 4 parámetros (2º ejecución)

La segunda prueba con los 4 parámetros, explicada a continuación, se realiza, sobre todo, para comprobar cómo varía la influencia de las medidas en el algoritmo y qué métrica o métricas se eligen como más relevantes a la hora de agrupar los puntos. Esto último corresponde a aquellos valores que provocan que la división entre los clusters sea más diferenciada, o dicho de otra manera, agrupar aquellos nodos con similar valor de dicho parámetro o parámetros en un mismo grupo. Como en los anteriores apartados, se comienza ejecutando con únicamente dos agrupaciones, que se irán incrementando a medida que se avance.

CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,3541839	25449010,704	16,515	5383,552
Cluster 1	0,2507886	8047977,744	23,121	4121,782

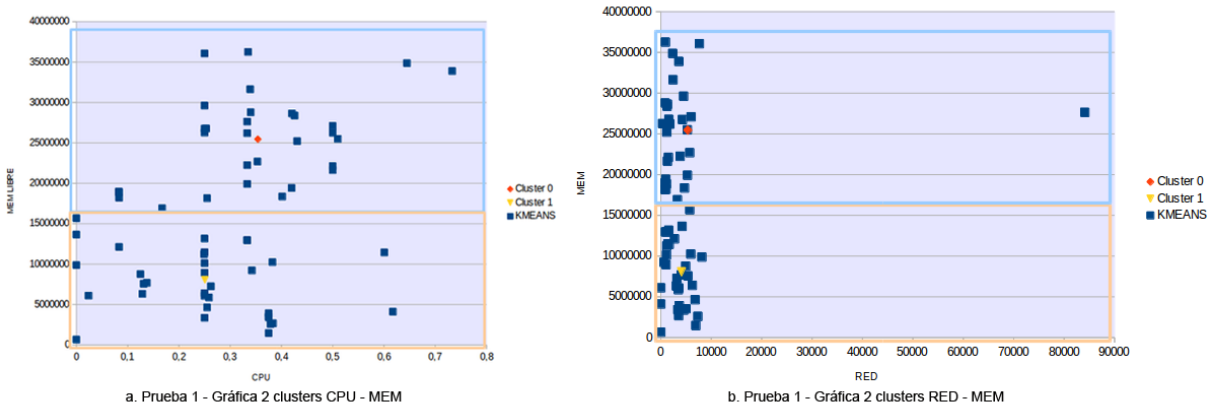


Figura 5.10: Prueba 1.4 - Gráfica 2 grupos

Observando los valores de la tabla se ve cómo es la memoria libre la que más difiere entre sus valores. Entonces, se combina esa medida con cualquiera de las otras tres métricas, como CPU y RED, para formar las dos primeras gráficas. En la primera, **Figura 5.10.a**, correspondiente a CPU(X)-MEMORIA LIBRE(Y) se observan dos grupos perfectamente diferenciados. En este caso, es el valor de la memoria el que distingue el agrupamiento 0, con un promedio de memoria libre en torno a los 25 GB, del grupo 1, que tiene una media de casi 8 GB. Por tanto, aquellos nodos cuyo parámetro de memoria se aproxime a uno de los dos valores del centroide, se situa en un grupo u otro. Dicho de otra manera, los nodos con una memoria libre superior a 1.5 GB se colocan en el 0 mientras que los que cuenten con una memoria inferior se concentran en el 1. Se observa, perfectamente, como el valor de CPU no toma gran importancia ya que en cada uno de los clusters existen puntos con similar carga de cpu.
En el segundo gráfico **5.10.b**, se sustituye el valor de X, CPU, por la RED dejando en la componente Y el valor de memoria libre. Con esta gráfica, se vuelve a comprobar cómo es la memoria la que vuelve a ser más importante a la hora de dividir los grupos, mientras que con el valor de red ocurre lo mismo que con la CPU. Por eso, el nodo cuya red es superior al resto se coloca en el grupo cuyo valor de memoria es similar a él. En estas dos figuras, nodos del grupo 0 serán más ideales para ejecutar aplicaciones que requieran más espacio libre en RAM para ser ejecutadas.

Se van a poner otros dos ejemplos con 3 y 4 grupos dónde no solo uno de los parámetros cobre importancia a la hora de dividir los nodos, sino que sean dos, conjuntamente, los que tengan mayor peso a la hora de decidir qué nodos se colocan en cada grupo.

Al comprobar, primero, los valores de la tabla existen grandes diferencias, en función de los cuatro parámetros. Por tanto, obserbando la primera gráfica **5.11.a** representadas CPU(X) y MEM (Y) se dividen claramente 3 grupos. El primero con cpu y memoria media, el segundo con cpu media y memoria baja, y el último con cpu más baja que los dos anteriores y memoria libre más alta que el 0 y 1. Ninguno de los dos atributos es clave para la partición, sino que hay que combinar los dos para dividir de manera clara el conjunto. En consecuencia de la observación de la gráfica, aplicaciones pequeñas se ejecutarán en algúno de los nodos del grupo 1, mientras que aquellas que necesitan cargar una gran cantidad de librerías, como puedan ser *Matlab* o *Photoshop*, requieren de nodos con mayor espacio

libre en RAM, como son los nodos del grupo 0 y 2.

CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,4157811	20373709,424	21,286	6392,968
Cluster 1	0,2522552	6090361,536	31,929	4794,726
Cluster 2	0,2192567	24433950,448	2,936	2607,658

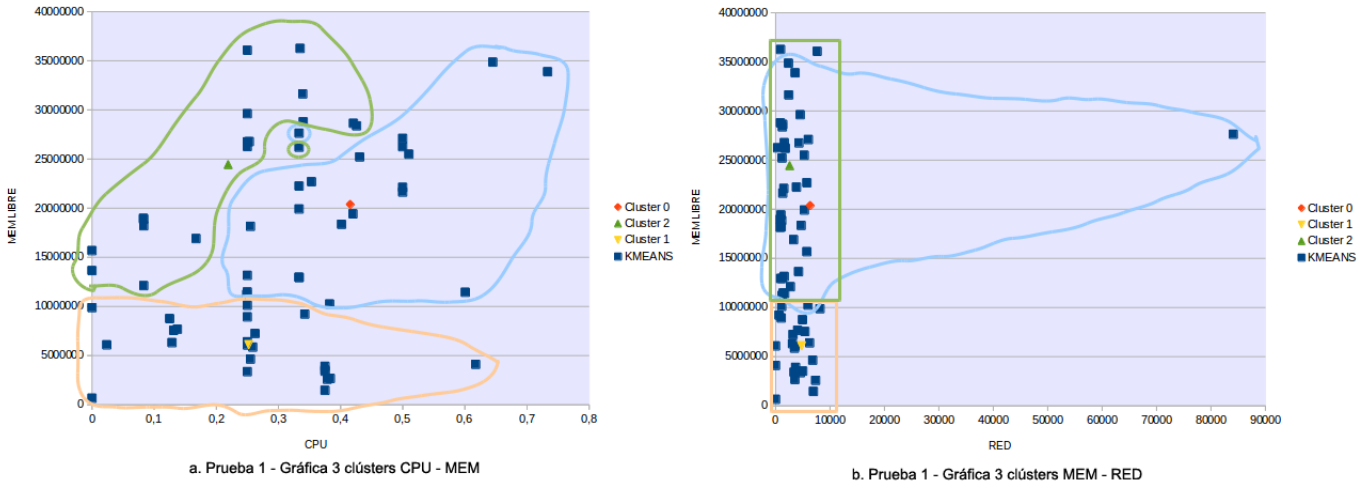


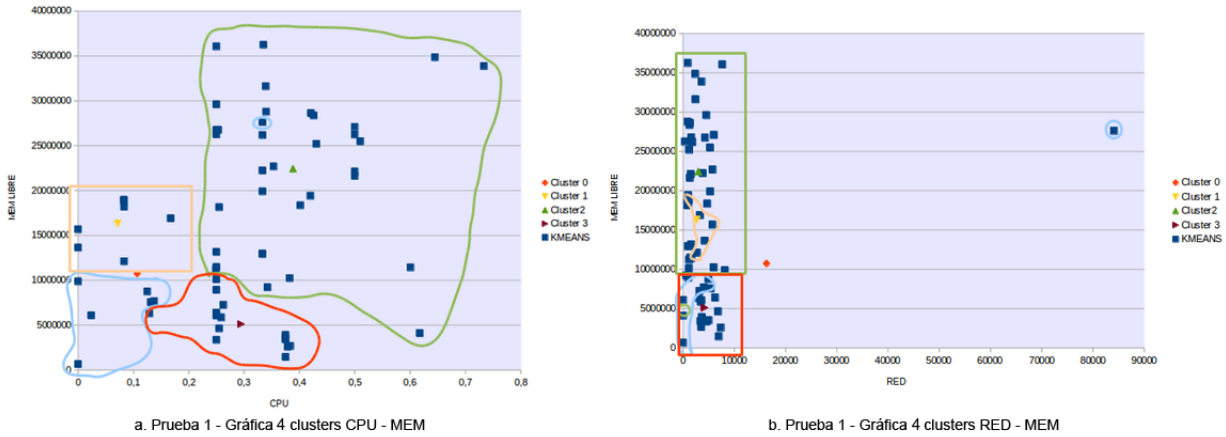
Figura 5.11: Prueba 1.4 - Gráfica 3 grupos

Añadiendo una nueva gráfica,**5.11.b** donde se sustituye el parámetro de cpu por el de red (X) y se mantiene la memoria (Y), el uso de red no influye en la partición ya que hay grupos de nodos que se superponen, como es el caso del 0 y el 2, sobre todo.

Como se observa en los valores de la tabla siguiente son las 4 medidas las que difieren entre sí en los 4 grupos. Por tanto, no se tienen preferencias y se escogen los mismos parámetros que en la prueba con 2 grupos, combinándolos entre sí. En la primera de las gráficas (**Figura 5.12.a**), se representa la CPU (X) y la memoria libre (Y). Existen 4 grupos perfectamente diferenciados. El 0 con cpu y memoria libres bajos, el 1 con memoria libre media y cpu baja, el 2 con cpu media-alta mientras que la memoria libre media tiene un valor alto y por último, el 3, cuya cpu es media junto a una memoria baja. En vista del grafo, si se intantara dividir los grupos solo en base a la cpu, nodos del grupo 0 y del grupo 1 estarían superpuestos ya que su valor de cpu sería similar. Lo mismo ocurre con el otro parámetro, nodos del 0, 2 y 4 tienen valores similares de memoria pero es el parámetro de cpu el que obliga a realizar una división entre ellos. Es por ello, que procesos que requieren que el procesador esté disponible y que, además, la memoria RAM libre bastante alta, se ejecutarían en nodos del grupo 1. Estos nodos no son lo suficientemente ideales para ejecutarlos, pero son con los que mayor rendimiento se sacaría del sistema.

Igual que se hizo en el caso anterior, se ejemplifica esta prueba con otra gráfica, **5.12.b** para comprobar si existe algún otro parámetro que pueda intervenir en la división. Para ello, se sustituye el valor de CPU por el de red. A la vista de la figura, se observa como no hay grupos diferenciados, sino que existen nodos de una agrupación con similares valores x-y que los de otras. En este caso, la memoria y la cpu, combinadas, diferencian, de forma clara, 4 grupos. En cambio, si se representan junto a otro tercer parámetro como es la red, ninguna de las 4 métricas es clave y existen nodos que se superponen entre grupos.

CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,1070618	10730636,992	78,905	16318,892
Cluster 1	0,0718634	16313468,4	3,303	2691,776
Cluster 2	0,388649	22440082,088	6,606	3028,248
Cluster 3	0,2947866	5111553,432	30,461	4205,9



a. Prueba 1 - Gráfica 4 clusters CPU - MEM

b. Prueba 1 - Gráfica 4 clusters RED - MEM

Figura 5.12: Prueba 1.4 - Gráfica 4 grupos

En conclusión, no siempre un único parámetro puede marcar la diferencia entre los grupos y sean dos o tres los que, en conjunto, el algoritmo utilice para relaizar el *clustering*. Aumentar el número de grupos provoca que el algoritmo no use un único parámetro, memoria libre restante en un equipo en el caso anterior, sino que necesita combinar dos de ellos, memoria y cpu para agrupar los nodos. No siempre van a ser estas dos medidas las más importantes, sino que puede haber casos en los que las métricas de tiempo o de red tengan más peso. Además, comparando con la anterior prueba donde el parámetro de red tenía una gran relevancia, en este caso, únicamente la cpu y la memoria cobran más peso. Es por ello, que se observa cómo van variando los grupos en función de los valores de las métricas que se monitoricen.

5.1.2. Ejecutar un nuevo proceso en un nodo para ocupar más recursos

En este apartado se van a realizar dos monitorizaciones más. La primera mantiene el valor de los nodos tal como están, pero en la siguiente, se escoge aquel nodo que tenga una diferencia de carga de cpu importante entre las dos pruebas. Este aumento de carga se puede realizar o bien añadiendo un proceso que haga consumir al sistema mucha carga o simplemente, habiendo realizado las dos ejecuciones, comprobar aquel nodo en el que varíe mucho esa carga (en el clúster existen otros muchos procesos que se están ejecutando en el mismo momento de la ejecución). Así, se ejecutan dos nuevas pruebas para los cuales se va a dejar pasar un tiempo entre una y otra ejecución. Una vez realizadas las dos monitorizaciones y habiendo ejecutado el algoritmo kmeans se obtienen los resultados. Para evitar un desarrollo tan largo como el de la anterior prueba no se entra en detalle en lo que ocurre si solo se tienen 2 o 3 parámetros y se ejecuta directamente con las 4 métricas totales.
Pero sí es importante comprobar donde está cayendo ese nodo a medida que se aumentan las particiones, ya que influye conocer si las medidas de ese nodo se asemejan a las del *centroide* en el que está cayendo para 2, 3 y 4 agrupaciones.

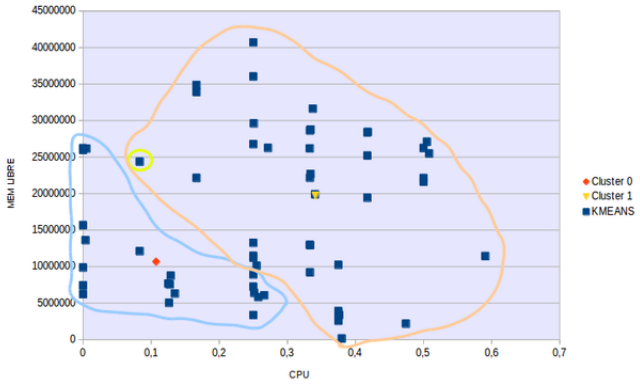
Así, se comienza mostrando los resultados para 2 particiones entre cada una de las ejecuciones.

En la **figura 5.13.a** los dos grupos están diferenciados, lo que es de suponer que el algoritmo se riga por cpu y memoria. Entonces a la hora de situar el nodo 65 se observa cómo podría discutirse el porque no se encuentra en el cluster 0 si su cpu es más cercana a él. Pero se ve a basar en la memoria a la hora de colocar el nodo. No hay datos que indiquen si el tiempo puede tener algo que ver, pero está claro que tiene mayor semejanza al cluster 0 respecto a esta variable. A la vista de la otra **figura 5.13.b** se ve cómo la elección de los clusters está hecha más en base a la memoria. Es por eso que el nodo pertenece al cluster 0 cuya memoria libre tiene un valor semejante a la de este nodo.

Comparando la colocación del nodo 65 en las dos gráficas se observa como en la primera, el nodo tiene CPU baja mientras que su memoria es alta, lo que le coloca en aquel cluster que tiene una mayor memoria. Al modificar la cpu de ese nodo, su memoria ha bajado y en la segunda de las gráficas se observa cómo se coloca en el clúster 0 con una memoria más baja.

CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,1075256	10662443.176	23.736	4276.132
Cluster 1	0,3414824	1967032.432	7.176	4537.936

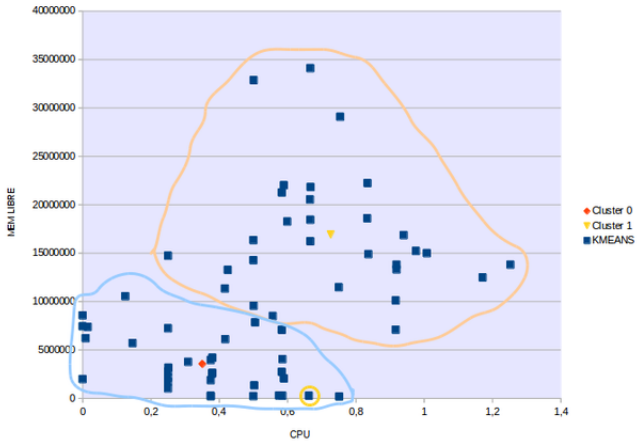
NODO	CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Nodo 65	1	0,0833	24365832	0	1160



a. Prueba 1 : Gráfica 2 clústers CPU - MEM

CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,3507	3546661.248	26.79	8783.547
Cluster 1	0,72645	16914845.952	8.93	6043.725

NODO	CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Nodo 65	0	0,6625	268096	0	1680

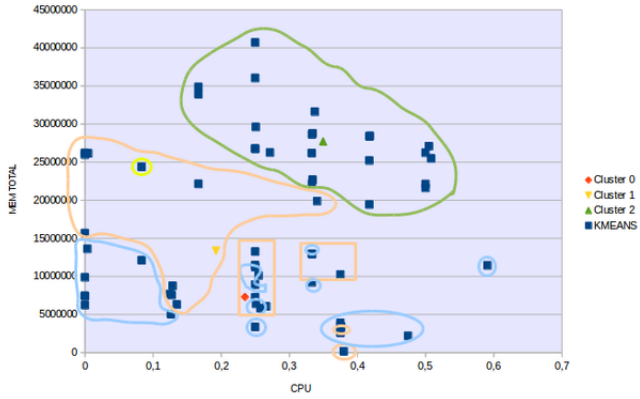


b. Prueba 1 : Gráfica 2 clústers CPU - MEM

Figura 5.13: Prueba 2: Cambio de nodo 65 - 2 grupos

CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,2351384	7284646,292	2,208	3839,792
Cluster 1	0,1926008	13348402,144	34,776	7941,388
Cluster 2	0,3497536	27714212,988	2,76	1832,628

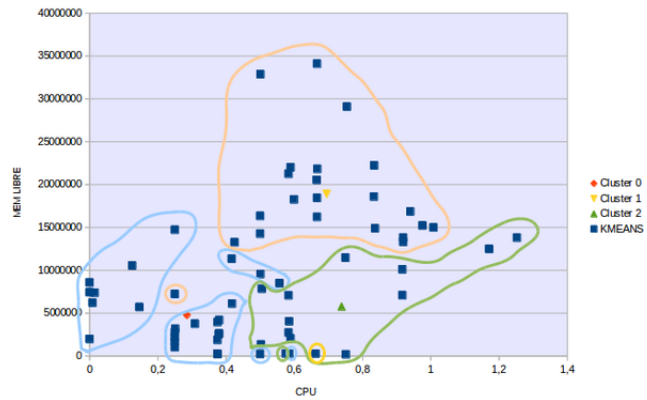
NODO	CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
65	1	0,0833	24365832	0	1160



a. Prueba 2 : Gráfica 3 clústers CPU - MEM

CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,28557	4774351,68	25,08	9589,377
Cluster 1	0,6951375	18892791,648	15,39	7816,551
Cluster 2	0,738975	5763324,528	9,88	2900,988

NODO	CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
65	2	0,6625	268096	0	1680



b. Prueba 2 : Gráfica 3 clústers CPU - MEM

Figura 5.14: Prueba 2: Cambio de nodo 65 - 3 grupos

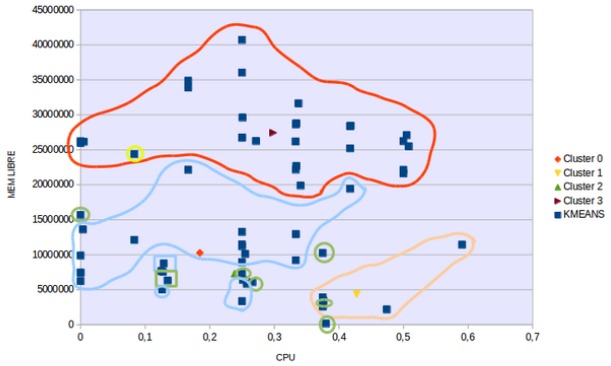
En la gráfica (**Figura 5.14.a**) de encima, se muestra cómo los clusters no escogen cpu o memoria únicamente para distinguir los grupos. Aún así, el nodo se sitúa en el clúster 1 ya que cuenta con una memoria y cpu más cercanas a las de su centroide como se ve en la tabla. En cuanto a memoria podría encontrarse en el clúster 2. Pero puede deberse al parámetro de red, el que se coloque más cerca de los nodos de su grupo. Modificando la crga del nodo,(**Figura 5.14.b**) el nodo se encuentra en el clúster 2 cuyo centroide tiene cpu y memoria más cercanas a él, así como de los nodos de su grupo. Ésto puede deberse a que el parámetro de red también influya a la hora de colocarlo.

Comparando ambas gráficas, ninguna de las dos muestra que sean cpu o memoria las que predominen en la selección. Es por eso que el que se encuentre en esos clústers pueda deberse a la influencia del parámetro de red.

Cualquiera de los clústers donde se sitúa tiene una media de red relativamente baja respecto a los otros dos grupos, semejante a su valor.

NODO	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,1849204	10255479,696	6,21	7679,584
Cluster 1	0,4277392	4354509,236	6,21	4188,864
Cluster 2	0,2386832	7284646,292	70,518	4450,668
Cluster 3	0,298354	27429338,552	2,346	1832,628

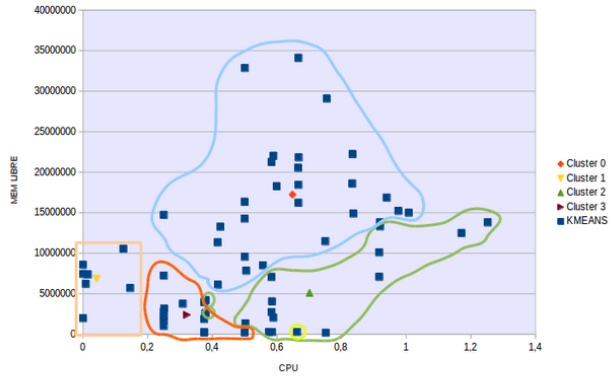
NODO	CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
65	4	0,0833	24365832	0	1160



a. Prueba 2: Gráfica 4 clústers CPU - MEM

NODO	CPU	MEM LIBRE	Tiempo IO	RED
Cluster 0	0,648795	17221768,56	8,36	6124,308
Cluster 1	0,042585	6820502,4	18,43	13699,11
Cluster 2	0,7014	5115376,8	12,35	8541,798
Cluster 3	0,3218925	2387175,84	46,17	5560,227

NODO	CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED
65	2	0,6625	268096	0	1680



b. Prueba 2: Gráfica 4 clústers CPU - MEM

Figura 5.15: Prueba 2: Cambio de nodo 65 - 4 Grupos

Al igual que pasaba con 3 agrupaciones, en la **Figura 5.15.a** se observa cómo cpu y memoria puedan distinguir completamente todos los clústers. Aún así, el 3 donde se encuentra el nodo sí se distingue del resto. Por tanto se puede comprobar cómo su colocación en este caso sí depende de la memoria ya que ambos valores, el del nodo y el del centroide de su clúster, son bastante similares. Mirando el resto de parámetros, como es el de red, también son bastante similaresambos.

En la última gráfica (**Figura 5.15.b**) el nodo se coloca de la misma manera que se colocoba con 3 agrupaciones, en el 2. Los valores de los dos clústers (en verde) son bastante similares así que la razón para que el computador se coloque en este grupo es la misma, aunque diferiría en el tema de red, ya que no se basaría en él para colocarse y sí en base a cpu y memoria.

Comparando ambas gráficas se observa la colocación del nodo en clústers totalmente diferentes. El primero se situaba en el cluster cuya memoria era bastante alta, similar a la de su valor, mientras que en el segundo caso es más semejante al clúster cuya cpu es alta mientras que la memoria todo lo contrario.

Con ésto se concluye que si se introduce un nuevo proceso, como ya se sugería en las pruebas anteriores, se modifica el estado de los componentes, sobre todo de la CPU y la memoria. Es por ello que en un nodo que no está sobrecargado se ejecuta un nuevo proceso, su uso de CPU aumenta, al igual que la memoria libre disminuye, moficándose de grupo. En consecuencia de ello, pasa de colocarse en grupos cuya cpu es baja mientras que la memoria libre es media o alta, a colocarse en particiones donde la cpu es mayor y la memoria libre es baja.

5.2. Discusión

En la última parte del capítulo 5 se comentan los resultados extraidos anteriormente, de una forma general. Primero, se comienza con una comparativa de lo que ocurre cuando se incrementa el número de particiones con un mismo valor en un conjunto de nodos y, seguidamente, se hace una nueva comparativa en función de añadir más parámetros. Además, se recalcan aquellas métricas que hayan sido más importantes a la hora de escoger los grupos de cómputo y la razón de aquellas que no han sido tan relevantes. Se finaliza añadiendo algunos aspectos que se han analizado pero no se han incluído en los resultados.

Así, primero se compara qué es lo que ha ocurrido en las pruebas anteriores cuando se tiene un mínimo número de parámetros, dos, para seleccionar los agrupamientos.

Cuando se tienen **dos parámetros** es mucho más fácil representarlos gráficamente y distinguir cual de los dos o, ambos dos con los más importantes. En las distintas pruebas realizadas sustituyendo unos parámetros por otros se ha llegado a la conclusión de que en estas pruebas la división entre las particiones es mucho más clara, ya que

suele ser uno de los dos parámetros el que escoga el algoritmo para dividir los equipos.
A la hora de **añadir una nueva** métrica al algoritmo hay que tener en cuenta dos posibles resultados. Uno, o que esa métrica no tenga importancia en el resultado final, o dos, ese parámetro, influya, con mucha o poca relevancia, en la división de agrupaciones. Teniendo dos métricas prioritarias durante todas las pruebas como son carga de cpu y memoria libre se ha querido añadir una tercera para comprobar si ya los grupos, no se diferencian tan claramente. Y eso es lo que ha ocurrido al añadir el parámetro de red, donde había una pequeña parte de nodos que creaban confusión porque, a simple vista en las gráficas 2D de coordinadas cpu y mem, se encontraban situados dentro de grupos a los cuales no pertenecían. Es decir, se distinguen de forma clara 2, 3 o 4 grupos pero se visualizaban nodos dentro de ellos que pertenecían a otro distinto. Esto quiere decir que esos nodos tienen más en cuenta el tercero de los parámetros. Así, al añadir una nueva representación donde se incluya ese tercer atributo, los nodos que antes creaban confusión ahora están situados cerca de la mayoría de nodos del agrupamiento al que pertenecen.
El primer objetivo de este trabajo era tener por lo menos 4 parámetros que se pudieran medir en cada nodo. Por tanto se han realizado pruebas ejecutando el algoritmo con todas las métricas. Lo que se ha obtenido primeramente en las pruebas es lo que se lleva explicado durante todo el proceso, los dos principales parámetros dividen en gran medida los grupos, aún habiendo alguna que otra excepción, como ocurre cuando se tenían 3 grupos. Y luego existe una serie de puntos que traen a equívoco el pensar en unicamente dos principales medidas. Esos nodos tiene en cuenta otra métrica que no tiene que ver con alguna de las dos. Seguramente coincida que haya nodos que se fijen en la red y otros nodos, por su parte, que tengan más en cuenta el tiempo.

A la hora de realizar una comparativa en lo que sucede al aumentar el número de agrupaciones, que es uno de los puntos clave de las pruebas, hay que fijarse que pocas veces se tiene en cuenta en los resultados. En alguna ocasión ha ocurrido que el tener 4 particiones ha provocado que un conjunto de nodos que estaban localizados en una posición cercana se han dividido en dos. Pero ésto no ha provocado que nodos que están situados en una posición en un grupo se muevan a otra partición que esté en una posicióon lejana sino que si se han modificado de grupo es a aquellos que, seguramente, compartan nodos cercanos del anterior.

Con respecto a las métricas que tienen más relevancia a la hora de dividir los grupos, se concluye que en la mayor parte de las pruebas realizadas son la carga de cpu y la memoria. En alguno de los casos es la red la que provoca que no se identifiquen del todo claro y que los nodos se encuentren bastante espandidos. Pero en la mayoría se observa como el número de nodos al que la red le influye es bastante bajo y que las dos primeras medidas tienen mayor peso en todo el conjunto. Bien es así, que en aquellas en las que están bien diferenciados se suele tener en cuenta mayormente a una de ellas para realizar los trazos, o si no es así, son las dos conjuntamente las que hay que tener en cuenta para la selección. ¿Qué es lo que ocurre con el parámetro de **TIEMPO** de entrada/salida? En todos los casos, no es relevante a la vista de los resultados. Cuando se tenían dos parámetros y uno de ellos era el tiempo, el algoritmo escogía el otro para realizar la selección. Si se añadía como tercer parámetro en las siguientes pruebas, tal vez, uno o dos nodos se encontraban en clusters a los que no pertenecían. Si ese tercer parámetro era el de red, ese número de nodos aumentaba. Y lo mismo con 4 parámetros, solo uno o dos nodos tenían en cuenta el tiempo.
La razón de mayor peso a todo ésto es analizar el rango de valores del tiempo. Mientras que las otras tres variables mantienen expandidos los nodos en su rango, el tiempo tiene la mayor parte de los nodos en los valores bajos y solo algún nodo se dispara provocando que sea el máximo en el rango de valores de tiempo. Esto provoca que al normalizar los valores de tiempo la mayoría estén en torno a 0 - $\simeq$ 0.1. y el promedio de tiempo en el centroide sea bajo para ser prioritario en la ejecución del algoritmo. Cabe decir que en otras pruebas que no se han analizado en profundidad, como el caso reflejado en la prueba 1.4 ocurre lo mismo con el parámetro de red y uno de los nodos tiene un valor demasiado alto quedando los demás nodos en la zona baja de la gráfica. En ese caso, la métrica no tendrá tanta repercusión como en la prueba 2 analizada donde los nodos estaban mejor expandidos respecto al atributo dew red. Por tanto, es una variable que se podría haber suprimido a la hora de realizar las pruebas, y no crear pequeñas discrepancias que a la hora de la verdad no tienen tanta importancia.

En la mayoría de las pruebas se ha ido comentando qué grupo sería más ideal a la hora de ejecutar un proceso, lanzar una nueva aplicación o tener un nodo que actúe como servidor de aplicaciones web. Es por ello, que los grupos con carga de cpu baja son mejores para la ejecución de aplicaciones o procesos que requieran del uso del procesador. Aquellos con memoria libre de RAM más alta son perfectos para lanzar aplicaciones que requieran mayor espacio de ejecución. Y aquellos grupos con un tráfico de red más bajo pueden ejecutar nodos que actúen como servidores de aplicaciones web donde el número de transacciones con el usuario es muy alta.

Además, la última de las pruebas comprueba perfectamente el correcto funcionamiento del algoritmo. Si uno de los nodos modifica uno de sus parámetros más importantes, es seguro que cambie de agrupación pasando a colocarse en una partición cuyos valores en los parámetros sean más similares. Si el nodo pasa de tener una carga de sistema

muy baja a una relativamente alta, es muy probable que esa memoria baje. Al modificar los dos parámetros más importantes su colocación va a variar, por tanto se ve cómo se encuentra en aquel grupo más similar a los nuevos valores de esas variables. Esta prueba confirma el correcto funcionamiento de k-means. El objetivo principal es agrupar nodos en función del valor conjunto de todas las métricas, por eso, en este caso se observa cómo si se varía uno de los valores de las métricas más relevantes de un punto cambía de grupo.

Ya se ha comentado al principio de esa sección que se iban a analizar aspectos que no se hayan incluido en las pruebas. Uno de los puntos claves es el número de grupos en el que se divide el conjunto de nodos. Como ya se ha visto, incrementar el número de grupos no ha tenido mucho que ver en la creación de discrepancias entre los nodos, solo trataba de dividir los equipos en función de las métricas. En varias ocasiones se ha tenido que el número de nodos en un cluster era bastante menor que en uno de los otros grupos, algo normal cuando se ejecuta k-means, ya que depende del valor de las medidas, si algunas se encuentran más alejadas de otras, se formarán grupos con estos puntos más alejados. Sobre todo, habría que destacar cuando se tienen 4 agrupaciones y provoca una división entre en dos subconjuntos de nodos que podría pertenecer a un único subconjunto. Ahí se llega a la conclusión de que se tiene un número elevado de divisiones para el rango de nodos que se tienen a prueba. Ese es prácticamente el porqué de no añadir más particiones a las pruebas. En la mayoría de las realizadas con 4 grupos se tenía que éstos eran bastante equitativos en cuanto conjunto de nodos, si alguno destacaba en ese valor es que sus nodos estaban en puntos más cercanos. Seguro que añadir un nuevo clúster puede provocar que haya grupos que contengan un número de nodos demasiado bajo para tener ésto en cuenta. En vista de los resultados se puede comprobar si es óptimo el número de nodos que se ha escogido anteriormente. Así, se va a analizar qué es lo que ocurriría si se ejecutara el algoritmo con las mismas medidas que en la prueba 1.3 y teniendo 5 grupos.

CLUSTER	CPU	MEM LIBRE	Tiempo IO	RED	Nº NODOS
Cluster 0	0,1440751	14773590,348	5,605	1084,176	7
Cluster 1	0,1930723	6348980,976	32,745	4961,611	21
Cluster 2	0,3546464	23564487,084	4,13	4946,553	9
Cluster 3	0,1878226	29425084,908	94,695	1136,879	4
Cluster 4	0,3680623	21366762,9	14,16	1490,742	22

Cuadro 5.1: Resultados con 5 grupos

A la vista del cuadro 5.1 se observan tres particiones, 0, 2 y 3 que cuentan con menor número de nodos. No se va a entrar mucho en detalle de por qué esta división de grupos sino que es mejor compararlo con la **figura 5.9** dónde representando los mismos datos se tenían 4 particiones. En la figura 5.9 ya se veía un grupo 2 en verde que contaba con menor número de nodos pero a la vista general la división hecha por parte del algoritmo era bastante clara y equitativa. En cambio, en esta última, definir un nuevo grupo empeora los resultados.

Como conclusión decir que está claro que si en un futuro se añaden más nodos los 4 grupos se quedarían cortos y se necesiten definir más grupos. Pero con el número de nodos escogidos en las pruebas es más que suficiente con los 4 grupos definidos.

Otra de las cosas en las que no se ha hecho incapié es en el número de veces que se ejecuta k-means con los mismos datos en el fichero de entrada. Cada vez que se han realizado las pruebas se han analizado los resultados que devolvía la primera ejecución, no se ha realizado en cada una de las pruebas una nueva ejecución para comprobar qué es lo que ocurriría si volvemos a ejecutar el algoritmo. Eso sí, es seguro que la posición de las particiones variaría, así como los nodos dentro de ellos.

Además, nunca se ha forzado a que el algoritmo haga más de 1 iteración para colocar perfectamente los nodos dentro de un centroide. Dicho algoritmo no realiza una nueva iteración porque a la primera es capaz de comprobar qué nodos se encuentran dentro de un grupo. En vista de los resultados y de que para cada una de las discrepancias existe un por qué, no sería necesario una nueva iteración como podría haberse pensado en vista de las gráficas de CPU - MEM y que esos nodos que crean confusión se colocasen 'correctamente'.

Capítulo 6

Conclusiones y Trabajos Futuros

En este capítulo se explica si se han cumplido cada uno de los objetivos propuestos en el primer capítulo y si se ha alcalzado el próposito principal del mismo. Además, se detalla el alcance que puede tener este proyecto indicando algunos trabajos futuros que se pueden realizar.

6.1. Conclusiones

En la primera sección de este capítulo se explican si se han cumplido los objetivos y las conclusiones que se extraen después de desarrollar el proyecto.

El primer objetivo era definir una serie de métricas que ayudasen a monitorizar el estado de los recursos, pero sin salirse del objetivo principal, es decir, se han definido únicamente aquellas que comprueban el correcto funcionamiento del algoritmo k-means para un pequeño número de métricas. Supervisar los recursos de los que consta un entorno es muy extenso, hay métricas que ayudan a conocer el estado de un equipo en función del tipo de trabajo que haga, analizar, en este trabajo, métricas como la temperatura, el número de dispositivos utilizados, el número de servicios que se ejecutan por minuto, no es relevante. Por eso, se han analizado los parámetros más interesantes en un *clúster HPC* y se han escogido, unicamente cuatro: carga de cpu, memoria libre en RAM, tiempo de entrada y salida en disco, y uso de la red.

Un segundo objetivo marcado, era utilizar una herramienta que permitiese controlar los recursos de forma continua. Para ello, se ha instalado y configurado correctamente *Graphite*. Mediante la construcción y posterior ejecución de un script global o scripts para cada una de las métricas, *Carbon* ha permitido obtener los valores de las métricas en los últimos instantes medidos, y almacenarlos en archivos independientes de la base de la base de datos, *whisper*. Mientras el script continue ejecutándose, *Carbon* seguirá actualizando cada una de las medidas en la base de datos.

En el tercer capítulo, mediante la aplicación web de *Graphite*, existe una interfaz gráfica, *dashboard*, donde se representan las métricas de los nodos que más interesen a los supervisores tener controlados. Además, mejorando el control y rendimiento de los equipos se pueden creargráficas que contengan métricas de distintos nodos para poder comparar la situación en la que se encuentra cada uno en cada instante de tiempo. Por ejemplo, si se representa la carga de dos equipos y hay una diferencia significativa entre ambos, se puede enviar parte del trabajo de un nodo a otro, todo ello en el menor tiempo posible. Si se quieren controlar sólo las métricas de un equipo se pueden representar en un mismo gráfico o por separado, en la misma pantalla. Por otro lado, *graphiteweb* también permite guardar los gráficos de una sesión a otra.

En el cuarto objetivo se perseguía encontrar el algoritmo que mejor se adecuara a nuestro objetivo, analizando los distintos tipos de técnicas de clustering existentes: jerárquicas, en base a la densidad o a la distancia, etc. *K-means* es un algoritmo de *clustering* que analiza una gran cantidad de nodos y los divide en "k" grupos con características relacionadas entre sí. Recordando que parte de nuestro objetivo es agrupar el conjunto de nodos en distintas agrupaciones en función del valor de una serie de métricas, k-means es el algoritmo idóneo.

Después de explicar los resultados obtenidos por las pruebas en el capítulo anterior, se observa cómo el algoritmo k-means funciona perfectamente con, únicamente, 4 parámetros monitorizados. Ha sido útil representar gráficamente los grupos de nodos, para visualizar que si se ejecuta una nueva aplicación que necesita el uso del procesador o una alta memoria libre, qué grupos de nodos serían más ideales para cubrir esas necesidades y aprovechar mejor el uso de los recursos. La última prueba realizada, ha sido clave para comprobar el correcto funcionamiento del algoritmo. Modificandos alguno de los parámetros de un nodo y ejecutando k-means sobre estos nuevos valores se comprueba

cómo el nodo cambia de grupo y se coloca en aquel donde sus valores nuevos son similares a los del *centroide* de la agrupación. Por tanto, se concluye cómo el uso de esta técnica de clustering agrupa nodos en función de que los valores de métricas estén relacionados entre sí.
Uno de los problemas que se explicaba en el primer capítulo es que los nodos de un Centro de Datos se dividen para crear grupos que realicen una determinada función en el sistema sin basarse, en gran medida, en la dependencia con los componentes de un equipo. Esto ayuda a mejorar el rendimiento de los clústers ya que, se puede enviar carga de trabajo de nodos que están sobrecargados, a aquellos grupos de nodos con parámetros de carga más bajos, transcurrido el menor tiempo posible. Observando los buenos resultados proporcionados por k-means se puede destinar una agrupación para un trabajo u otro. Por tanto, en vez de tener un conjunto de equipos fijo realizando un trabajo, se enviará dicho trabajo a uno de los nodos de la agrupación con carácterísticas más propensas para realizarlo.

Como último objetivo, se querían analizar aquellos aspectos que no se han reflejado en las pruebas. Es importante definir el número de clústers óptimo teniendo en cuenta el conjunto de nodos que se monitorizan. En este trabajo, la cantidad es baja y los grupos son acordes a ellos. Si k-means se ejecutase sobre casos más reales, se tendrían que definir muchos más subconjuntos de nodos debido a que hay DataCenters con enormes cantidades de computadores. En estos casos, sería más útil utilizar algoritmos que defininen el número de agrupamientos en base al número de equipos que componen el conjunto.

6.2. Trabajos Futuros

En la segunda sección del capítulo se detallan algunos trabajos futuros que se pueden realizar en vista de las conclusiones de este proyecto.

- Observando lo bien que funciona k-means sobre, únicamente, 4 parámetros, ya tenemos un buen punto de partida para cuando se quiera trabajar en casos más cercanos a la realidad, donde el número de parámetros ya no es tan limitado. En un futuro, en base a la función para la que se dediquen un conjunto de nodos, se evaluará unas métricas u otras. En un DataCenter existen grupos de equipos que se dedican a una función u otra. Cada trabajo para el que se destina un nodo está directamente relacionado con los recursos que se utilicen. Por lo tanto, es esencial evaluar distintos parámetros para cada grupo, por lo que al final la cantidad de métricas a monitorizar es mucho mayor.

- En este trabajo, se ha utilizado el algoritmo k-means porque se asemejaba más a la evaluación que se quería realizar de las métricas escogidas. Evaluar el rendimiento de cada uno de los grupos de un Centros de Datos necesitará escoger una serie de métricas diferentes lo que provoca que ya no se use el mismo criterio a la hora de agrupar los equipos. Como ya se ha explicado, existen cientos de técnicas de Minería de Datos para cada uno de los propósitos requeridos, es por eso que antes de escoger un tipo de algoritmo es necesario explorar los datos obtenidos y saber lo que se quiere hacer con ellos.

- Dependiendo del tamaño del Centro de Datos, la cantidad de recursos que se monitorizan es distinta. Entonces, es esencial automatizar las técnicas que permitan determinar el número de grupos óptimos para un conjunto de equipos sobre todo para aquellos entornos donde la cantidad de computadores es enorme. Es por ello que se pueden utilizar una serie de algoritmosque ayuden a determinar el valor *"k"*, aquellos más conocidos son:

  - *Silhouette*: Este algoritmo define una medida de cómo de cerca los datos se emparejan con su grupo y no a las agrupaciones vecinas cuya distancia desde el nodo es más baja. Una medida con valor 1 implica que el dato está en el grupo correcto mientras que un valor de -1 está en el grupo equivocado.
  - *Bayesian Information criterion (BIC)*: Es un criterio para la selección de un modelo de k grupos entre un conjunto de modelos. Se basa, en parte, a la función de probabilidad y a otro criterio llamado Akaike information criterion (AIC). AIC es una medida que selecciona un modelo en base a un conjunto de datos. Cuando se realiza el ajuste de modelos, es posible aumentar la probabilidad mediante la adición de parámetros, aunque puede provocar un sobreajuste.

- Si aumentar el rendimiento de un sistema pequeño es fundamental para poderle sacar el mayor provecho, en una empresa esta cuestión adquiere mayor importancia. Es por eso, que el proyecto siempre ha estado focalizado a casos prácticos que se puedan realizar en una empresa o institución.

Bibliografía

[1] *¿Qué es un Data Center?*http://www.acens.com/blog/que-es-un-data-center.html

[2] *Concepto de Green Computing* http://es.wikipedia.org/wiki/Green_computing

[3] Matthew L. Massie, Brient N.Chun y David E. Culler, *The ganglia distributed monitoring system: design, implementation and experience*, University of California, USA, April 2004. http://ganglia.info/papers/science.pdf

[4] Matt Massie, Bernard Li, Brad Nicholes y Vladimir Vuksan, *Monitoring with Ganglia*, Noviembre 2012.

[5] *Airplanes company* www.orbitz.com, 2001.

[6] Chris Davis, *Graphite Documentation-Release 0.10.0*,18 de Octubre 2014.

[7] Thomas Finley y Thorsten Joachims. *Supervised K-means Clustering*. Department of Computer Science, Cornell University, USA.

[8] *Algoritmos de Clustering* http://en.wikipedia.org/wiki/Cluster_analysis#Algorithms

[9] Andrea Vattani. *K-means Requires Even in the Plane*. Discrete & Computacional Geometry journal (2011), 596-616. 7 de Diciembre, 2009.

[10] R.O., Hart, P.E. y Stork, D.G., *Pattern Classification*. Wiley, New York, 2000.

[11] Martín, *Round Robin Databases*, 27 de Marzo 2008. http://brigomp.blogspot.com.es/2008/03/round-robin-databases.html

[12] Django Project. https://www.djangoproject.com/.

[13] David Hand, Heikki Mannila y Padhraic Smyth. *Principles of Data Mining*, 293-305. Massachusetts Institute of Technology, 2001.

[14] *Concepto de Distancia Euclídea* http://en.wikipedia.org/wiki/Euclidean_distance

[15] *Sistema de archivos proc* http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-proc.html

[16] *Determinar el número de clústers en un conjunto de datos* http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set

[17] Departamento de Arquitecttura y Tecnología de Computadores, *Sistema de colas SGE EE en los servidores de cálculo paralelo del CPF 3MARES: Cluster HPC CALDERON*. Facultad de Ciencias, Universidad de Cantabria.

[18] *Manual de Referencia Red Hat Enterprise Linux 4. Capítulo 20: Protocolo SSH.* http://www.gb.nrao.edu/pubcomputing/redhatELWS4/RH-DOCS/rhel-rg-es-4/ch-ssh.html.