Benquerenca Mendes, Nicolás Martín

Sistemas de archivos paralelos con aplicaciones de Machine Learning

2022

Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Carrera de Ingeniería en Informática

PRÁCTICA PROFESIONAL SUPERVISADA

Informe final

Sistemas de archivos paralelos con aplicaciones de Machine Learning

Benquerenca Mendes, Nicolás Martín

Florencio Varela, febrero de 2022

Estudiante
Benquerenca Mendes, Nicolás Martín
DNI: 36843895
Nº de Legajo: 3868
nicobenquerenca@yahoo.com.ar
Cantidad de materias aprobadas al comienzo de la PPS: 44
Práctica Profesional Supervisada (PPS) enmarcada en artículo (4 ó 7) de la Resolución (CS) 103/16

Organización donde se realiza la PPS.
Universidad Nacional Arturo Jauretche
Av. Calchaquí 6200, Florencio Varela, Buenos Aires
+54 11 4275 6100
Sector: Programa Tecnologías de la Información y la Comunicación (TIC) en aplicaciones de interés social, Instituto de Ingeniería y Agronomía

Tutor por la organización
Prof. Ing. Bond, Román
rbond@unaj.edu.ar

Docente supervisor por UNAJ
Prof. Mg. Encinas, Diego
dencinas@unaj.edu.ar

Docente tutor por el Taller de Apoyo para la Producción de Textos Académicos (UNAJ)
Lic. y Prof. Bein, Paula
paula.bein@gmail.com

Coordinador de la carrera de Ingeniería en Informática
Dr. Ing. Martín Morales
martin.morales@unaj.edu.ar

Resumen

En la presente Práctica Profesional Supervisada (PPS) se propone la investigación, análisis y evaluación del impacto de aplicaciones del tipo Machine Learning en sistemas de archivos paralelos, a nivel de rendimiento y uso de recursos. Para realizarlo se plantea el estudio en particular del sistema de archivos paralelo BeeGFS, como infraestructura, y aplicaciones de reconocimiento de imágenes, como herramientas de benchmark para obtener los resultados necesarios y posterior análisis. Los sistemas de archivos paralelos nos permiten incrementar el rendimiento de los “File Servers” que requieren de mayor capacidad de respuesta a operaciones de lectura y escritura por accesos recurrentes y concurrentes a datos, donde los sistemas de archivos convencionales como “Network File System” no pueden satisfacer esta capacidad, entre otras grandes ventajas.

Toda la implementación se realizará localmente en un cluster montado con VMware Workstation.

Abstract

This Supervised Professional Practice (PPS) proposes the investigation, analysis and evaluation of the impact of Machine Learning-type applications on parallel file systems, at the level of performance and use of resources. To do this, the study of the BeeGFS parallel file system, as an infrastructure, and image recognition applications, as benchmark tools to obtain the necessary results and subsequent analysis, is proposed. Parallel file systems do not allow to increase the performance of "File Servers" that require a greater response capacity to read and write operations due to recurrent and concurrent access to data, where conventional file systems such as "Network File System" do not can satisfy this capacity, among other great advantages.

All deployment is done locally on a cluster mounted with VMware Workstation.

Agradecimientos

En primer lugar, deseo agradecer al Prof. Mg. Encinas Diego y al Prof. Ing. Bond Román por acompañarme durante la realización de este trabajo, escuchando y respondiendo muy amablemente todas las dudas técnicas que fueron surgiendo e invitándome a compartir mis conocimientos y ganas junto a un gran grupo de personas.

Asimismo, agradezco a la Lic. y Prof. Bein Paula por su colaboración constante en las revisiones de la redacción del presente trabajo. De igual forma, extiendo el agradecimiento al Dr. Ing. Martín Morales por su apoyo y rápida gestión para la realización del mismo.

Por sus enriquecedoras charlas en conocimiento y motivacionales, un agradecimiento especial al Ing. Oscar L. Cortes Bracho.

A mi padre, mi madre y mi hermana. Por su comprensión, paciencia y cariño que me brindaron durante años y que me dieron fuerza para seguir adelante y llegar a esta etapa.

A mi esposa, desde su amor y soporte incondicional, brindándome las energías necesarias para avanzar y no abandonar.

Sin ellos, junto a muchas personas más, esto no hubiera sido posible.

A todos, muchas gracias.

Índice

Índice de figuras

1. Introducción

La era digital atraviesa año tras año nuevos desafíos tecnológicos, por lo que indefectiblemente aparecen nuevas barreras a superar tanto a nivel de hardware como de software. Hace unos años, con el crecimiento continuo de aplicaciones y usuarios como por ejemplo en las redes sociales, los sistemas tradicionales como *Network File System (NFS)* comenzaron a mostrar sus falencias respecto a soluciones que demandaban alto rendimiento. Así ocurrió en otros ámbitos tecnológicos como la migración de *Asymmetric Digital Subscriber List (ADSL)* a fibra óptica, los sistemas de archivos se enfrentaron también a un mismo factor común: el aumento de demanda de recursos. La época de la centralización, escalabilidad vertical y las soluciones unificadas en arquitecturas de hardware empezaron a mostrar sus limitaciones y se inició la migración a soluciones descentralizadas. Por eso, fue necesario, por un lado, la innovación tecnológica y, por otro, los sistemas de archivos paralelos entraron en auge. Con características como escalamiento horizontal, alta disponibilidad, duplicidad de información, acceso concurrente, la barrera se fue superando.

Hoy el desafío es distinto. No sólo por el simple hecho de que día a día aumenta la demanda, sino porque ésta actúa de modo distinto. Las aplicaciones de hace cinco años ya no se comportan igual, y los usuarios tienen nuevas necesidades. Estos últimos, ya no sólo no les alcanza con tener acceso a los programas sino que se está en un periodo en donde se necesita que sean inteligentes. Es decir, se espera que los GPS den la mejor ruta de un camino no sólo basada en kilómetros, si la frase que se está escribiendo en un procesador de texto tiene sentido, o se creen subtítulos en un video en vivo, que las recomendaciones estén basadas en el estado de ánimo del usuario o hasta que detecten patrones para la detección temprana de enfermedades. Estas son algunas de las funcionalidades con las que ya se está interactuando y que vienen a dar sentido a esta nueva era digital inteligente: la de la Inteligencia Artificial (IA).

Debido a que esta transformación avanza rápidamente, es necesario empezar a supervisar los sistemas, analizar la información recolectada y desarrollar herramientas de monitoreo y benchmark específicas que permitan- desde el lado del hardware- determinar si las actuales soluciones disponibles de sistemas de archivos paralelos están preparadas para el cambio que se avecina. Para contribuir a actuales y futuras investigaciones sobre esta cuestión, en el presente trabajo se buscará aportar información mediante la monitorización, sobre el impacto y comportamiento de aplicaciones del tipo Machine Learning en sistemas de archivos paralelos, como BeeGFS.

2. Sistemas de archivos paralelo

Se da este nombre a un tipo de sistema de archivos distribuido. A su vez, se diferencian de los convencionales -como NFS- porque almacenan datos a través de varios

servidores conectados a la red, denominados comúnmente como “nodos”, que utilizan la técnica de stripes y storage targets, que se explicará en posteriores apartados.

Este método de trabajo permite acceso simultáneo y de alto rendimiento a los datos almacenados en los servidores, a través de múltiples canales mediante procesos informáticos que generan las aplicaciones. Al ser escalado fácilmente, permiten trabajar enormes volúmenes de datos sin inconvenientes.

Una característica que comparten todas las distribuciones de sistemas de archivos paralelos es la utilización de diversos servicios dependiendo de la función que tengan, en diferentes servidores. Por ejemplo, los de metadata y almacenamiento son comunes entre ellos; algunos software incorporan servicios de supervisión, gestión y administración. Esto último, sumado a la utilización de múltiples canales, permite a los clientes acceder por caminos independientes tanto a los servidores de metadatos como a los que almacenan los datos, a diferencia de como ocurre en los sistemas tradicionales donde todos acceden a un mismo nodo.

Algunas arquitecturas populares, con características distintivas y principalmente orientadas a HPC, son:

- General Parallel File System (GPFS)
- BeeGFS
- Lustre
- GlusterFS
- OrangeFS

2.1. Ventajas

- Altamente escalables: Con características sencillas permiten incorporar más nodos a la arquitectura y así, de ser necesario, agrandar el rendimiento del sistema.
- Soporta replicación de datos: Es la característica que hace referencia a copiar y mantener los datos actualizados en varios nodos.
- Alto rendimiento: Permiten trabajar con grandes volúmenes de datos sin inconvenientes.
- Alta disponibilidad: Es la característica que permite, a través de distintas técnicas y en caso de falla de algún servidor, no perder la operatividad del sistema

2.2. Desventajas

- Requieren de mayor conocimiento y configuración por parte del administrador.

- Gestión más compleja que sistemas centralizados.

2.3. Escalamiento

Es una característica que, a menudo, se menciona en el informe. Hace referencia al aumento de recursos dentro de una infraestructura. Se divide en dos grandes conceptos: el escalamiento horizontal y el vertical.

2.3.1. Escalamiento Vertical

Señala al incremento de hardware dentro de un nodo específico. Es limitada y explica el por qué de la importancia del escalamiento horizontal. Su barrera radica en los límites del hardware que lo compone. Por ejemplo, la placa madre de un servidor soporta un máximo de 48 Gigabyte (GB) de memoria RAM.

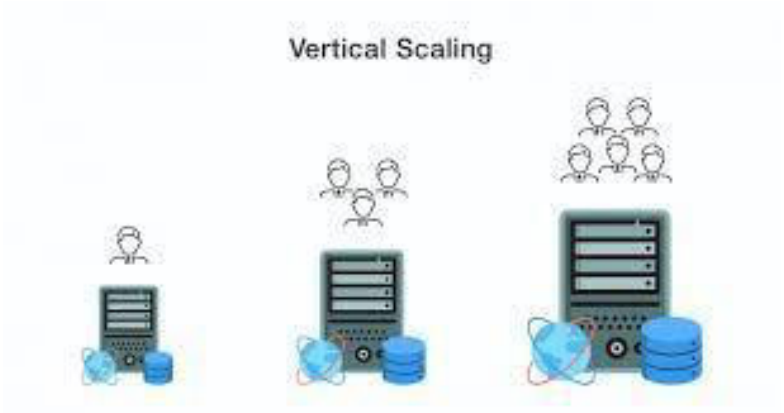

Figura 1. Escalamiento Vertical

2.3.2. Escalamiento Horizontal

Se refiere a la ampliación de poder de cómputo mediante la adición de nodos a la infraestructura. Su única limitación es el presupuesto o espacio disponible. Representó gran parte de la innovación durante el auge de los sistemas de archivos paralelos y, actualmente, también se utiliza como la principal herramienta para contrarrestar las necesidades de aplicaciones de IA. Esto, sumado a la técnica de stripes y storages targets, permite obtener resultados de alto rendimiento[1].

[1] Esto se observa en el trabajo de investigación "Rendimiento de sistema de archivos en arquitecturas distribuidas y paralelas" disponible en http://conaiisi2020.frsfco.utn.edu.ar/actas.html

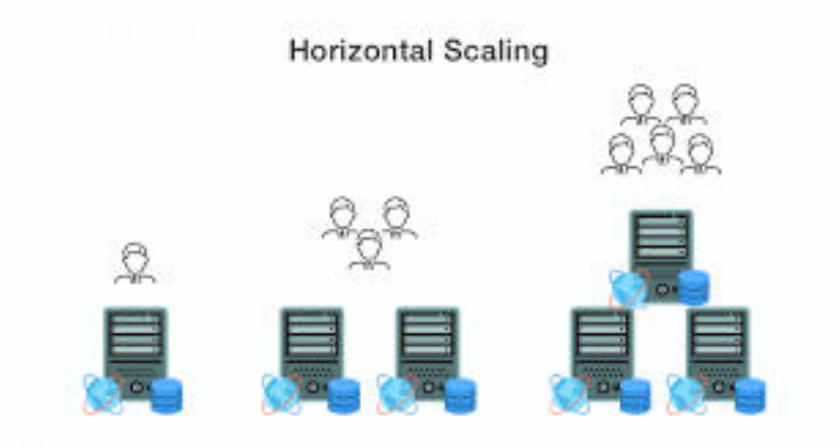

Figura 2. Escalamiento Horizontal

2.4. Sistema de archivo paralelo y distribuido

Una manera de diferenciar ambas soluciones es a nivel de arquitectura, como se puede visualizar en la siguiente imagen.

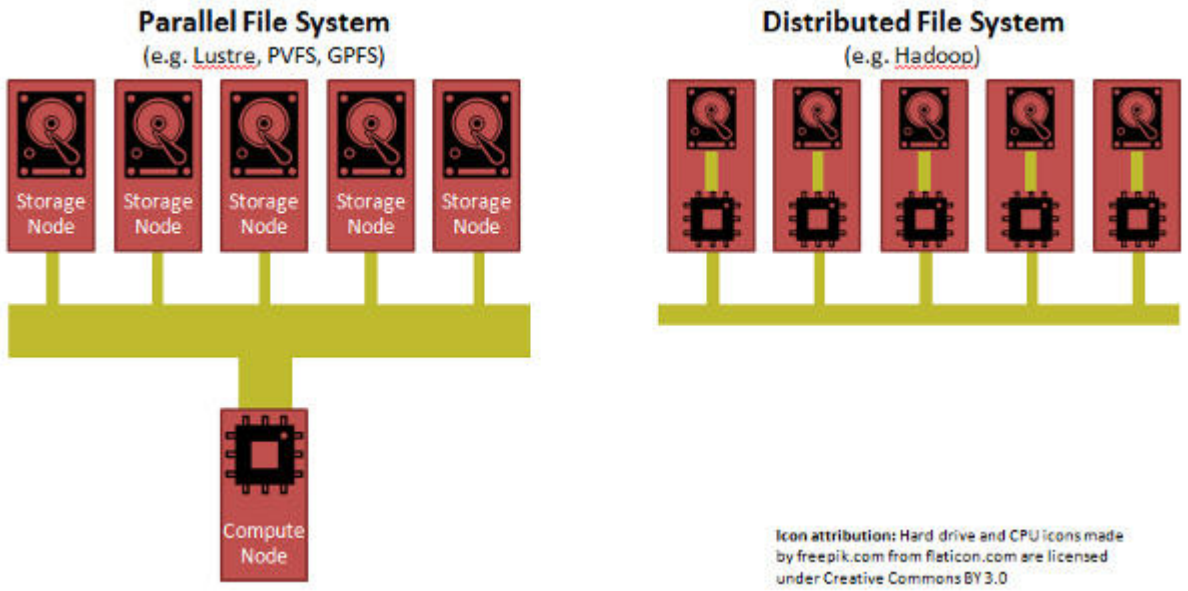

Figura 3. Sistema de archivo paralelo y distribuido

Por un lado, el acceso al distribuidor por parte del cliente se realiza nodo a nodo. Es decir, quien requiere el servicio accede a él directamente por medio de un nodo centralizado que contiene toda la información. El clásico sistema NFS. Por otro lado, en la solución paralela los datos se encuentran distribuidos a lo largo de los de almacenamiento. El servidor de Metadata es consultado frecuentemente por los clientes para obtener información de ubicación de los datos y posterior envío de mensajes de lectura y escritura a los servidores de almacenamiento.

A su vez, también es posible diferenciar ambas opciones a través de características tecnológicas:

Característica	Paralelo	Distribuido
Sistema de archivo Cluster	✔	✘
Tolerancia a fallos	✔	✘
Escalabilidad	✔	✘
Alta Disponibilidad	✔	✔
Alto Rendimiento	✔	✘
Open Source	✔	✔

Figura 4. Sistema de archivo paralelo y distribuido

3. Cluster

Básicamente este concepto hace referencia a un conjunto de computadoras interconectadas entre sí por medio de una red de alto rendimiento y que a la vista del usuario se ven como una única computadora.

3.1. Tipo de Cluster

Se suele hacer una distinción entre dos tipos, dependiendo del objetivo principal del mismo:

- Cluster de Alto Desempeño (HPC): Acelera un cálculo complejo realizando una partición de éste, de modo que cada nodo pueda ejecutar una parte. Así un cluster de 10 nodos podría resolver un sistema de ecuaciones en un décimo del tiempo que tardaría un único nodo[2].
- Cluster de Alta Disponibilidad (HA): Es un conjunto de dos o más servidores que se caracteriza por compartir el sistema de almacenamiento y porque se están constantemente monitoreando mutuamente. Si se produce un fallo del hardware o de los servicios de alguna de las máquinas que forman el cluster, el software de alta disponibilidad es capaz de volver a arrancar automáticamente los servicios que han

[2] Implementación y Operación de un Cluster HPC utilizando Laboratorios de Computadoras en Horarios de Inactividad, Ernesto Sale , Sebastián Rodríguez

fallado en cualquiera de los otros equipos del cluster. Y cuando el servidor que ha fallado se recupera, los servicios se migran de nuevo a la máquina original[3].

La arquitectura clásica de los cluster es denominada “Beowulf”[4], y respeta el siguiente diseño:

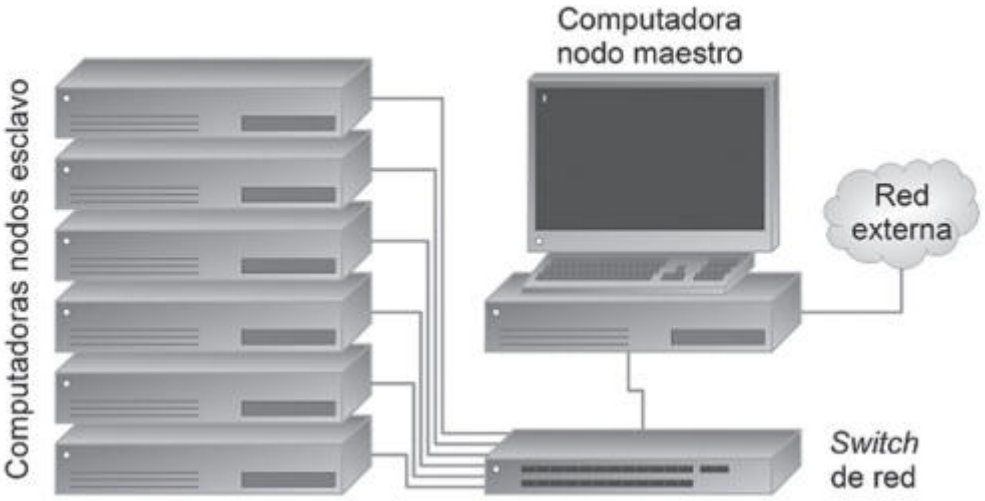

Figura 5. Cluster tipo Beowulf.

4. BeeGFS

Es un sistema de archivos paralelo POSIX (Portable Operating System Interface)[5], independiente del hardware, enfocado en el rendimiento y diseñado para facilitar el uso, la instalación y la gestión. Su código fuente es público y se ofrece en dos modos: por un lado, como desarrollador con soporte de la comunidad y por otro lado, como Enterprise, que cuenta con características y funcionalidades adicionales. Se encuentra diseñado para trabajar en una variedad de entornos, entre los que se destacan los orientados al rendimiento, como HPC, IA y Deep Learning, entre varios más.

BeeGFS distribuye de manera automática los datos que carga el usuario, entre distintos servidores. Esto permite escalar fácilmente en rendimiento y capacidad del cluster, simplemente añadiendo servidores a la infraestructura de acuerdo a las necesidades que se presenten.

[3] Clúster de Alta Disponibilidad, disponible en https://www.ecured.cu/Cl%C3%BAster_de_Alta_Disponibilidad
[4] Sterling, T., Becker, D. J., Savarese, D., Dorband, J. E., Ranawake, U. A., Packer, C. V., (1995). BEOWULF: A Parallel Workstation for Scientific Computation. Proceedings of the 24th International Conference on Parallel Processing.
[5] ¿Qué es POSIX?, disponible en https://blog.carreralinux.com.ar/2018/02/posix-conceptos-fundamentales/

Entre las principales características de BeeGFS, se destacan:

- Uso de tecnologías HPC: BeeGFS se basa en componentes multiproceso altamente eficientes y escalables. Los nodos dan servicio de conexiones de red RDMA InfiniBand, (Omni-Path), RoCE y TCP / IP, en simultáneo y alternan a conexiones redundantes en caso de fallas.
- Simple de usar: A diferencia de otros sistemas de archivos paralelo, como Lustre, BeeGFS no requiere de parches del kernel. El cliente es un módulo de kernel sin parches y los componentes del servidor son demonios del espacio de usuario.
- Acceso altamente concurrente: BeeGFS fue diseñado específicamente para brindar robustez y rendimiento en situaciones de alta carga de operaciones de E/S.
- Distribución de datos y Metadatos: Respetando el fin más próximo de los sistemas de archivo paralelo, BeeGFS permite escalar linealmente la infraestructura a nivel de almacenamiento y metadatos. Divide tanto el contenido de un archivo en diversos nodos de almacenamiento, como así también el sistema de archivos de metadatos entre distintos servidores de metadatos.
- Cliente y Servidor en cualquier máquina: No se requiere paquetes especiales de distribuciones para realizar funciones específicas. A través de las diferentes particiones podemos darle variadas funcionalidades a cualquier nodo dentro de la infraestructura de BeeGFS, es decir, establecer que una misma máquina actúe como cliente y servidor en simultáneo.

4.1. **¿Por qué BeeGFS?**

Una de las razones principales en esta elección es el hecho de que una de sus características que se destaca es que sus desarrollos están orientados a aplicaciones del tipo Machine y Deep Learning, entre otras tecnologías de punta como HPC.

Es de las distribuciones que mejor documentación en calidad y cantidad se puede encontrar. Tiene toda una sección de wiki orientada a explicar cada detalle, opción, configuración y funcionamiento del mismo. Esto es vital para realizar un laboratorio en donde se busca monitorear y analizar comportamientos de aplicaciones, y así tener clara la información de parámetros a modificar.

Otro punto a resaltar de BeeGFS es la facilidad en su implementación. Mientras en otras distribuciones es necesario aplicar parches en kernel y configuraciones adicionales para el correcto funcionamiento de los servicios que los compone; en BeeGFS esto no es así. Gracias a la calidad de la documentación y facilidades propias del software, la instalación y

configuración típica de la arquitectura no debe llevar un tiempo similar a otros sistemas de archivos. Con los nodos previamente instalados y configurados con el software de virtualización de preferencia, simplemente involucra la corrección de las dependencia de paquetes, descarga e instalación de dichos paquetes y posterior configuración de los nodos con unas pocas líneas en consola. Con esto, ya se estará en el mundo de la descentralización de los sistemas de archivos.

Respecto al rendimiento no se han encontrado artículos actuales que comparen a BeeGFS frente a la competencia. Las investigaciones realizadas que se circulan en la web demuestran que este es similar a otras soluciones, aunque muchas actualizaciones salieron desde esos análisis[6].

5. Collectl

Es una herramienta *All In One* de monitoreo de rendimiento. A través de la misma podemos realizar una supervisión y recopilación de datos en tiempo real de un amplio conjunto de subsistemas, como lo son: buddyinfo, cpu, disco, infiniband, memoria, red, nfs, procesos, quadrics, slabs, sockets y tcp. Esto es útil para verificar el estado general de un sistema, determinar qué estaba haciendo el mismo en un punto determinado o simplemente para realizar múltiples evaluaciones necesarias.

Su instalación es muy sencilla y utiliza recursos mínimos dentro del equipo para su ejecución. En el presente trabajo será utilizado para tomar información de disco y memoria en tiempo real de ejecución de la aplicación, grabando dicha información en archivos .txt para su posterior recopilación y análisis.

La ejecución básica de collectl luego de su instalación se realiza mediante la siguiente línea en consola:

```
collectl
```

Ésta muestra información general de varios subsistemas, aunque seteando parámetros seguidos de la línea mostrada anteriormente podemos indicar al programa que tome distinta información sobre un subsistema en particular. En el presente caso se utilizaron los seteos para memoria y disco, con los siguientes comandos:

Para la obtencion de informacion de disco:

```
collectl -sCDN -oT
```

[6] Comparison between different online storage systems, D.Autiero, D.Caiulo, S.Galymov, J.Marteau, E.Pennacchio E.Bechetoille, B.Carlus, C.Girerd, H.Mathez

Para la obtencion de informacion de memoria:

```
collectl -sjmf -oT
```

El seteo -oT hace referencia a la inclusión de la hora en los registros de collectl, sumamente útil para el posterior análisis de la información recopilada en los nodos.

6. Oracle VM VirtualBox

Es un software open source desarrollado por Oracle que permite, a través de sus herramientas, crear máquinas virtuales con sistemas operativos distintos y corriendo bajo un mismo host. A su vez, también posibilita la conectividad y comunicación bidireccional con el nodo que hace de host de las máquinas virtuales (VM).

6.1. Características

Entre sus características principales se destacan:

- Implementaciones fáciles y rápidas: Gracias a su interfaz sencilla y sus avances en la virtualización, con la simple carga de un archivo de imagen de un sistema operativo, el software automáticamente preconfigura características recomendadas para dicho sistema.
- Reducción de costo: Al tratarse de un software open source, no tenemos costos en su utilización.
- Escalabilidad: Nos permite adaptar hasta 32 máquinas virtuales con diferentes sistemas operativos y características.

6.2. ¿Por qué Oracle VM VirtualBox?

Se suele utilizar Oracle no sólo por los costos nulos de su uso no comercial, sino por la simplicidad de su interfaz y la facilidad de su operatoria. Es un software muy intuitivo tanto para usuarios inexpertos que dan sus primeros pasos en el mundo de la virtualización, como para quienes tienen años de experiencia y necesitan una solución rápida y práctica para la realización de escenarios experimentales que no requieren explícitamente de un software de virtualización más profesional como Proxmox o Hyper-V, entre otros.

7. Dataset

Es una colección de piezas de datos que una computadora maneja como una unidad, con fines analíticos y predictivos. Los datos deben ser comprensibles y uniformes para ser

entendidos por la máquina, que los usará para entrenar un algoritmo con el simple objetivo de encontrar patrones predecibles de dicho conjunto de datos.

Existen distintas webs para obtener datasets gratuitos, donde muchos de ellos vienen preparados ya con su aplicación y que son ideales para realizar pruebas de comportamientos en laboratorios. En este trabajo en particular se utilizó la web de Kaggle.

Comúnmente estos tienen una estructura "normalizada" y compuesta por imágenes clasificadas en distintos tipos: las de entrenamiento y las de pruebas, entre ellas completando un dataset. Un 70-80% del dataset total es utilizado para set de entrenamiento mientras que el 30-20% restante es para set de pruebas. A su vez, estos también son utilizados para evaluar los modelos en etapas intermedias de ejecución de la aplicación. Por ejemplo, si se calcula sobre el set de entrenamiento, su finalidad es entender qué tan bien está aprendiendo el modelo. Por el contrario, si se hace sobre el set de pruebas, obtendremos una idea de que tan bien el modelo se está volviendo capaz de generalizar.

Si se traza sobre un plano las curvas de errores a los momentos de evaluaciones del modelo en ambos set se podrá obtener una idea de que tan bien ajustado está nuestro modelo. Cuanto menor sea la distancia entre curvas, mejor ajustado se encontrará.

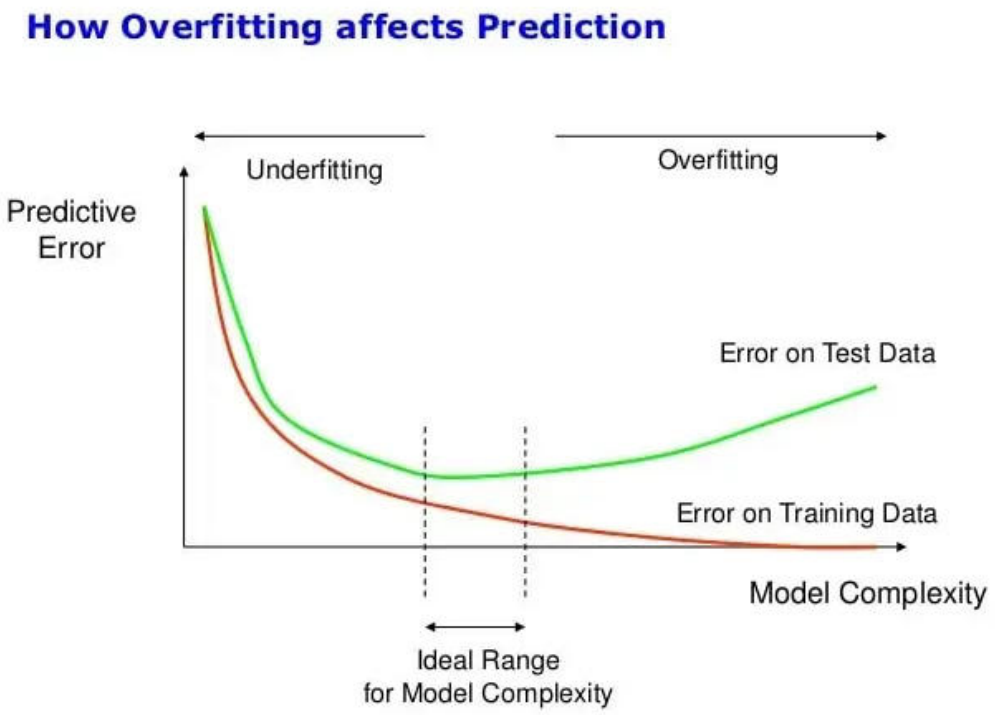

Figura 6. Curva de aprendizaje del modelo.

8. Machine Learning

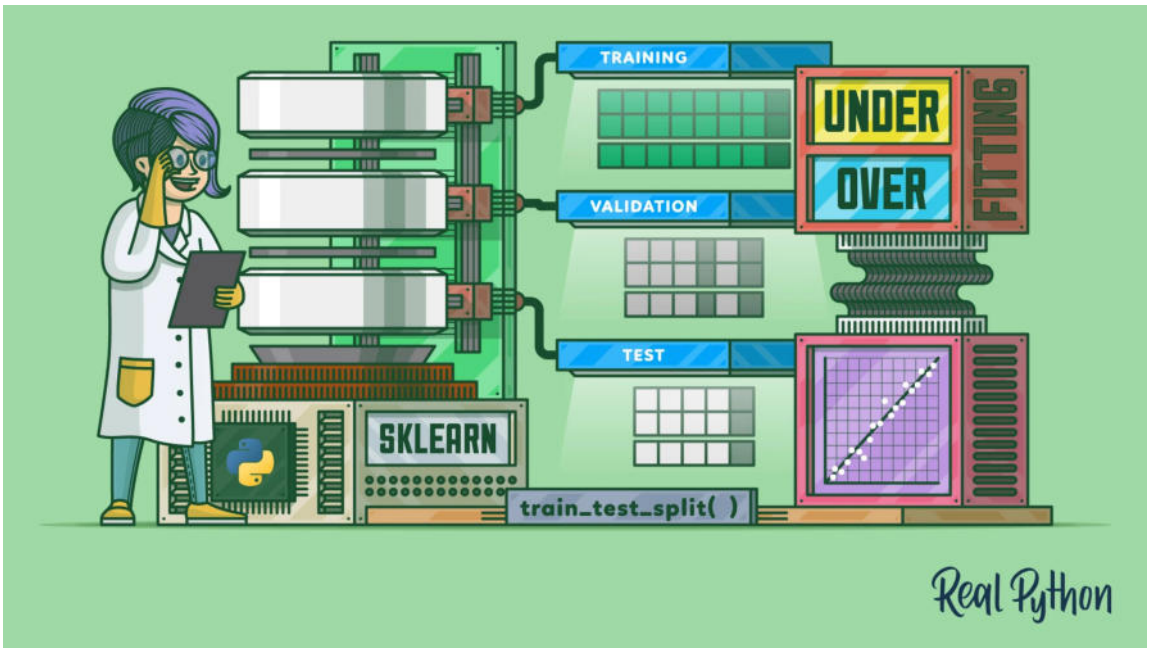

Figura 7. Machine Leaning en acción

A partir de la lectura de la bibliografía actual, se la define como el campo que se ocupa de las cuestiones de cómo construir programas de computadora que mejoran automáticamente con la experiencia. Para contribuir a esta definición, también es posible describirla como una evolución en la rama de la tecnología del desarrollo de software que permite diseñar algoritmos capaces de simular la inteligencia humana. Esto lo realiza mediante un proceso de aprendizaje y entrenamiento de modelos predictivos.

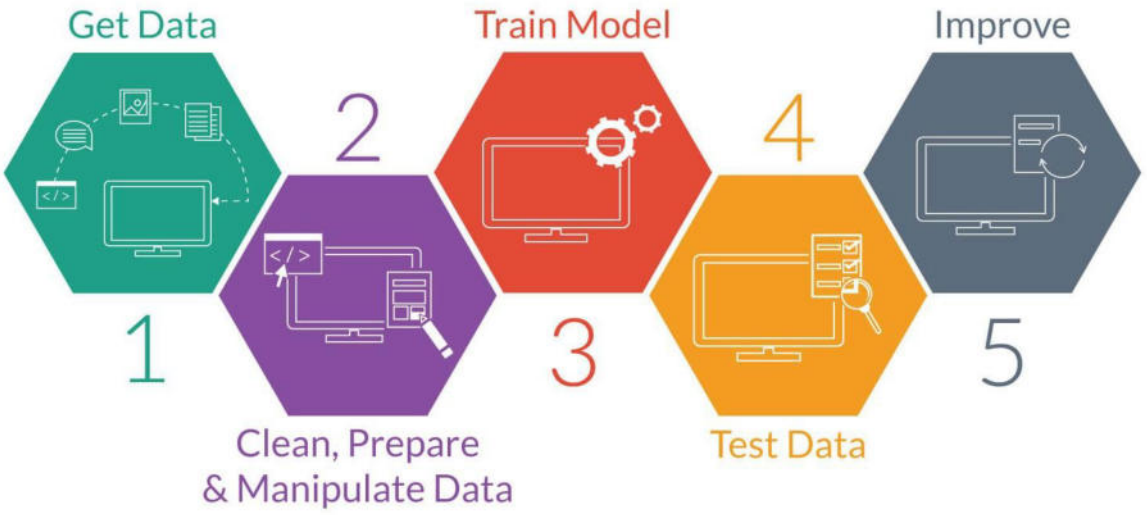

Figura 8. Procesos en Machine Learning

Se suele descomponer un proceso de Machine Learning en cinco etapas principales:

- Toma de datos: Acciones que tienen que ver propiamente con la toma de información necesaria para el desarrollo de los modelos. Las fuentes de estas dependen de la industria en la que se trabaje. Algunos ejemplos son imágenes, audios o sensores.
- Limpieza, Preparación y Manipulación de datos: Involucra las tareas de ordenamiento de información para un mejor tratamiento y optimización en el entrenamiento.
- Entrenamiento del modelo: Es la etapa donde se utilizan los datos de entrenamiento del set de datos para preparar el modelo
- Pruebas: Se refiere al proceso en donde se realizan las pruebas del modelo para verificar cuán ajustado se encuentra.
- Mejoramiento: Hace referencia a la utilización de la retroalimentación de la etapa de pruebas, para la optimización del modelo.

8.1. **Categorías**

Es factible dividir ML en dos grandes aspectos:

- Supervisado: En este tipo de entrenamiento se suele utilizar lo que se denomina "etiquetado" de los datos. Por ejemplo, si se quiere identificar frutas en la etapa de entrenamiento se le agrega a las imágenes una etiqueta con el tipo de fruta que es (manzana, banana, naranja). Esto permite tener datos de alta calidad para el aprendizaje, ya que anteriormente fueron clasificados por humanos. Una vez terminada esta fase, ya se le envía al modelo nuevas imágenes y será capaz de identificar el tipo de fruta que le estamos mostrando.
- No supervisado: Contrario al anterior, aquí no se utilizan etiquetas. Normalmente se suele usar una técnica denominada agrupamiento. Se basa en dividir en grupos a los datos de acuerdo a ciertas características que comparten.

8.2. **Estados**

Como se mencionó anteriormente, el modelo de una aplicación de Machine Learning permite evaluar su estado en distintos momentos mediante una curva de aprendizaje. Esto permite determinar qué tan ajustado se encuentra de acuerdo a las necesidades.[7]:

[7] Machine Learning Basics, I Goodfellow, Y Bengio, A Courville

- Desajuste: Ocurre cuando el modelo no logra obtener un valor de error suficientemente bajo para el entrenamiento.
- Sobreajuste: Sucede cuando el espacio entre el error de entrenamiento y el de pruebas es demasiado grande.

Además, los modelos tienen una característica adicional que los distingue: la capacidad.

- Una baja capacidad puede ocasionar un desajuste.
- Una alta capacidad puede ocasionar un sobreajuste.

8.3. **Fundamentos de Machine Learning**

Básicamente se fundamenta su existencia, importancia y continua expansión desde el hecho de la utilidad que le brinda a distintos sectores de la población: detección temprana de enfermedades[8], fraudes, marketing, optimización de recursos naturales y materiales, son solo una pequeña cantidad de funciones que esta tecnología ya se encuentra desplegando actualmente en la sociedad y generando beneficios constantes. Además, asiduamente leemos o escuchamos noticias de mala praxis, accidentes o fraudes ocasionados simplemente por conductas humanas maliciosas o no que llevan a ello y que sistemas inteligentes podrían evitar gracias a su imparcialidad, con las auditorías que esto respecta por su puesto.

8.4. **Ventajas**

A continuación se detallan algunos beneficios del uso de IA frente al ser humano:

- Retienen grandes volúmenes de información.
- A medida que avanza el tiempo, se vuelven más inteligentes.
- No está limitada a efectos de cansancio, distracciones o pérdida de memoria.

8.5. **Desventajas**

Entre los inconvenientes del uso de este tipo de tecnología se destacan los siguientes:

- Alto consumo en tiempo y recursos: Normalmente las aplicaciones de IA requieren de grandes equipos que permitan la operatividad el 100% debido al tiempo de entrenamiento de los modelos y ejecución de aplicaciones.

[8] Esto se observa en el trabajo de investigacion “Artificial intelligence–enabled public health surveillance—from local detection to global epidemic monitoring and control” disponible en https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7484813/

- Toma de datos: Para el aprendizaje, se requieren grandes volúmenes de datos, imparciales y de gran calidad. Dependiendo la industria de tratamiento, estos son complejos en su obtención.

8.6. Épocas

Hace referencia a la cantidad de pasadas que realiza el algoritmo de aprendizaje al set de datos de entrenamiento completo.

9. Darshan

Es un software diseñado para capturar de forma precisa el comportamiento de entrada/salida de una aplicación, incluyendo propiedades como patrones de acceso a archivos.

Se utilizará en el apartado de validación de datos para verificar que las aplicaciones de Machine Learning estén realizando efectivamente su trabajo sobre las unidades montadas en los clientes de los sistemas de archivos paralelos, más allá de que la herramienta Collectl muestre el incremento de uso de recursos en los servidores.

10. Arquitectura

En la figura 9 se visualiza la arquitectura típica de BeeGFS:

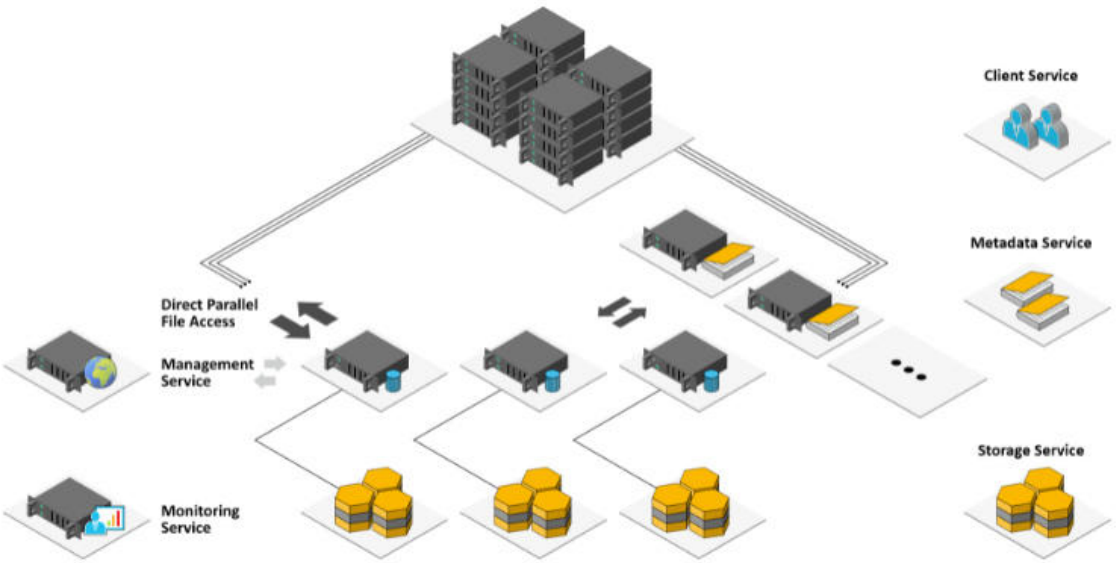

Figura 9. Infraestructura BeeGFS típica

A continuación se detallan los componentes principales de una arquitectura BeeGFS:

- Servicio de almacenamiento: Es el responsable de almacenar stripes (concepto que es explicado posteriormente) del contenido real de los archivos de los usuarios en los servidores de almacenamiento.
- Servicio de Metadata: Coordina la ubicación de los archivos y los stripes entre los distintos servidores de almacenamiento y brinda a los clientes cierta información de los archivos cuando es necesario.
- Servicio de administración: Sirve como registro y vigilancia para clientes y servidores. Su nodo se denomina servidor de administraciòn
- Servicio de supervisión: Recopila datos de rendimiento de los servidores y los envía a una base de datos. A partir de esto, el monitoreo en tiempo real y el análisis estadístico del sistema es posible con herramientas como Grafana .
- Cliente: Quien utiliza los servicios ofrecidos por BeeGFS.

Es habitual utilizar arquitecturas donde el servidor de Metadata actúa a su vez como servidor de administración y supervisión, a fin de ahorrar recursos a nivel de infraestructura y teniendo en cuenta que el uso de los mismos es limitado. En esta oportunidad se diferenciarán los servicios, cada uno en una máquina dedicada, para poder visualizar los resultados independientes.

Se utilizarán tres tipos de arquitectura para exponer los resultados luego de su respectiva evaluación, de acuerdo a los siguientes esquemas:

- Arquitectura 1: Compuesta por:
  - 1 servidor de administración
  - 1 servidor de metadata
  - 1 servidor de almacenamiento
  - 1 cliente
- Arquitectura 2: Compuesta por:
  - 1 servidor de administración
  - 1 servidor de metadata
  - 2 servidores de almacenamiento
  - 1 cliente
- Arquitectura 3: Compuesta por:
  - 1 servidor de administración
  - 2 servidores de metadata
  - 2 servidores de almacenamiento
  - 1 cliente

Las máquinas virtuales cuentan con la siguiente configuración de hardware:

- Servidor Metadata:

  - Disco: 60 GB
  - Memoria: 1 GB
  - Procesador: 2 CPU
  - Sistema Operativo: Ubuntu 20.4
- Servidor Administraciòn
  - Disco: 60 GB
  - Memoria: 1 GB
  - Procesador: 2 CPU
  - Sistema Operativo: Ubuntu 20.4
- Servidor almacenamiento:
  - Disco: 60 GB
  - Memoria: 1 GB
  - Procesador: 2 CPU
  - Sistema Operativo: Ubuntu 20.4
- Cliente:
  - Disco: 60 GB
  - Memoria: 2 GB
  - Procesador: 2 CPU
  - Sistema Operativo: Ubuntu 20.4

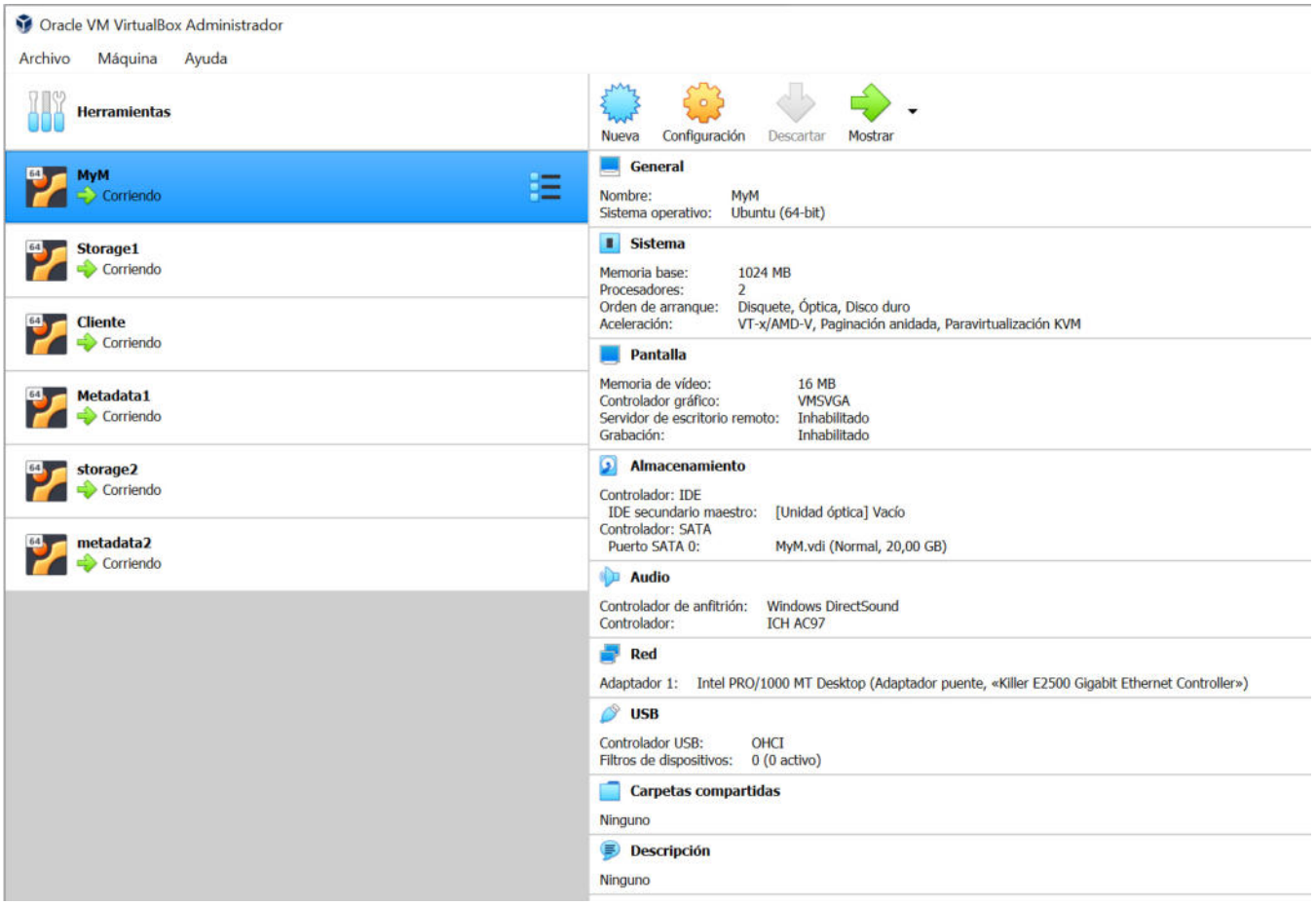

Figura 10. Máquinas en VirtualBox

La propuesta de estos tres tipos de arquitectura viene dada como una opción para visualizar, a través de los resultados obtenidos con *Collectl*, el aumento o disminución del rendimiento del sistema de archivos paralelo en la ejecución del mismo tipo de trabajo. Y además, la identificación del servicio más afectado por aplicaciones del tipo Machine Learning, a través del uso del escalamiento horizontal.

11. Benchmark

Benchmark es una técnica basada en una prueba realizada sobre un determinado sistema o componente, con el fin de medir el rendimiento de dicho sistema o componente. En el presente trabajo, se utilizarán dos aplicaciones de Machine Learning para medir el rendimiento:

- CIFAR-10: Es un dataset de 60000 imágenes en color de 32x32 en 10 clases, con 6000 por clase. Hay 50000 de entrenamiento y 10000 de prueba. Un peso aproximado de 160 MB.
- FRUITS 360: Es un dataset de imágenes que contiene frutas y vegetales de 100x100 en 131 clases. Está compuesto por 90483 en total, donde de ellas 67692 son de entrenamiento y 22688 de prueba. Un peso aproximado de 1,2 GB.

12. Parametrización BeeGFS

BeeGFS, así como gran parte de los sistemas archivos paralelos, permite definir opcionalmente una serie de parámetros, como por ejemplo número de storage targets, tamaño del stripe, caché o entre varios tipos de algoritmos de planificación dependiendo del entorno de producción y las necesidades particulares de la situación, junto a varias opciones más.

En este caso, se explicarán los que resultan relevantes para el trabajo y sobre los cuales se realizarán modificaciones para visualizar el cambio en los comportamientos, en caso de existir.

12.1. Stripe

Este parámetro hace referencia a los “fragmentos” entre los cuales el sistema divide un directorio o archivo y en el cual se define el tamaño del bloque de los mismos.

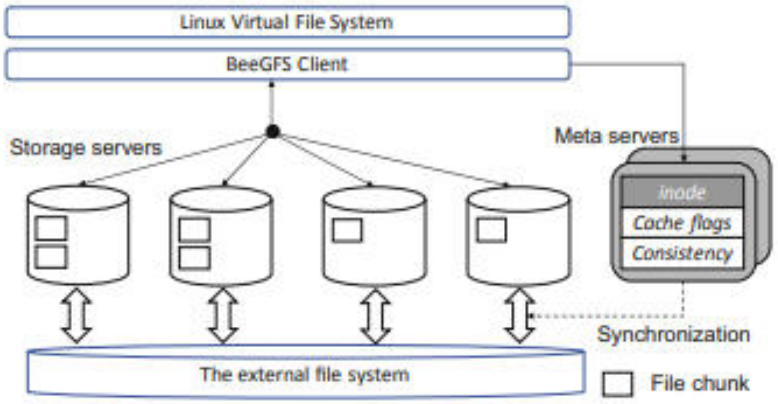

Figura 11. Stripe

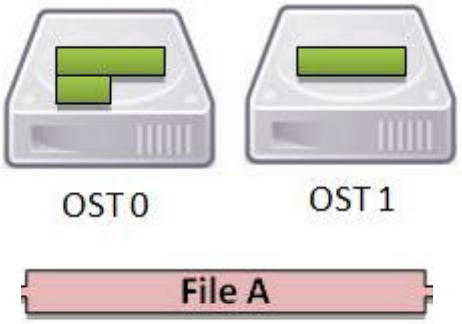

Figura 12. Stripe

En las figuras 11 cómo “File chunk” y 12 se puede visualizar los stripes correspondientes a los datos dentro de los nodos de almacenamiento. Cada uno de estos “fragmentos” en su conjunto forman el dato en sí.

Por defecto, BeeGFS define las siguientes características:

```
root@cliente1:/mnt/beegfs/fruits-360# beegfs-ctl --getentryinfo /mnt/beegfs
Entry type: directory
EntryID: root
Metadata node: meta1 [ID: 1]
Stripe pattern details:
+ Type: RAID0
+ Chunksize: 512K
+ Number of storage targets: desired: 4
+ Storage Pool: 1 (Default)
root@cliente1:/mnt/beegfs/fruits-360# _
```

Figura 13. Tamaño Stripe por defecto

En la figura 13, BeeGFS establece de manera predeterminada el uso de un stripe de 512K, denominado Chunksize, dividido a lo largo de 4 storage targets.

La modificación del mismo se realiza con el siguiente comando:

```
>> beegfs-ctl --setpattern --chunksize=1M /mnt/beegfs
```

12.2. Storage Targets

Este parámetro permite definir a través de cuantos Storage Targets se desea dividir el directorio que se esté estableciendo.

La modificación del mismo se efectúa con el siguiente comando:

```
beegfs-ctl --setpattern --numtargets=4 /mnt/beegfs
```

12.3. Algoritmo de planificación

Este parámetro permite seleccionar el tipo de algoritmo a utilizar por el sistema para la selección de ubicación de los datos en los storage, mediante la modificación del campo tuneTargetChooser en el archivo /etc/beegfs/beegfs-meta.conf.

12.3.1. Randomized

Es el algoritmo que por defecto viene definido como activo para la planificación del ordenamiento de archivos dentro de BeeGFS, y el más recomendado para ambientes de producción. Esto es debido a la aleatoriedad con la que varios usuarios constantemente crean archivos de distinto tipo y tamaño dentro del servidor. Como su nombre lo indica, con él, Metadata selecciona aleatoriamente los storage targets cuando se crea un nuevo archivo.

12.3.2. Roundrobin

Escoge los storage targets de manera determinista por turnos y ordenadamente.. Es recomendable principalmente para realizar pruebas de benchmark de archivos de gran volumen.

12.3.3. Randomrobin

Se comporta de igual forma que el algoritmo Roundrobin, pero el orden de selección es aleatorio.

Es importante notar que la modificación de los parámetros de stripe y storage targets tendrá validez únicamente sobre los nuevos archivos y no sobre los que ya se encuentran almacenados.

12.4. Caché

En informática, hace referencia a la utilización de un espacio para el almacenamiento de datos de forma temporal. Es una técnica comúnmente utilizada para agilizar procesos o reducir exigencias hacia servidores, disminuyendo operaciones de I/O, entre otras funcionalidades.

BeeGFS permite customizar dos técnicas de caché en el servicio del cliente[9]:

- Buffered: Opción seteada por defecto. Utiliza un pequeño grupo de búferes estáticos para lectura y escritura. Como máximo almacena unos cientos de kilobytes de un archivo.
- Native: Es una alternativa opcional a Buffered. Aún se encuentra en estado de experimentación por lo que no se recomienda su utilización en entornos de producción que se requiere de alta disponibilidad. Permite el almacenamiento en memoria de varios gigabytes, dependiendo de la capacidad de la RAM del cliente.

13. Resultados

En este apartado se busca reflejar los resultados obtenidos luego de las ejecuciones de las aplicaciones mencionadas en el apartado de Benchmark .

Por un lado, se analiza en disco las operaciones de lectura y escritura junto al tamaño de los paquetes; mientras que por el de la memoria se verá la capacidad libre vs usada. Los resultados obtenidos en cada nodo están sincronizados en tiempo de ejecución entre ellos, por medio de la utilización de la herramienta *tmux* y la división de paneles, junto al parámetro de horario de la herramienta *Collectl*. A través de los siguientes gráficos se puede visualizar lo que sucede a tiempos sincrónicos.

A continuación se detallan las métricas obtenidas para las distintas arquitecturas propuestas:

13.1. Resultados con CIFAR-10

Debido a que se trata de un set de datos de muy bajo volumen, no resulta interesante ni posible reflejar mediante gráficos el comportamiento de esta aplicación. Esto responde a que la exigencia hacia los servidores es tan ínfima que no se logra diferenciar el uso de recursos por parte del sistema operativo y por el software en cuestión.

[9] Client Side Caching Modes, disponible en https://doc.beegfs.io/latest/advanced_topics/client_caching.html

13.2. Resultados con FRUITS 360

De los datos recopilados de las ejecuciones correspondientes, se destacan los siguientes tópicos:

- Tiempo de ejecución arquitectura 1: 2 minutos y 11 segundos aproximadamente.
- Tiempo de ejecución arquitectura 2: 1 minuto y 59 segundos aproximadamente.
- Tiempo de ejecución arquitectura 3: 2 minutos y 5 segundos aproximadamente.

13.2.1. Resultados metadata

13.2.1.1. Arquitectura 1

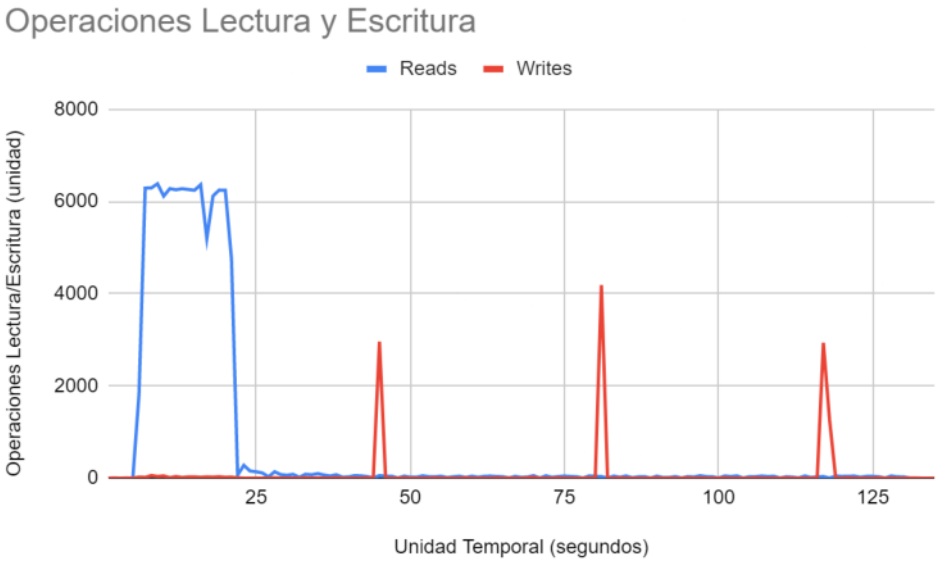

Figura 14. Operaciones lectura y escritura metadata arq. 1

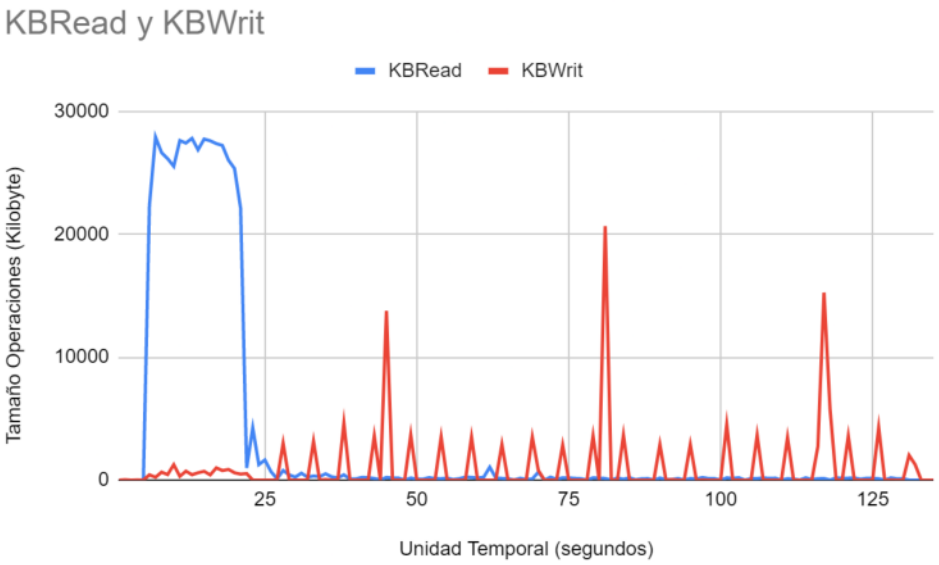

Figura 15. Tamaño operaciones lectura y escritura metadata arq. 1

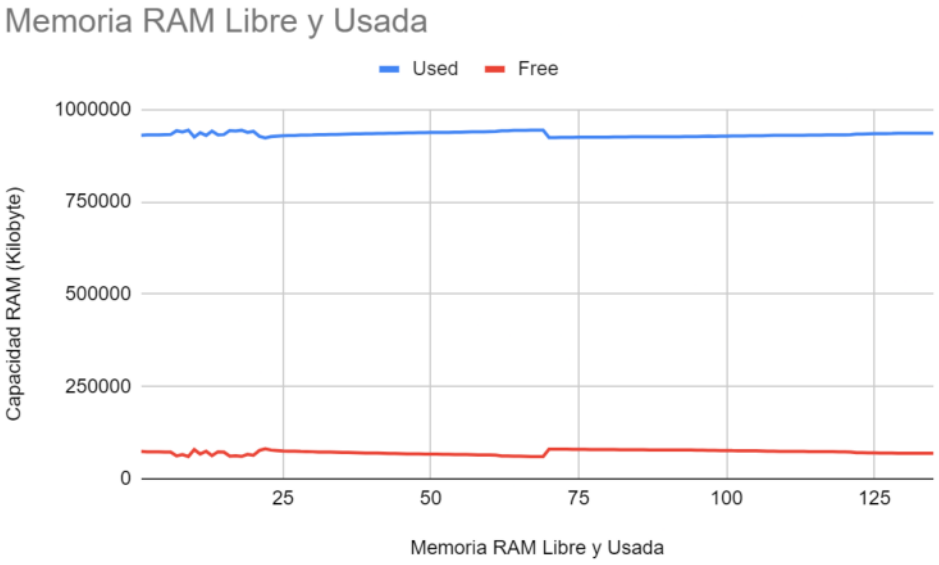

Figura 16. Capacidad Memoria RAM libre vs usada metadata arq. 1

13.2.1.2. Arquitectura 2

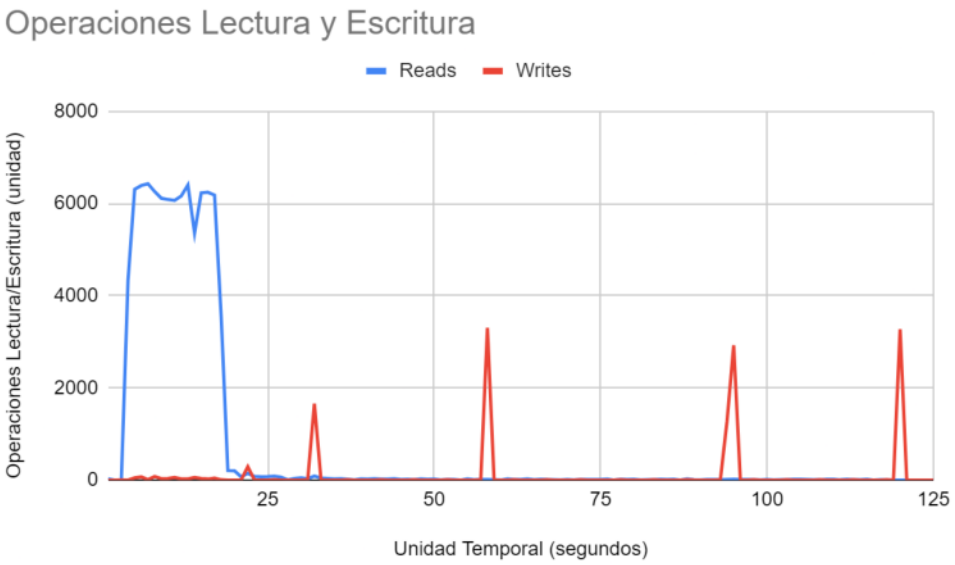

Figura 17. Operaciones lectura y escritura metadata arq. 2

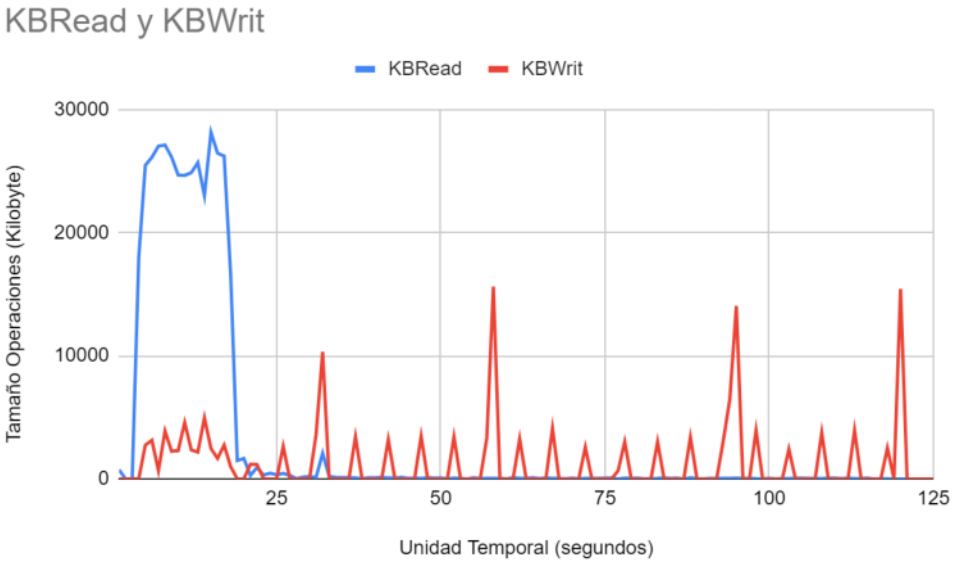

Figura 18. Tamaño operaciones lectura y escritura metadata arq. 2

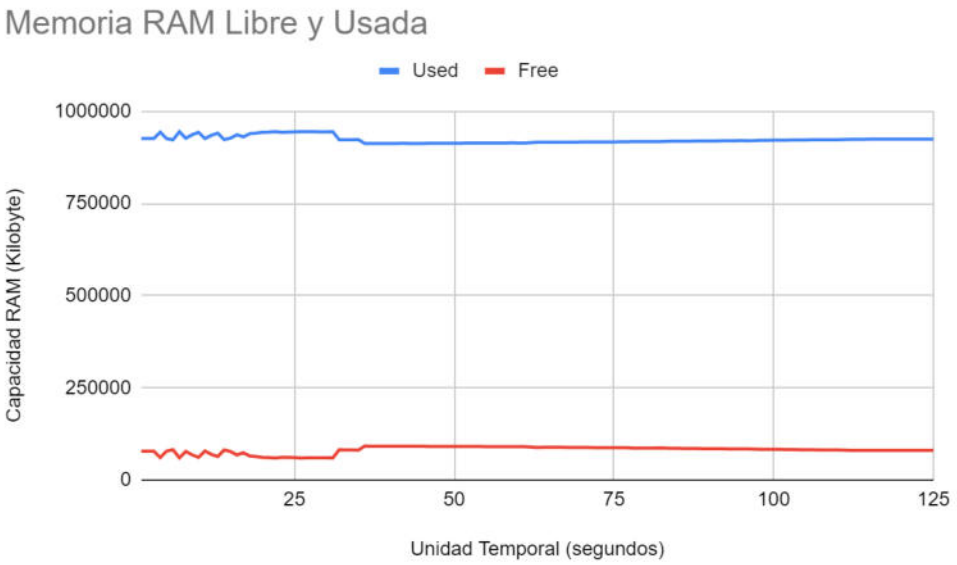

Figura 19. Capacidad Memoria RAM libre vs usada metadata arq. 2

13.2.1.3. Arquitectura 3

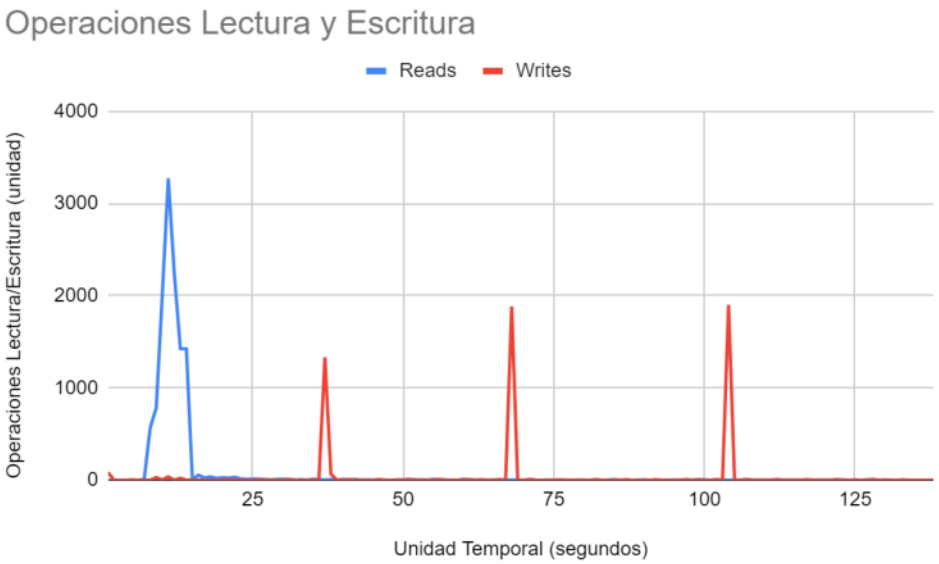

Figura 20. Operaciones lectura y escritura metadata 1 arq. 3

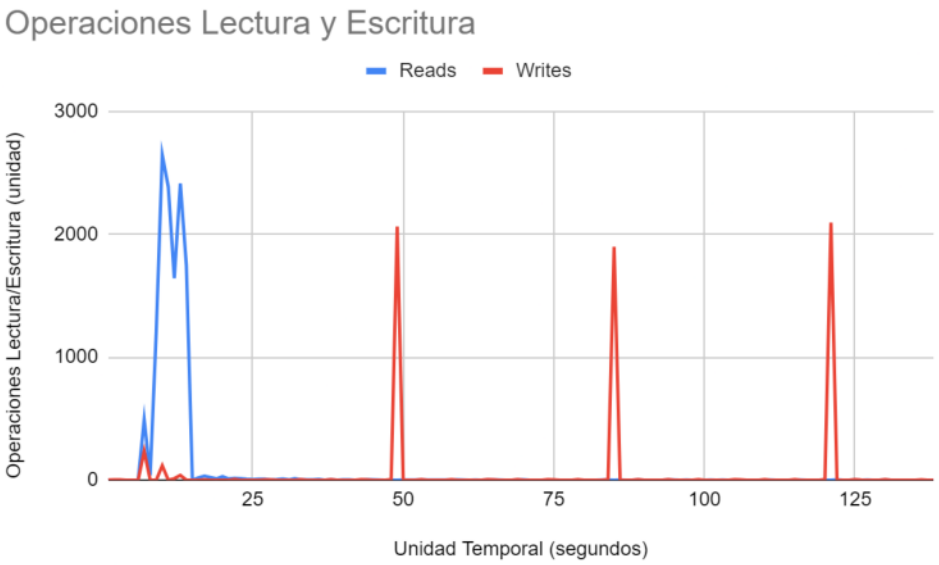

Figura 21. Operaciones lectura y escritura metadata 2 arq. 3

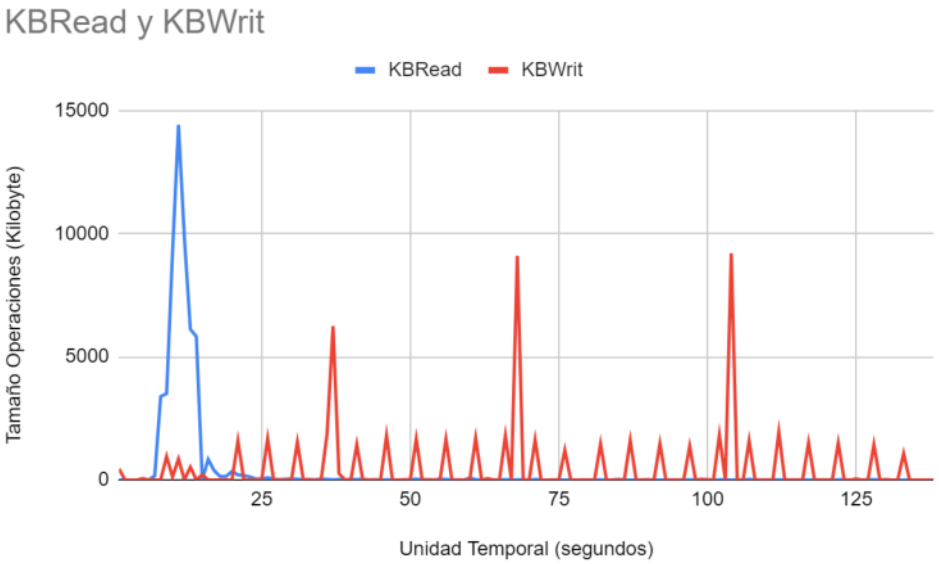

Figura 22. Tamaño operaciones lectura y escritura metadata 1 arq. 3

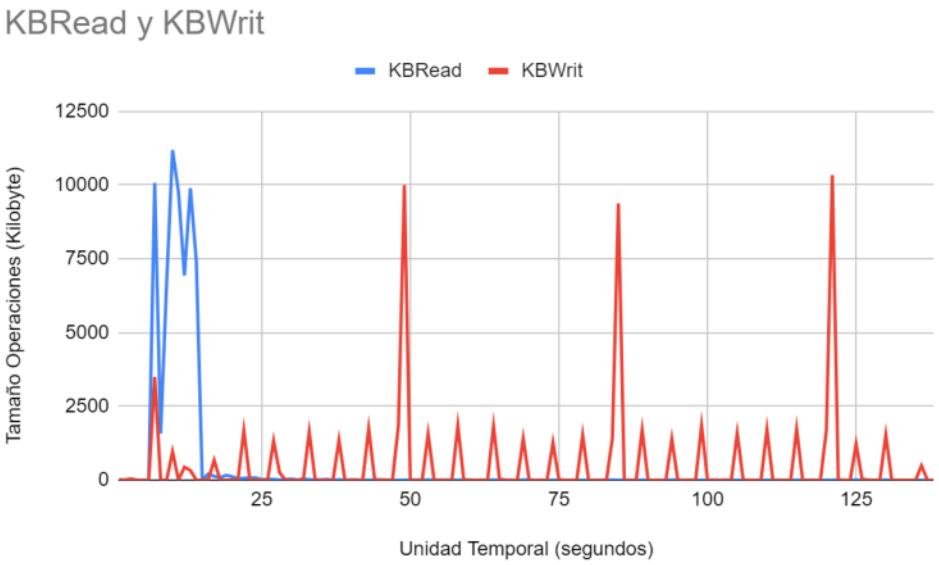

Figura 23. Tamaño operaciones lectura y escritura metadata 2 arq. 3

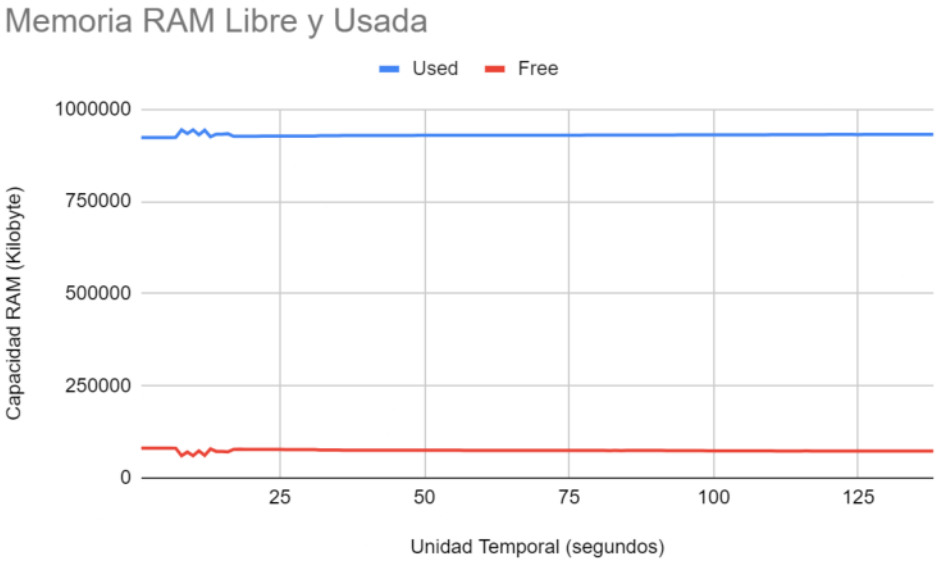

Figura 24. Capacidad Memoria RAM libre vs usada metadata 1 arq. 3

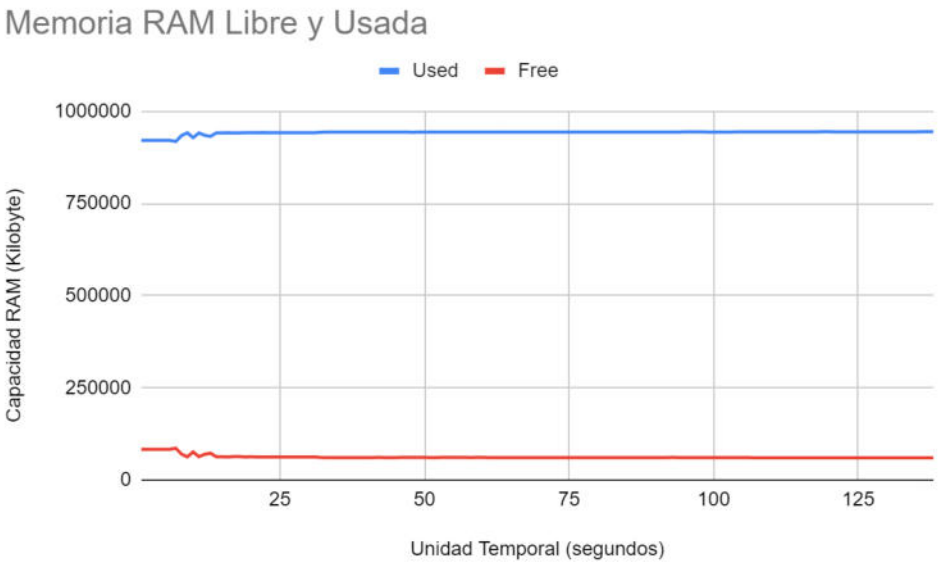

Figura 25. Capacidad Memoria RAM libre vs usada metadata 2 arq. 3

13.2.2. Resultados storage

13.2.2.1. Arquitectura 1

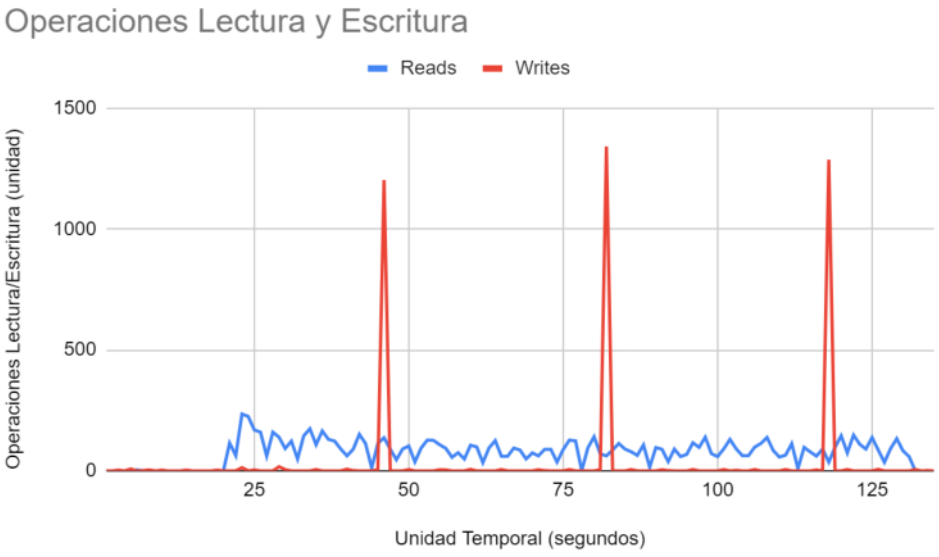

Figura 26. Operaciones lectura y escritura storage arq. 1

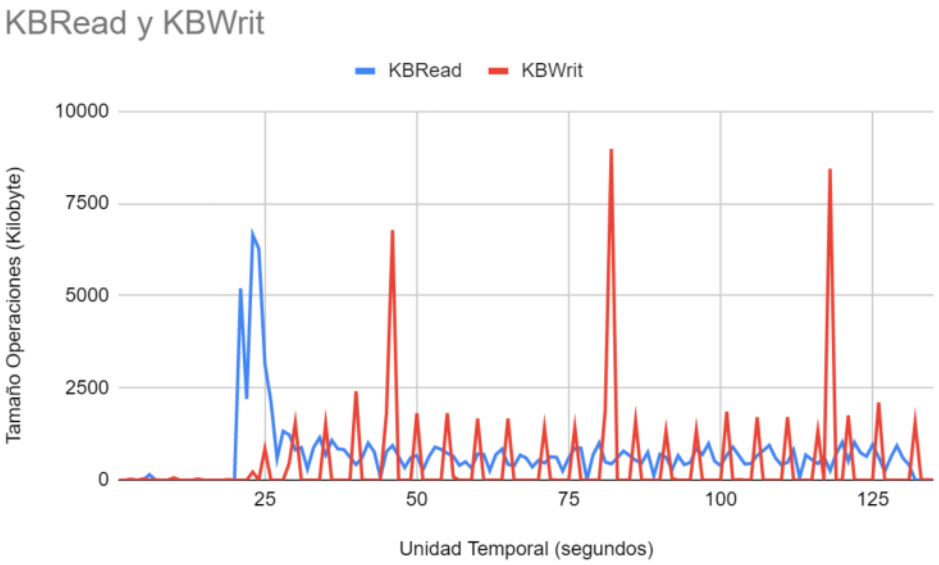

Figura 27. Tamaño operaciones lectura y escritura storage arq. 1

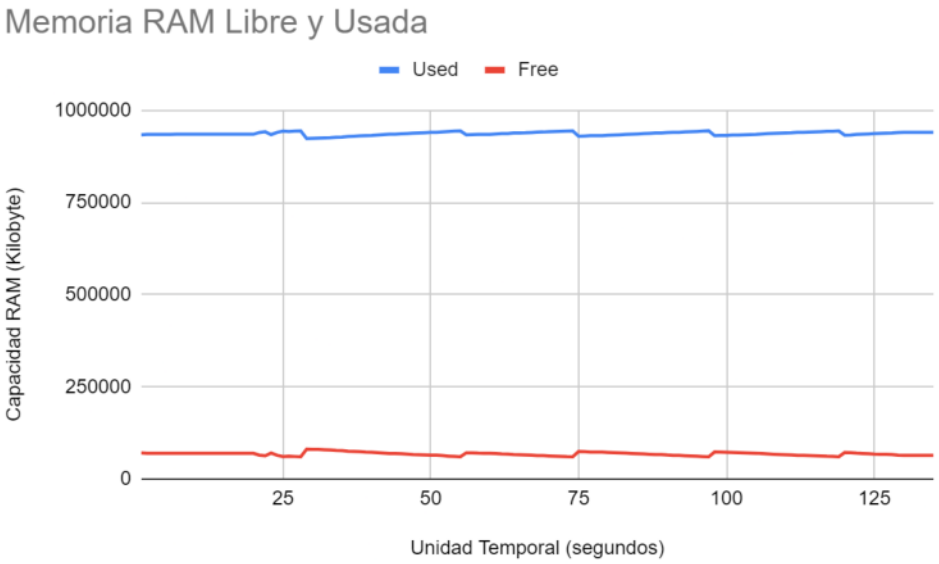

Figura 28. Capacidad Memoria RAM libre vs usada storage arq. 1

13.2.2.2. Arquitectura 2

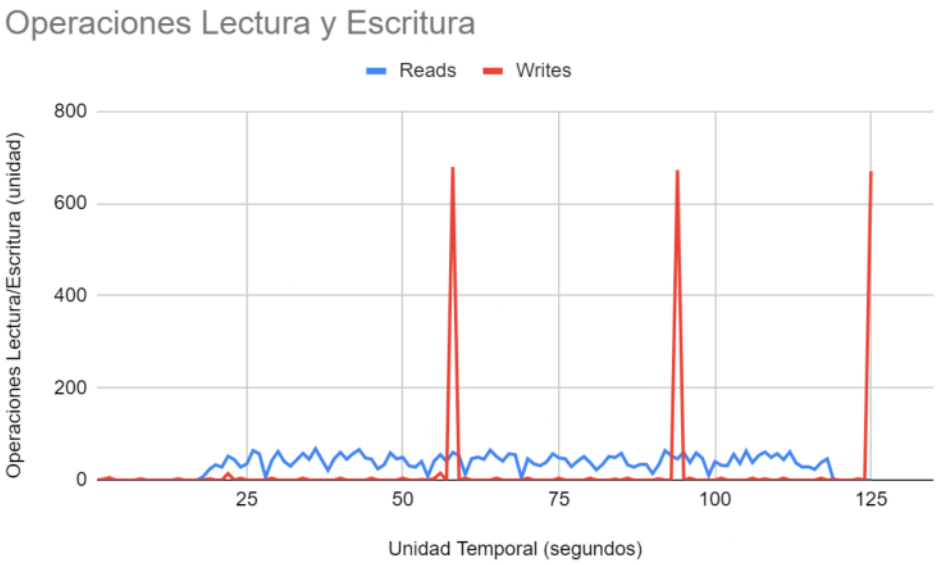

Figura 29. Operaciones lectura y escritura storage 1 arq. 2

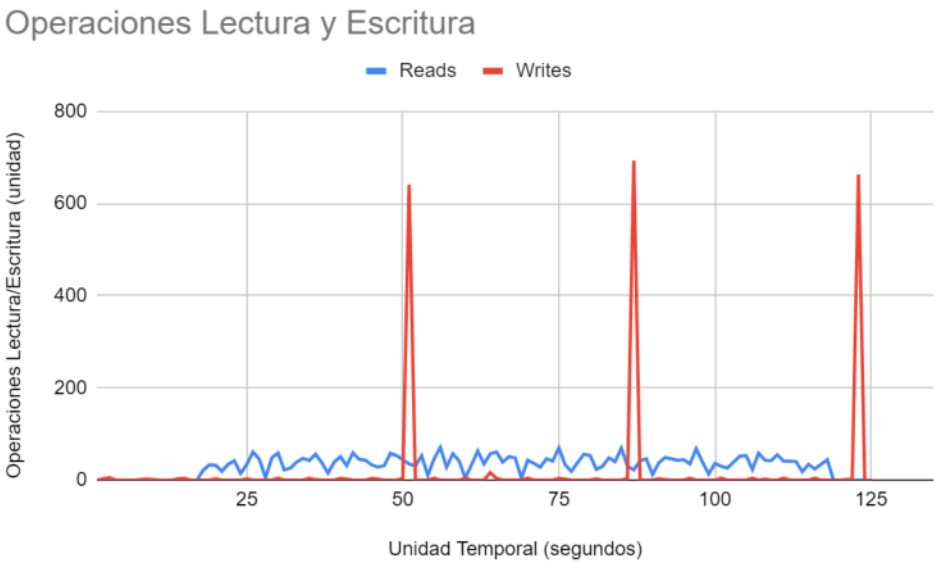

Figura 30. Operaciones lectura y escritura storage 2 arq. 2

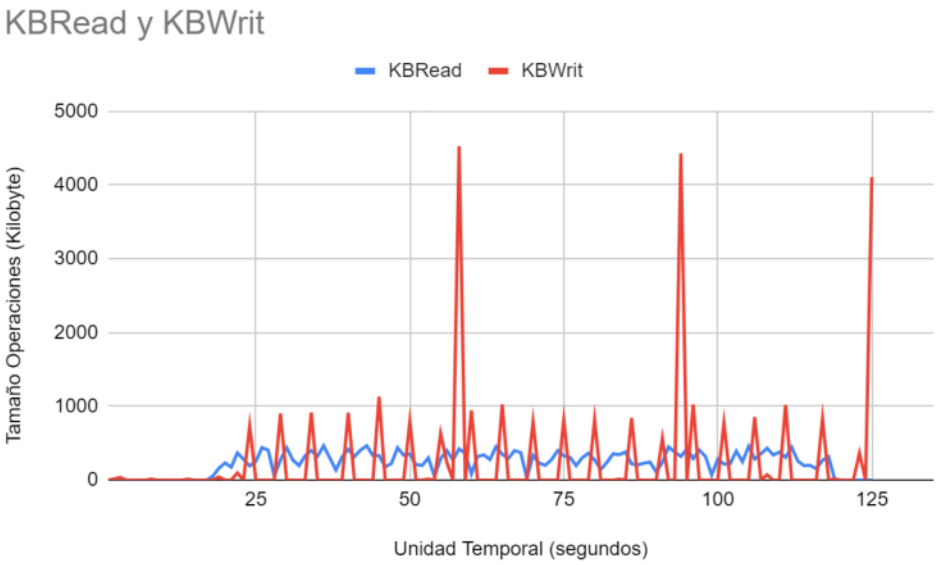

Figura 31. Tamaño operaciones lectura y escritura storage 1 arq. 2

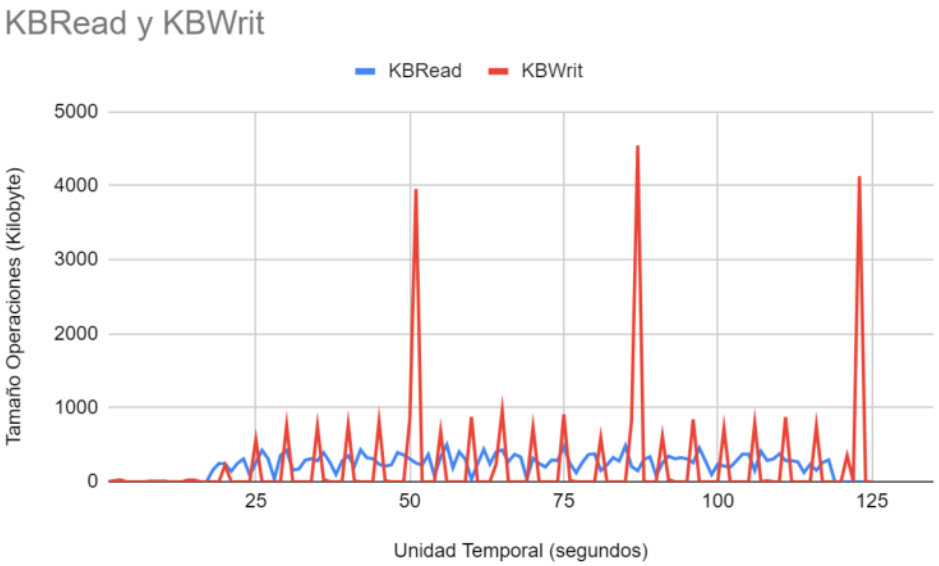

Figura 32. Tamaño operaciones lectura y escritura storage 2 arq. 2

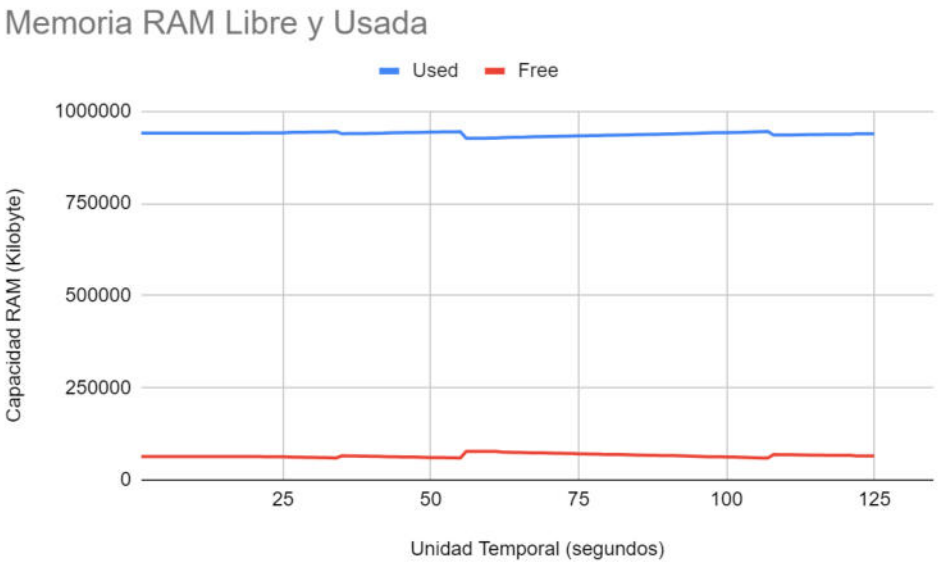

Figura 33. Capacidad Memoria RAM libre vs usada storage 1 arq. 2

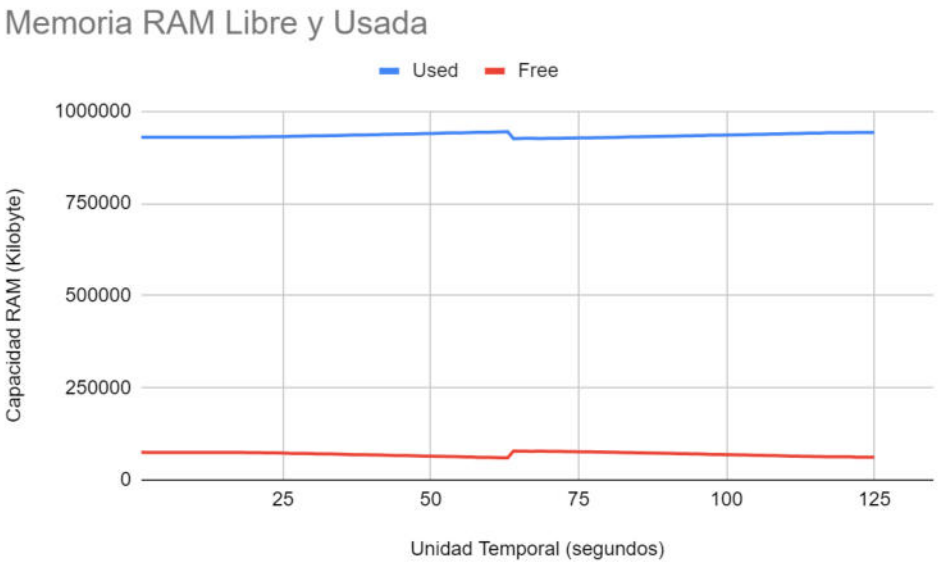

Figura 34. Capacidad Memoria RAM libre vs usada storage 2 arq. 2

13.2.2.3. Arquitectura 3

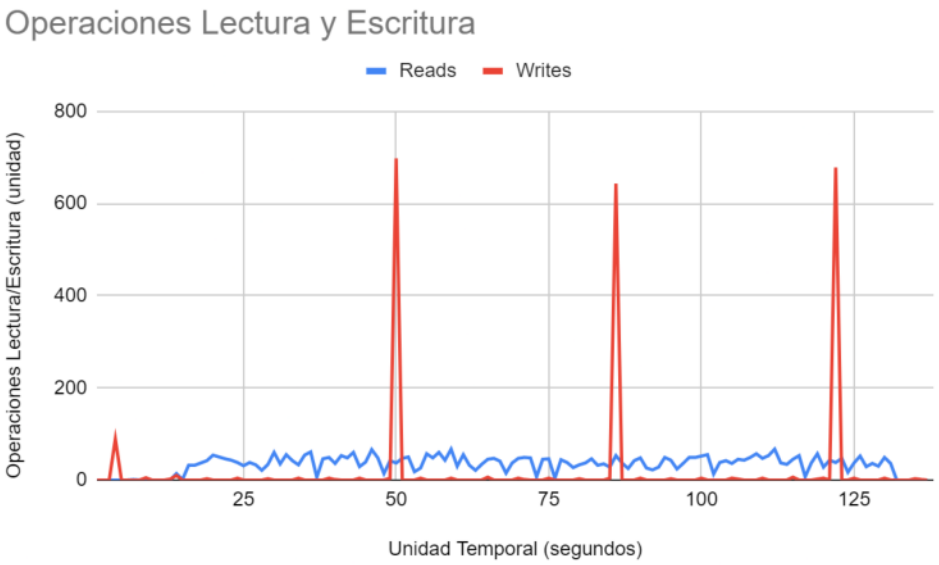

Figura 35. Operaciones lectura y escritura storage 1 arq. 3

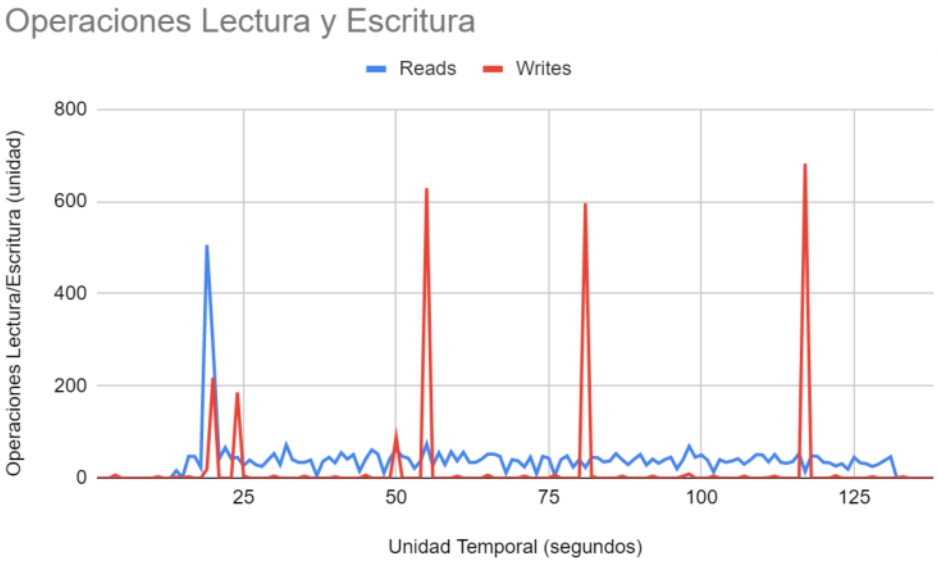

Figura 36. Operaciones lectura y escritura storage 1 arq. 3

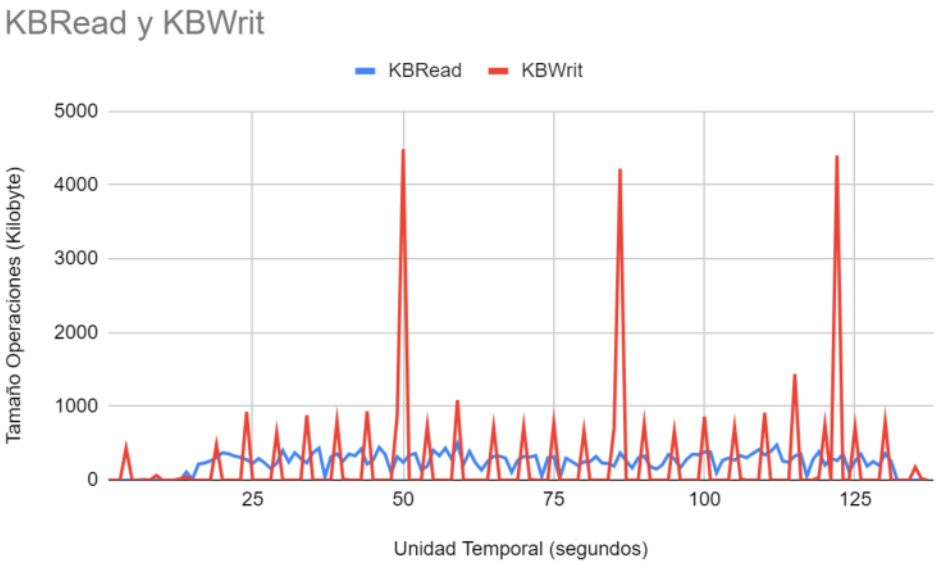

Figura 37. Tamaño operaciones lectura y escritura storage 1 arq. 3

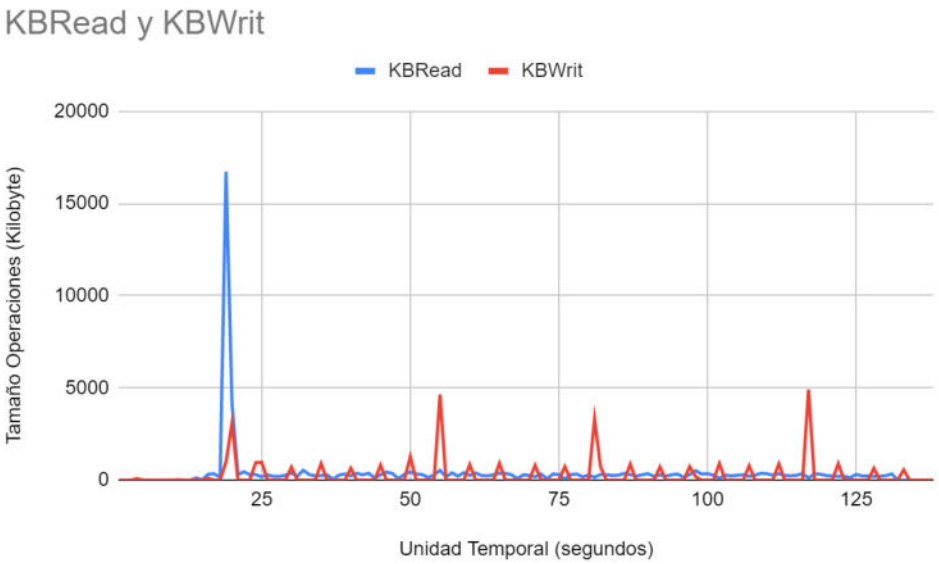

Figura 38. Tamaño operaciones lectura y escritura storage 2 arq. 3

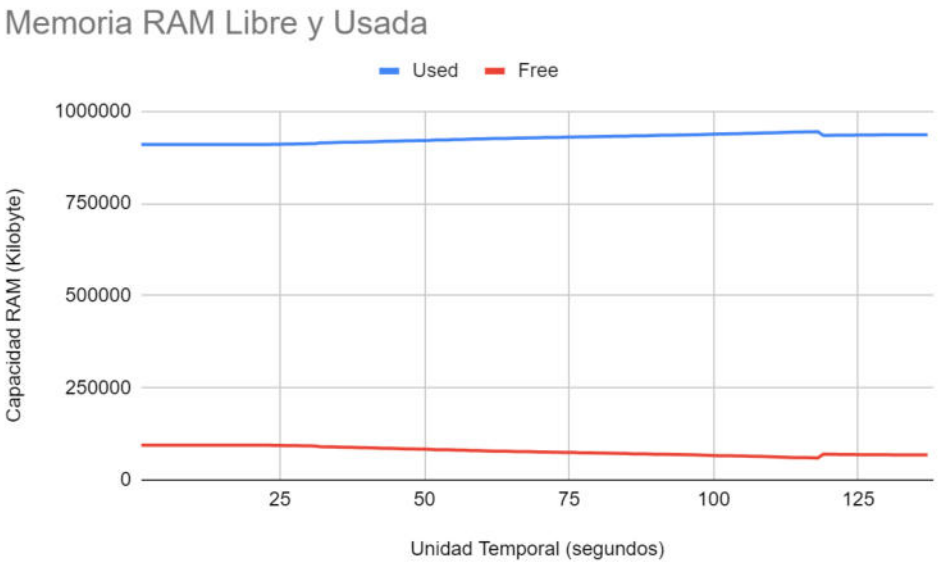

Figura 39. Capacidad Memoria RAM libre vs usada storage 1 arq. 3

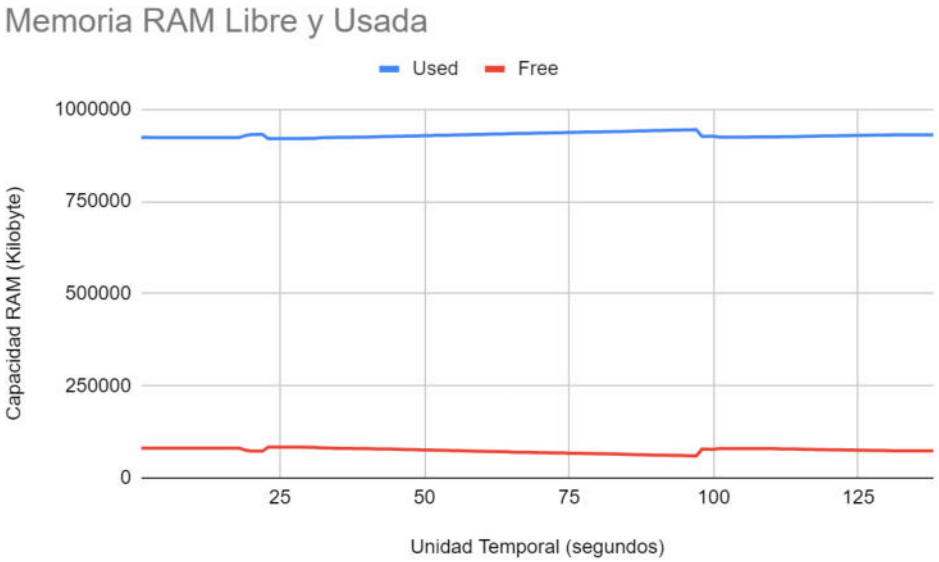

Figura 40. Capacidad Memoria RAM libre vs usada storage 2 arq. 3

13.2.3. Resultados cliente

13.2.3.1. Arquitectura 1

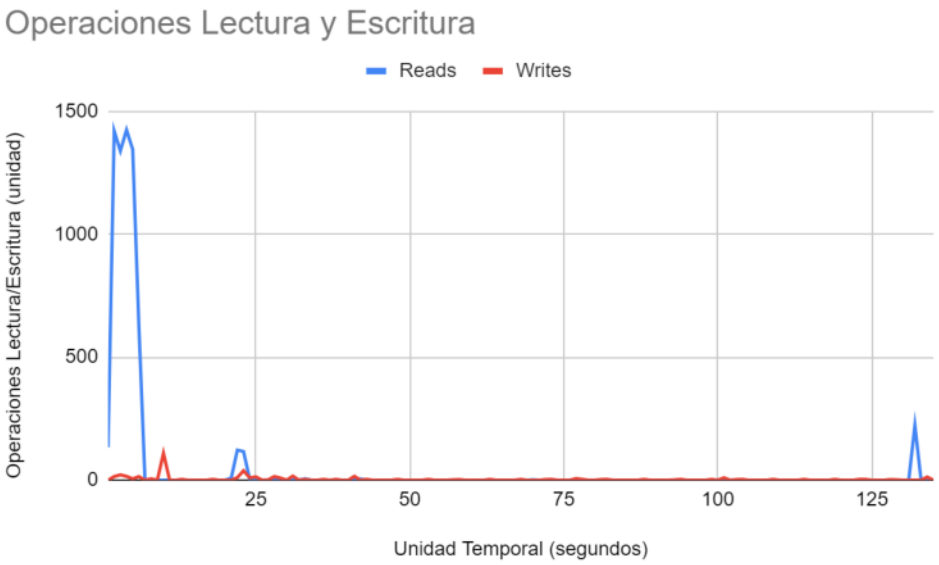

Figura 41. Operaciones lectura y escritura cliente arq. 1

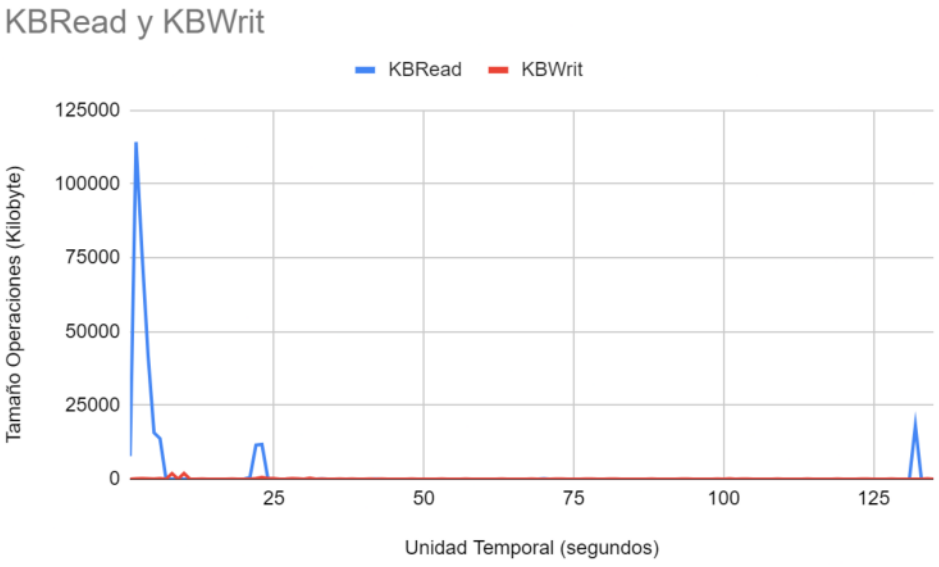

Figura 42. Tamaño operaciones lectura y escritura cliente arq. 1

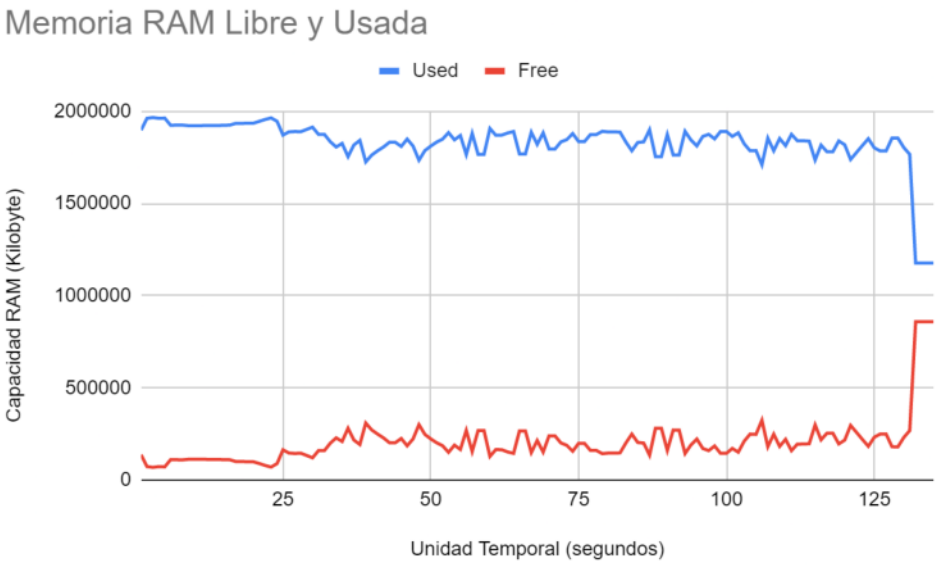

Figura 43. Capacidad Memoria RAM libre vs usada cliente arq. 1

13.2.3.2. Arquitectura 2

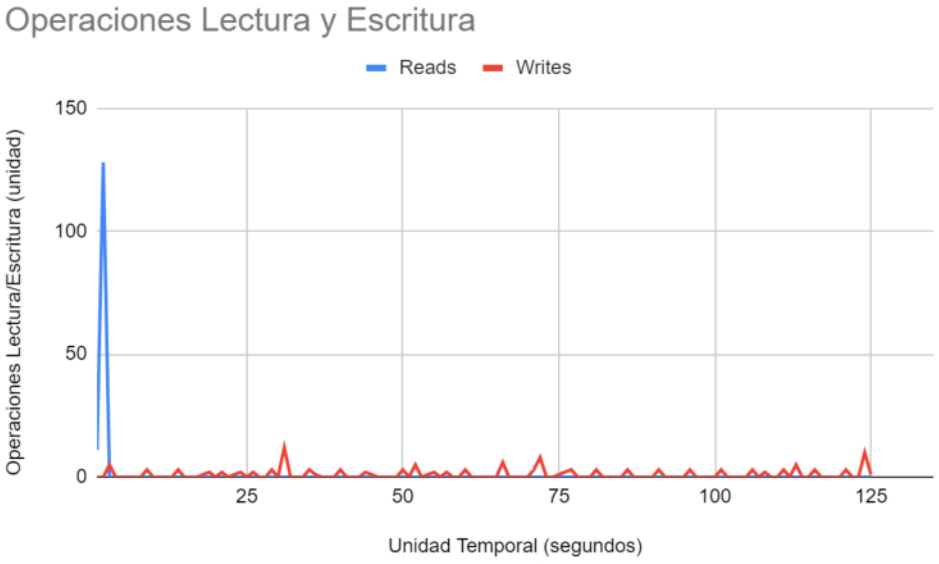

Figura 44. Operaciones lectura y escritura cliente arq. 2

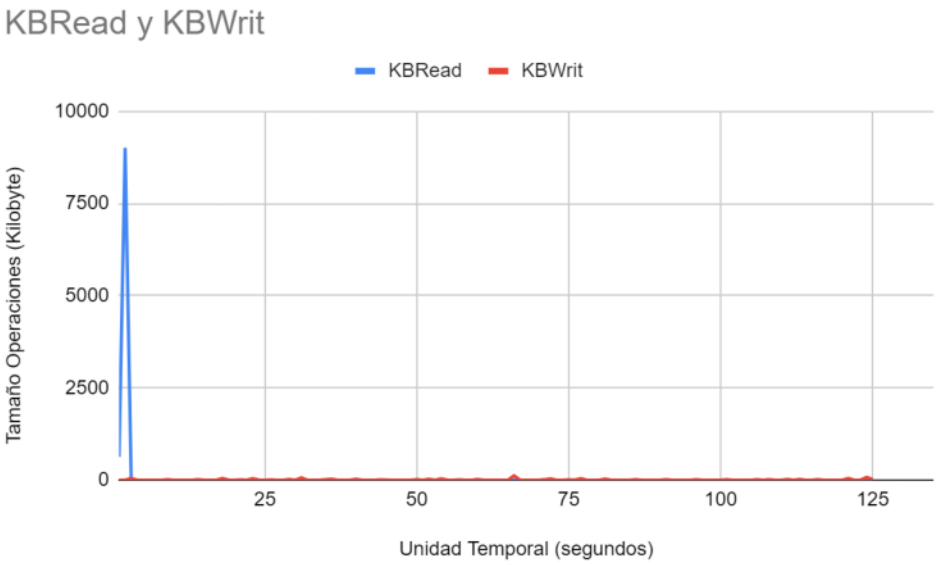

Figura 45. Tamaño operaciones lectura y escritura cliente arq. 2

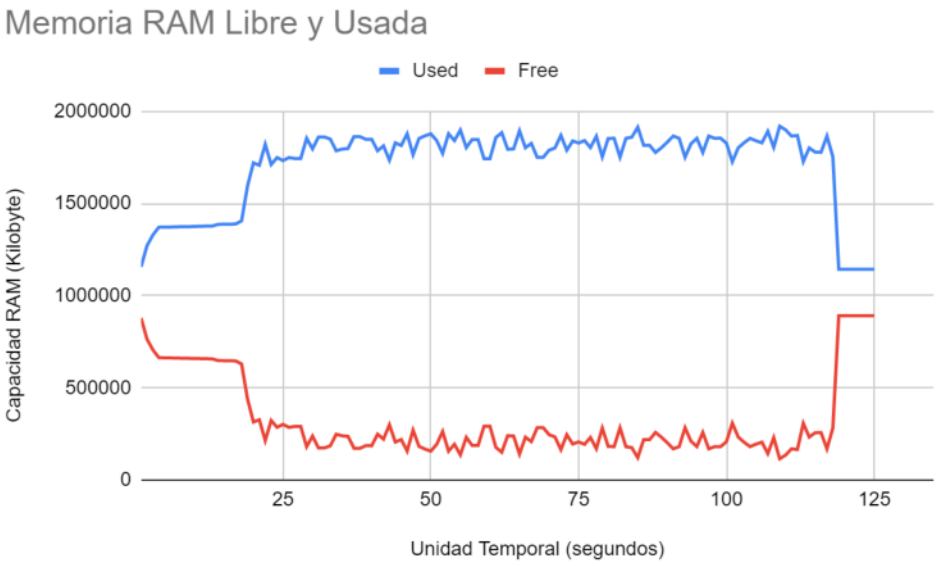

Figura 46. Capacidad Memoria RAM libre vs usada cliente arq. 2

13.2.3.3. Arquitectura 3

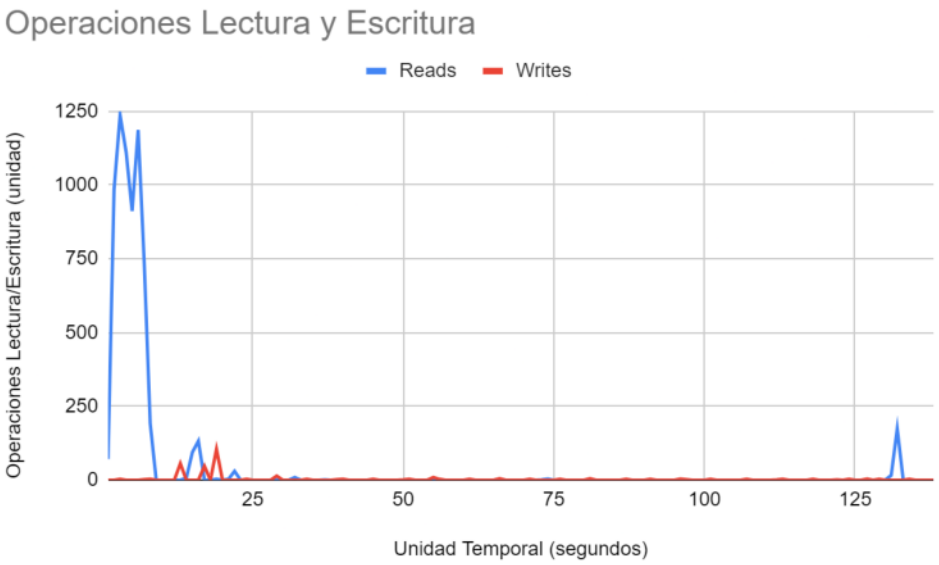

Figura 47. Operaciones lectura y escritura cliente arq. 3

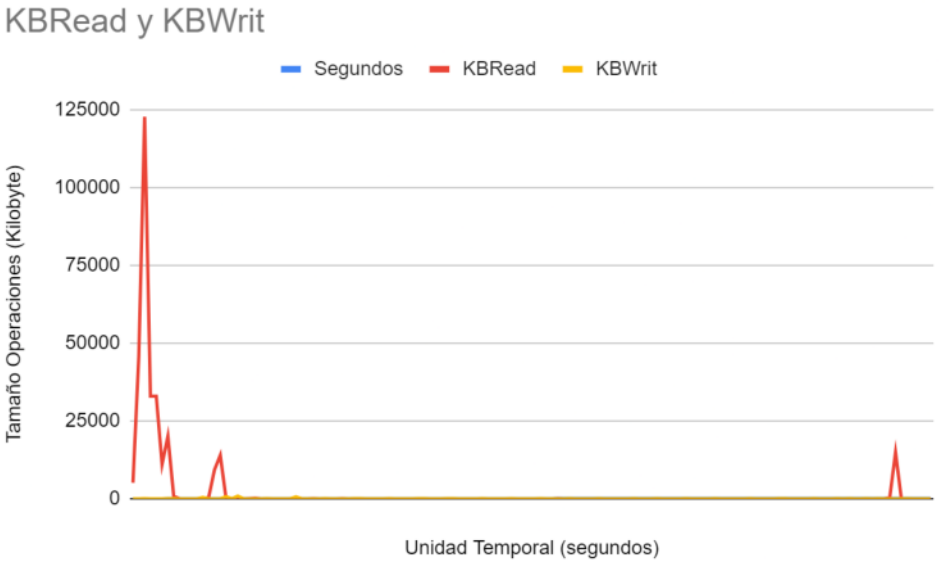

Figura 48. Tamaño operaciones lectura y escritura cliente arq. 3

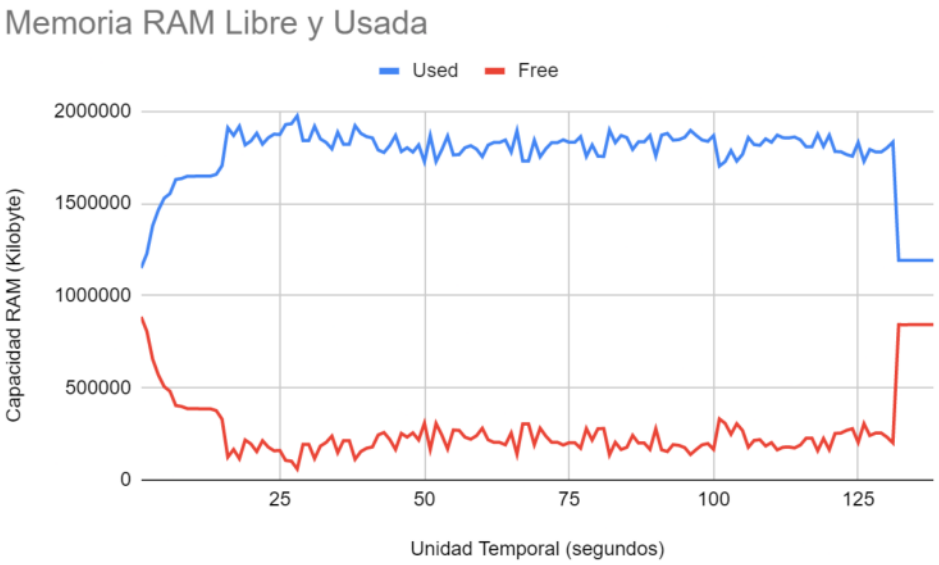

Figura 49. Capacidad Memoria RAM libre vs usada cliente arq. 3

13.2.4. Resultados management

13.2.4.1. Arquitectura 1

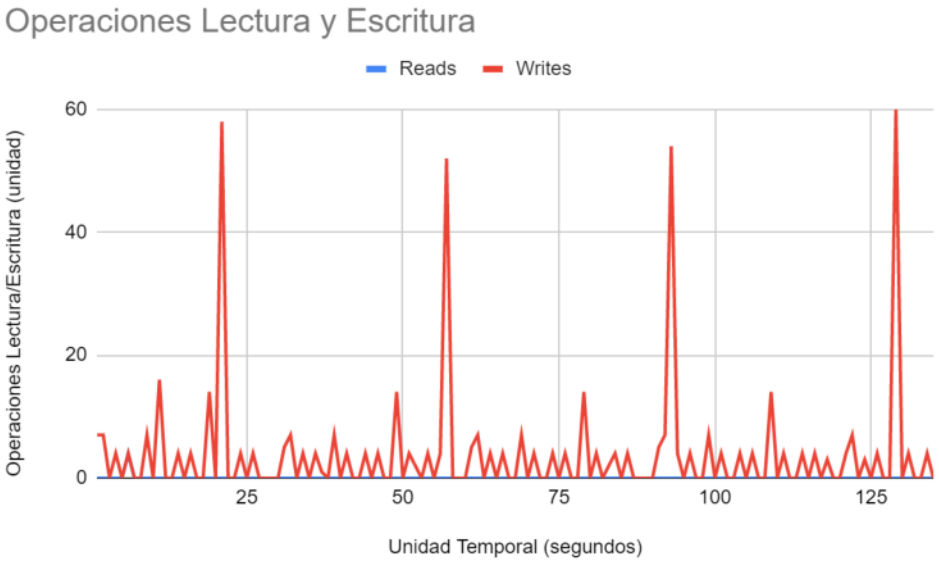

Figura 50. Operaciones lectura y escritura management arq. 1

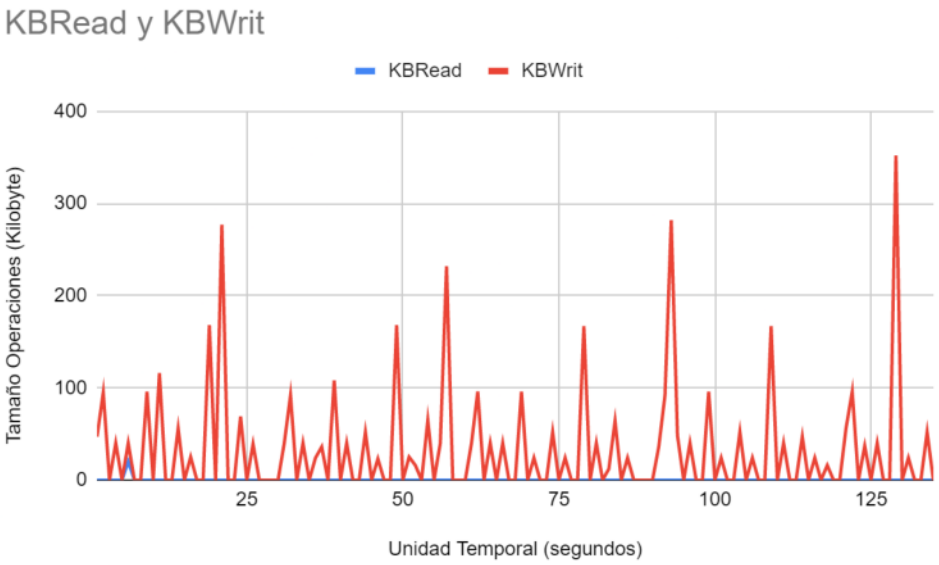

Figura 51. Tamaño operaciones lectura y escritura management arq. 1

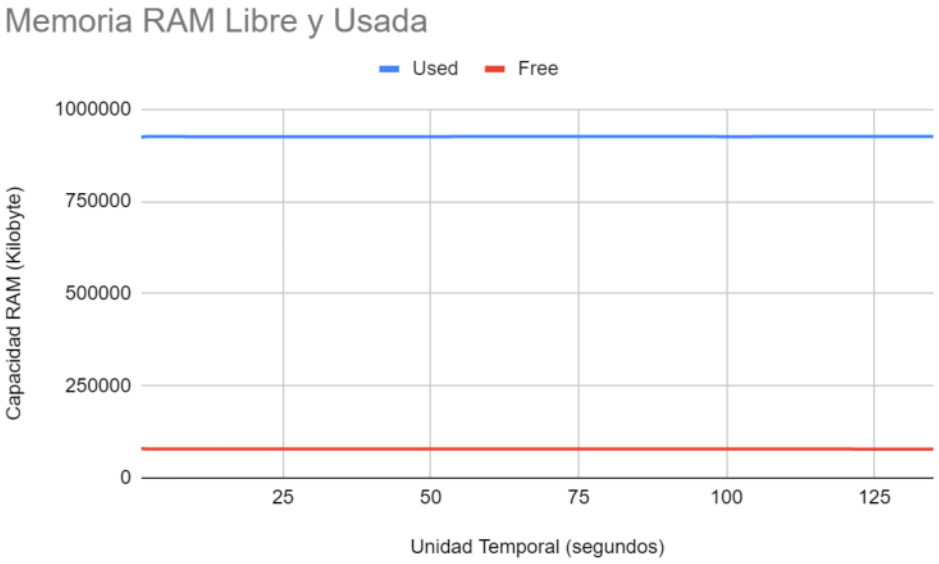

Figura 52. Capacidad Memoria RAM libre vs usada management arq. 1

13.2.4.2. Arquitectura 2

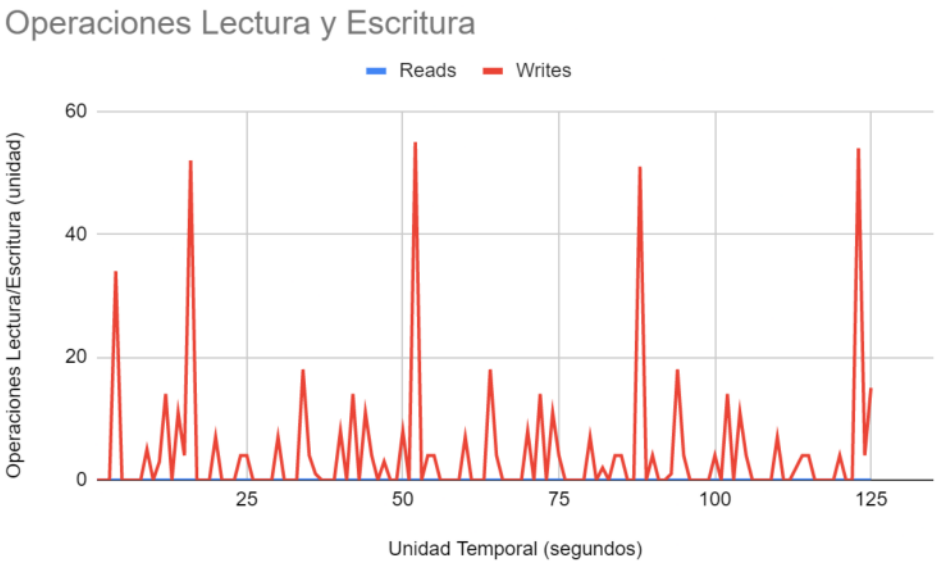

Figura 53. Operaciones lectura y escritura management arq. 2

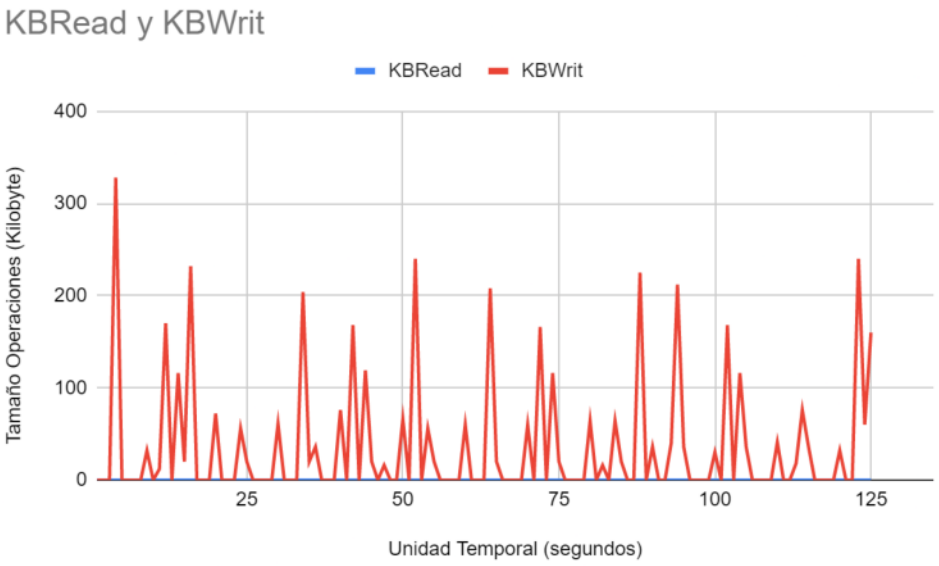

Figura 54. Tamaño operaciones lectura y escritura management arq. 2

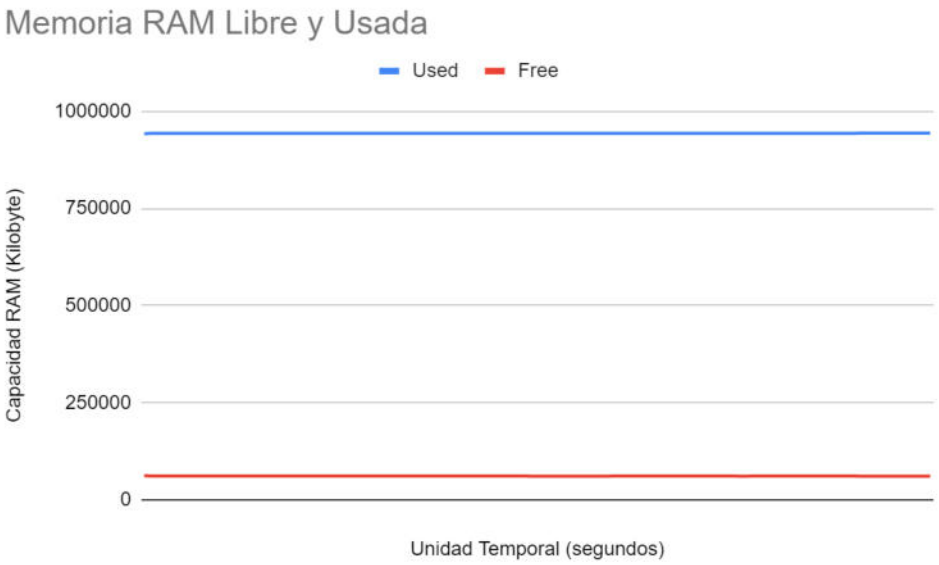

Figura 55. Capacidad Memoria RAM libre vs usada management arq. 2

13.2.4.3. Arquitectura 3

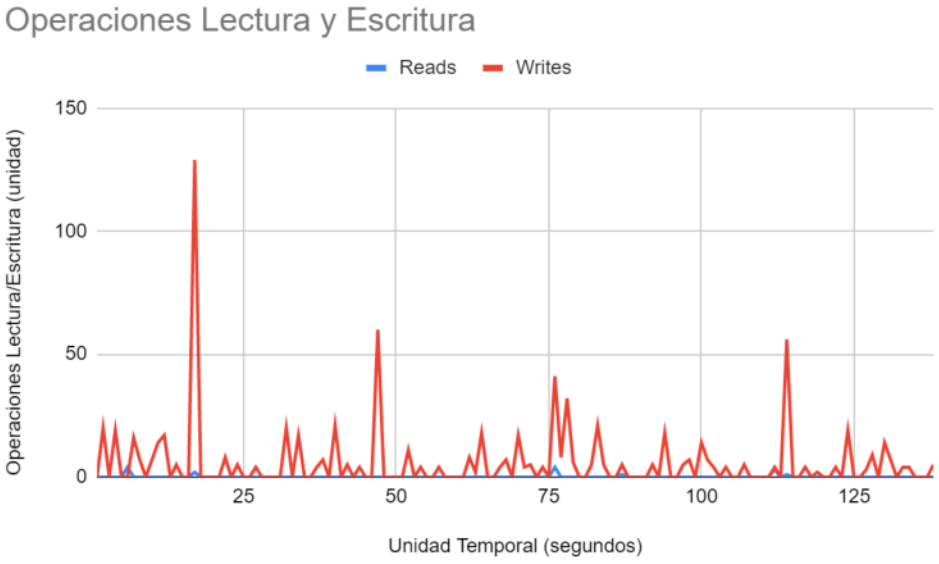

Figura 56. Operaciones lectura y escritura management arq. 3

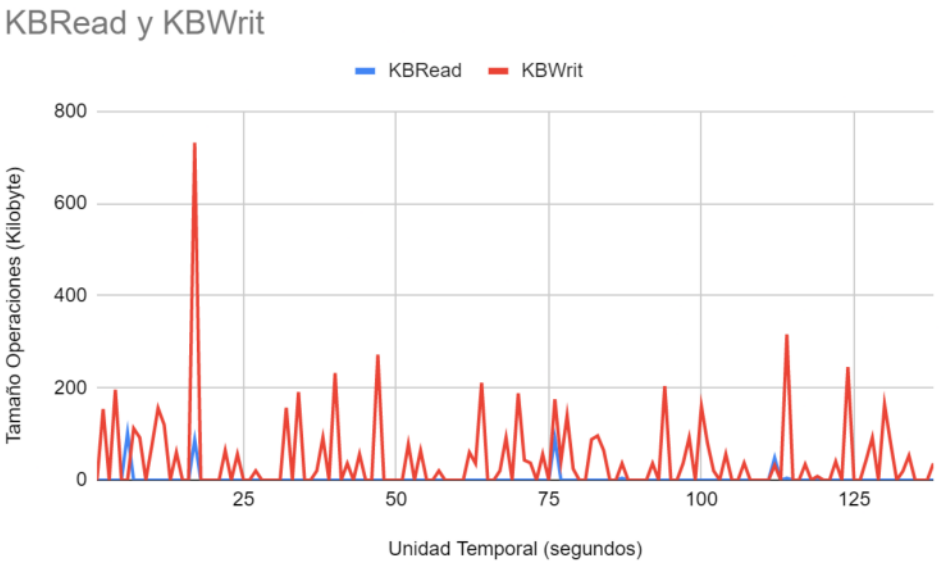

Figura 57. Tamaño operaciones lectura y escritura management arq. 3

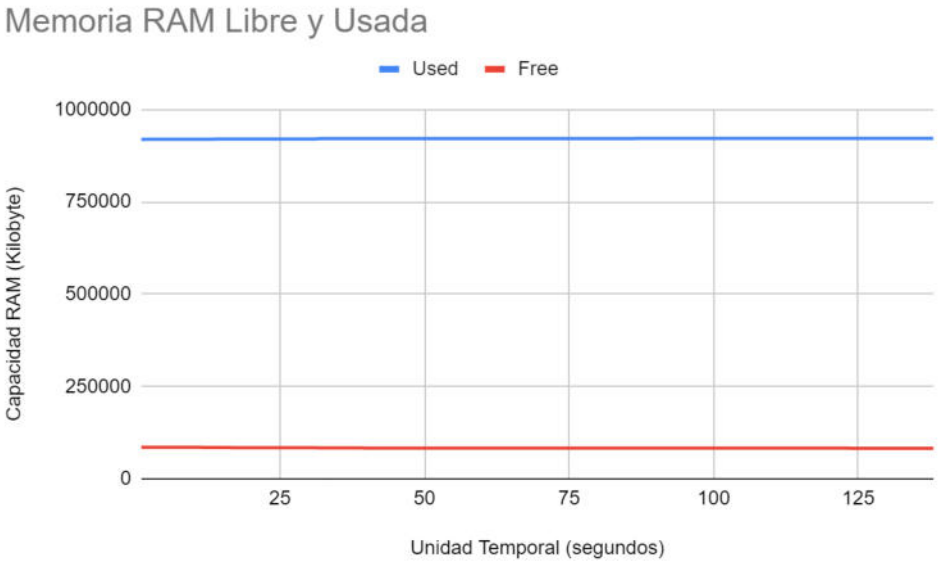

Figura 58. Memoria RAM libre vs usada management arq. 3

13.3. Parametrizaciones alternativas

Con el objetivo de visualizar diferencias en el comportamiento de las gráficas respecto de las parametrizaciones estándar del sistema de archivos, se realizaron pruebas editando los parámetros de tamaño de stripe en servidores, tipo de caché en cliente y memoria RAM en cliente. Los datos reflejados corresponden a la arquitectura 3.

Para que los cambios surtan efecto antes de realizar las pruebas nuevamente, fue necesario formatear la partición del sistema de archivos para cada ejecución. De esta forma los datos se alojaron de acuerdo a los nuevos parámetros establecidos.

13.3.1. Stripe

El aumento del tamaño del stripe es una técnica que se suele utilizar para reducir la sobrecarga de mensajes de parte del cliente hacia los servidores. Por ejemplo, cuando se escriben nuevos datos en un archivo dentro del sistema, el cliente consulta a Metadata los Storage que contienen los stripes del archivo y envía a ellos los mensajes de modificaciones. El tamaño de estos mensajes está directamente vinculado con el tamaño del stripe. Entonces, el objetivo de esta modificación es ver si este cambio produce una disminución o aumento en la cantidad de operaciones que se visualizaron anteriormente.

Lo primero que se realizó fue la modificación de los parámetros:

```
root@cliente1:/mnt/beegfs/fruits-360# beegfs-ctl --setpattern --chunksize=2M /mnt/beegfs
New chunksize: 2097152

root@cliente1:/mnt/beegfs/fruits-360# beegfs-ctl --getentryinfo /mnt/beegfs
Entry type: directory
EntryID: root
Metadata node: meta1 [ID: 1]
Stripe pattern details:
+ Type: RAID0
+ Chunksize: 2M
+ Number of storage targets: desired: 4
+ Storage Pool: 1 (Default)
root@cliente1:/mnt/beegfs/fruits-360#
```

Figura 59. Modificación Chunksize/Stripe

Al aumentar a 2 MB el tamaño del stripe predeterminado por BeeGFS, se obtuvieron los siguientes resultados. Los picos se encuentran asincrónicos, pero no resultan de relevancia para la información brindada por el gráfico.

13.3.1.1. Metadata

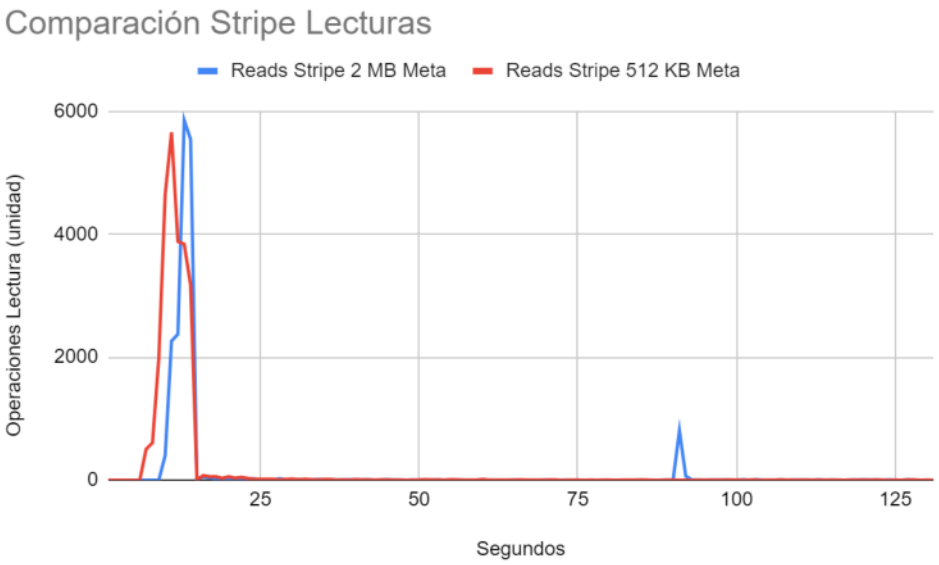

Figura 60. Comparación lecturas stripe modificado Metadata general

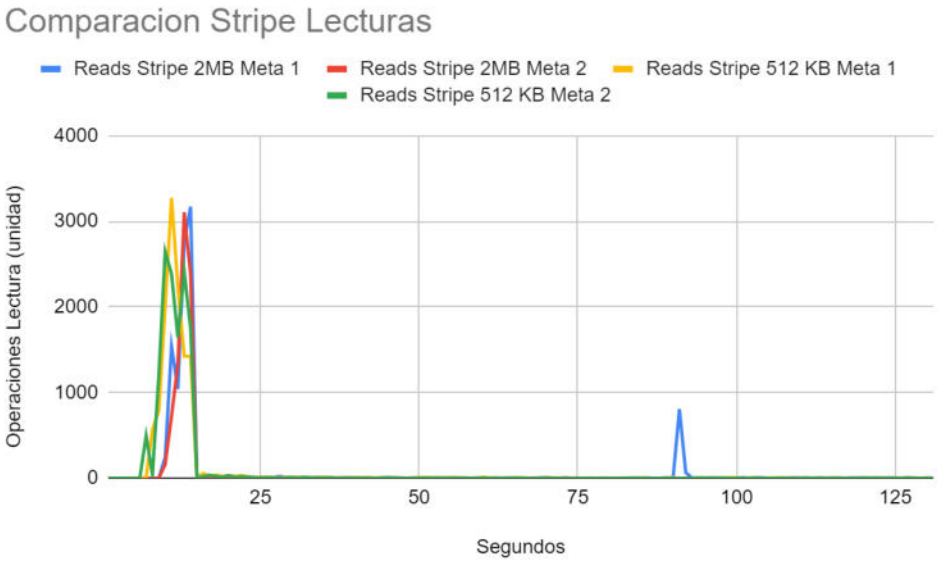

Figura 61. Comparación lecturas stripe modificado Metadata por nodo

De la figura 60, 61 y los resultados se desprenden los siguientes valores:

Operaciones Totales Lectura Stripe 2MB	Operaciones Totales Lectura Stripe 512 KB	Reduccion operaciones
17710	25078	29.38%

Figura 62. Reduccion operaciones lectura en Metadata

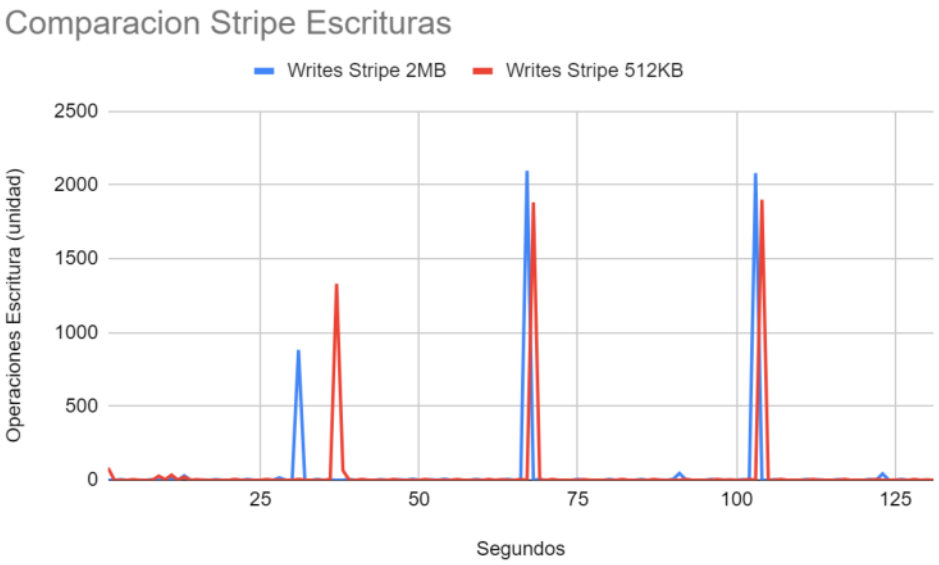

Figura 63. Comparación escrituras stripe modificado Metadata

De la figura 63 y los resultados se desprenden los siguientes valores:

Operaciones Totales Escritura Stripe 2MB	Operaciones Totales Escritura Stripe 512 KB	Reduccion operaciones
12022	12064	0.35%

Figura 64. Reduccion operaciones escritura en Metadata

Tanto para lectura como escritura se realizaron 3 ejecuciones y en todas se obtuvo un promedio similar a los indicados en las figuras 62 y 64.

13.3.1.2. **Storage**

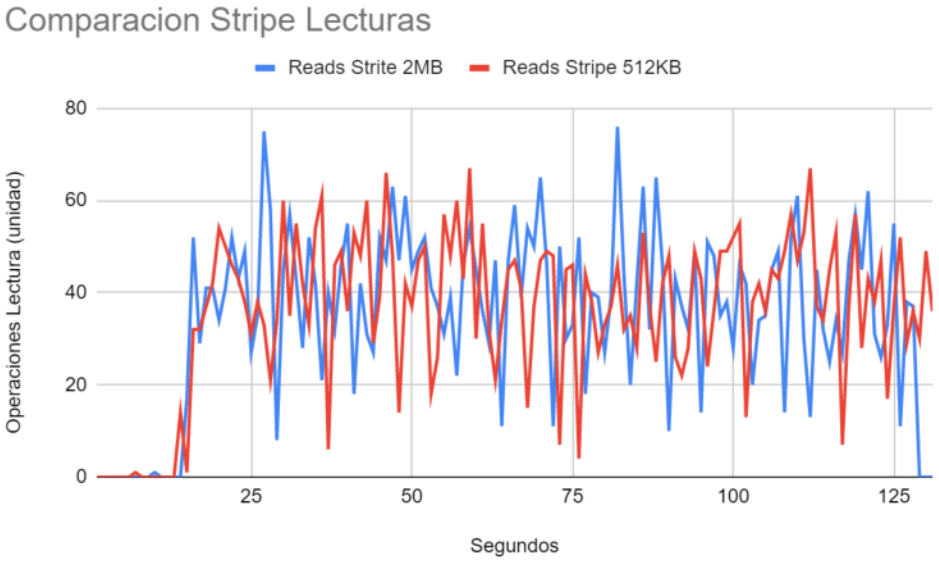

Figura 65. Comparación lecturas stripe modificado Storage

De la figura 65 y los resultados se desprenden los siguientes valores:

Operaciones Totales Lectura Stripe 2MB	Operaciones Totales Lectura Stripe 512 KB	Reduccion operaciones
9124	9853	7.40%

Figura 66. Reduccion operaciones lectura en Storage

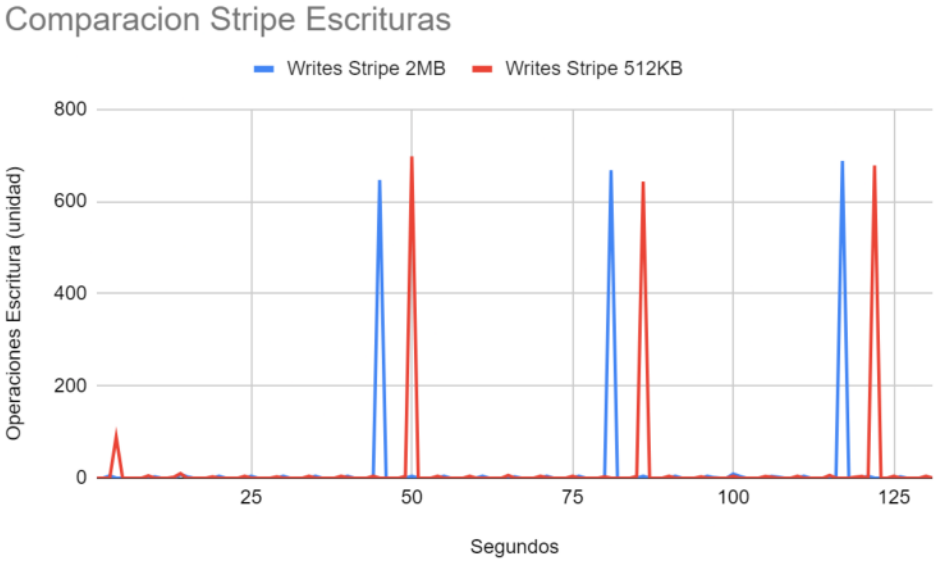

Figura 67. Comparación escrituras stripe modificado Storage

De la figura 67 y los resultados se desprenden los siguientes valores:

Operaciones Totales Escritura Stripe 2MB	Operaciones Totales Escritura Stripe 512 KB	Reduccion operaciones
4307	4751	9.35%

Figura 68. Reduccion operaciones escritura en Storage

Tanto para lectura como escritura se realizaron 3 ejecuciones y en todas se obtuvo un promedio similar a los indicados en las figuras 66 y 68.

13.3.1.3. **Cliente**

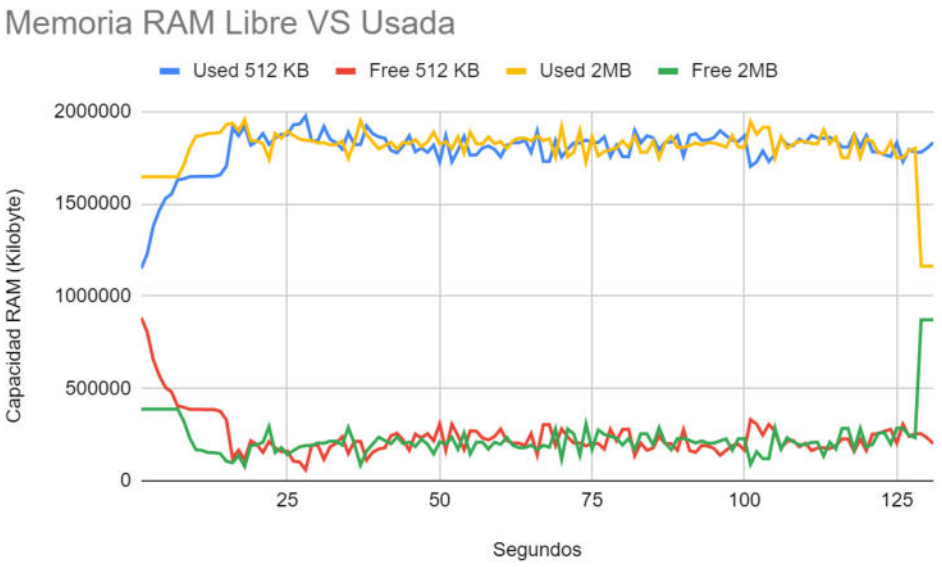

Figura 69. Comparación memoria RAM stripe modificado Cliente

De la figura 69, se aprecia que no se registraron cambios importantes en el uso de la memoria RAM en el lado del cliente.

13.3.2. **Caché**

Se modificó el tipo de caché en el cliente, pasando de Buffered a Native. Esto permitirá un mayor almacenamiento del lado del cliente, en caso de que la aplicación lo permita. La finalidad es detectar si varía el comportamiento en los servidores durante la ejecución de la aplicación.

Para esta prueba se mostrarán los resultados con el cliente corriendo con 2 GB de RAM y con 3 GB de RAM. Esto se debe a que el aumento de esta característica,

sumado al cambio de tipo de caché, modifica drásticamente el comportamiento en los servidores.

13.3.2.1. **Metadata**

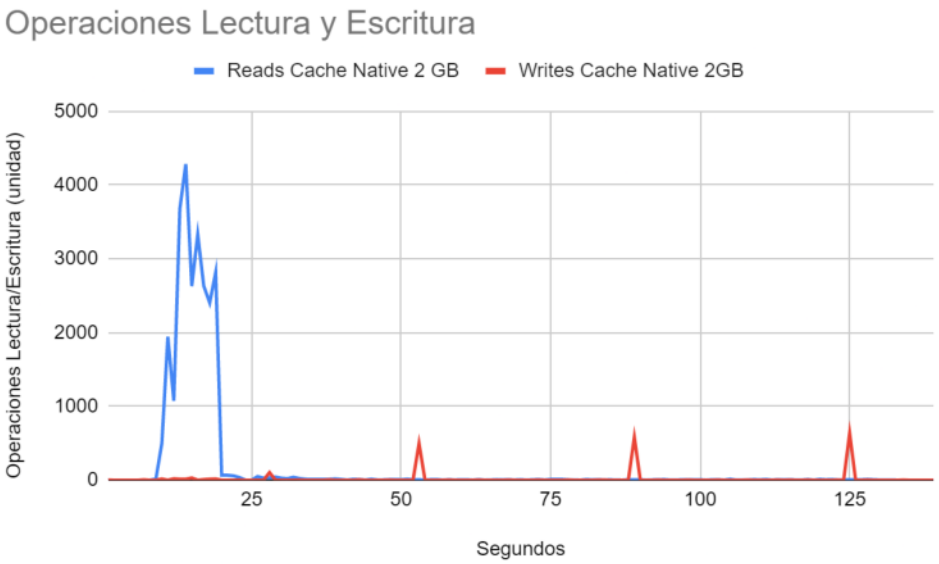

Figura 70. Operaciones Lectura/Escritura caché Native 2 GB Metadata1

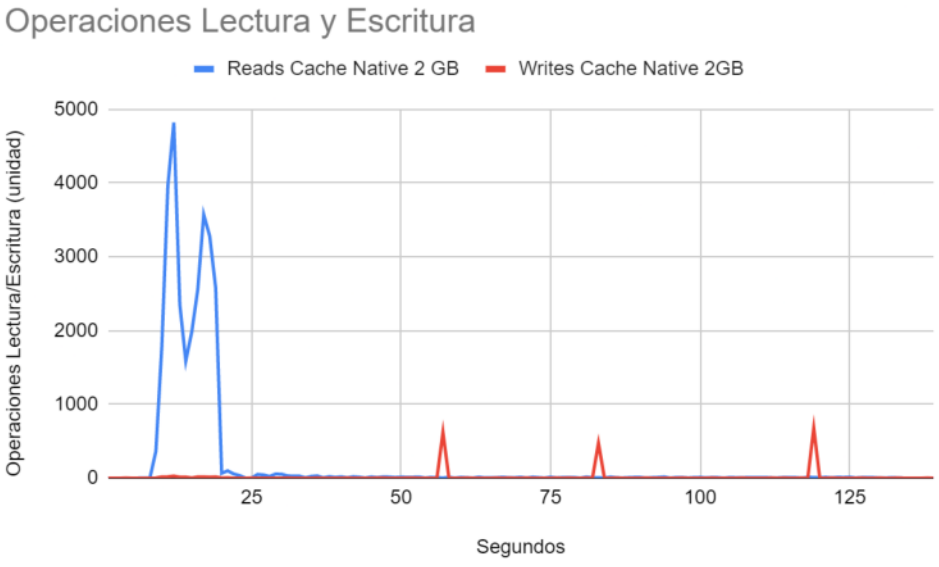

Figura 71. Operaciones Lectura/Escritura caché Native 2 GB Metadata2

Operaciones Totales Lectura caché Native 2 GB	Operaciones Totales Lectura caché Buffered 2 GB	Reduccion operaciones

56358	17710	-218,23%

Figura 72. Reducción operaciones lectura en Metadata

Operaciones Totales Escritura caché Native 2 GB	Operaciones Totales Escritura caché Buffered 2 GB	Reduccion operaciones
4030	12022	198,31%

Figura 73. Reducción operaciones escritura en Metadata

13.3.2.2. **Storage**

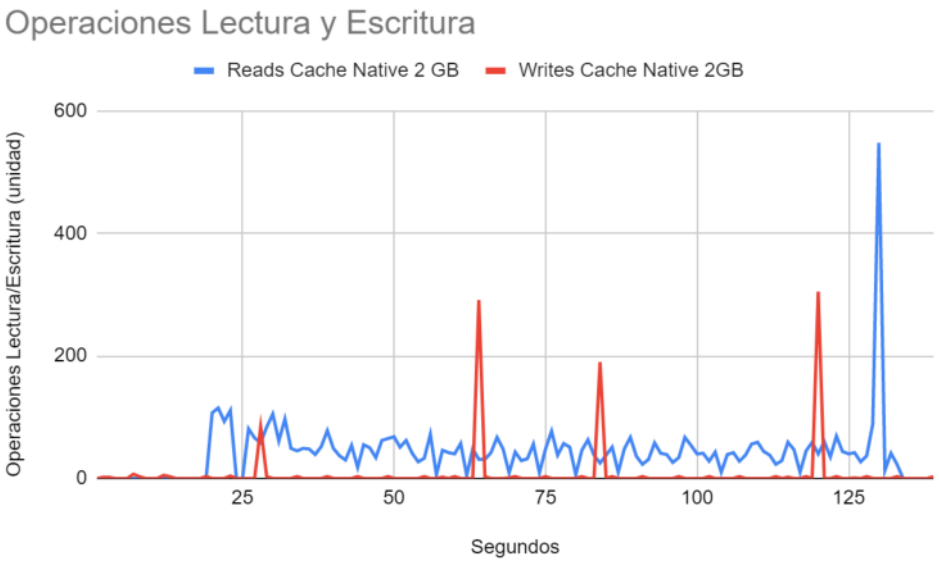

Figura 74. Operaciones Lectura/Escritura caché Native 2 GB Storage1

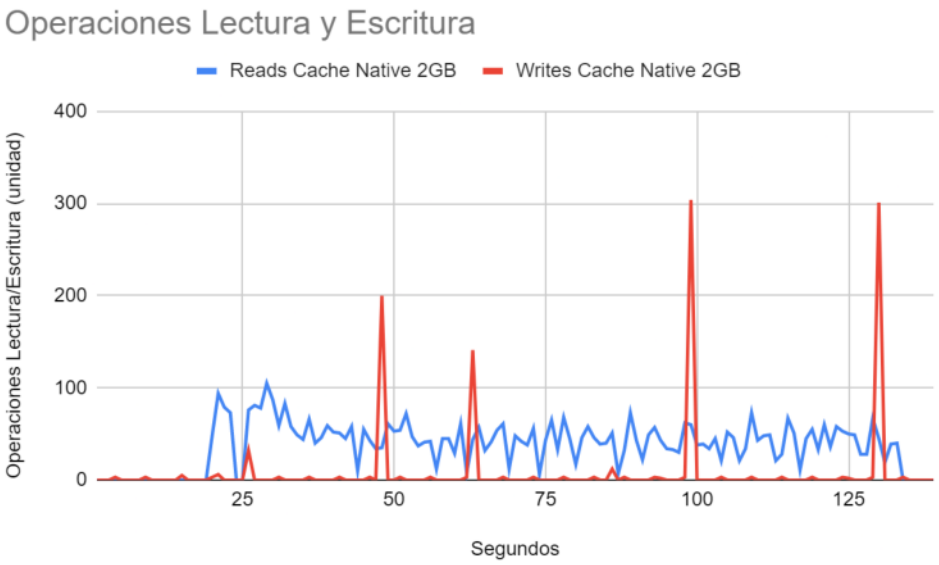

Figura 75. Operaciones Lectura/Escritura caché Native 2 GB Storage2

Operaciones Totales Lectura caché Native 2 GB	Operaciones Totales Lectura caché Buffered 2 GB	Reduccion operaciones
10973	9124	-16.85%

Figura 76. Reducción operaciones lectura en Storage

Operaciones Totales Escritura caché Native 2 GB	Operaciones Totales Escritura caché Buffered 2 GB	Reduccion operaciones
2048	4307	52.45%

Figura 77. Reducción operaciones escritura en Storage

13.3.2.3. **Cliente**

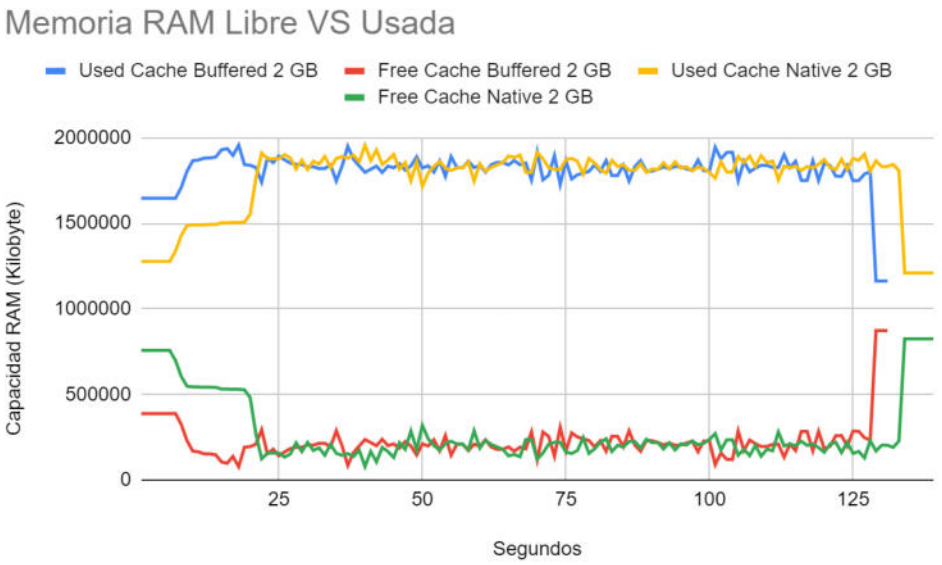

Figura 78. Comparación memoria RAM caché modificada cliente

En los resultados obtenidos con caché Native de 2 GB se describe un comportamiento anómalo de la aplicación. Con este cambio se vieron favorecidas las operaciones de escritura con una reducción de alrededor del 200%. Por otro lado, las de lectura se incrementaron cerca de un 220%. Esta alteración se percibió tanto en Metadata como en Storage, cada uno con sus respectivos porcentajes pero comportamiento similar. La repetición de la simulación no alteró los resultados.

Por el lado del cliente, la memoria RAM no mostró cambios destacables en su comportamiento. Esto seguramente se deba a que con el uso del sistema operativo sumado a la ejecución de la aplicación, los recursos del cliente ya se encontraban siendo utilizados prácticamente al máximo. Se recuerda que la ventaja de caché native, aún en etapa experimental, es el uso de GB de datos temporales frente a los KB que usa Buffered.

Entonces, se concluye que no se recomienda el uso de caché native en ambientes donde el cliente tenga recursos limitados para este.

Ahora bien, se analizará qué sucede cuando se le asigna al nodo cliente 1 GB más de RAM, quedando este con 3 GB. Los siguientes resultados corresponden a una segunda ejecución de la aplicación, ya que la primera arrojó datos similares a la experiencia previa.

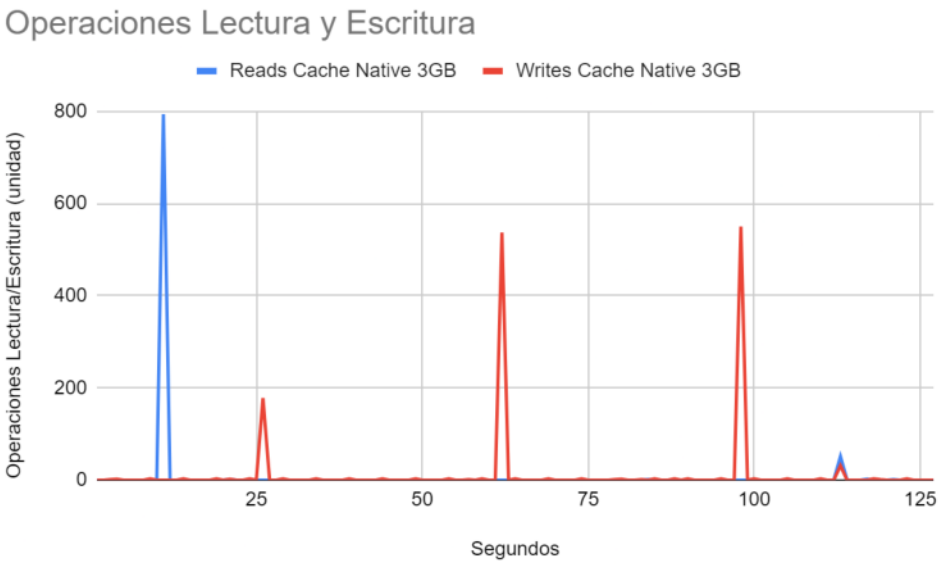

Figura 79. Operaciones Lectura/Escritura caché Native 3 GB Metadata1

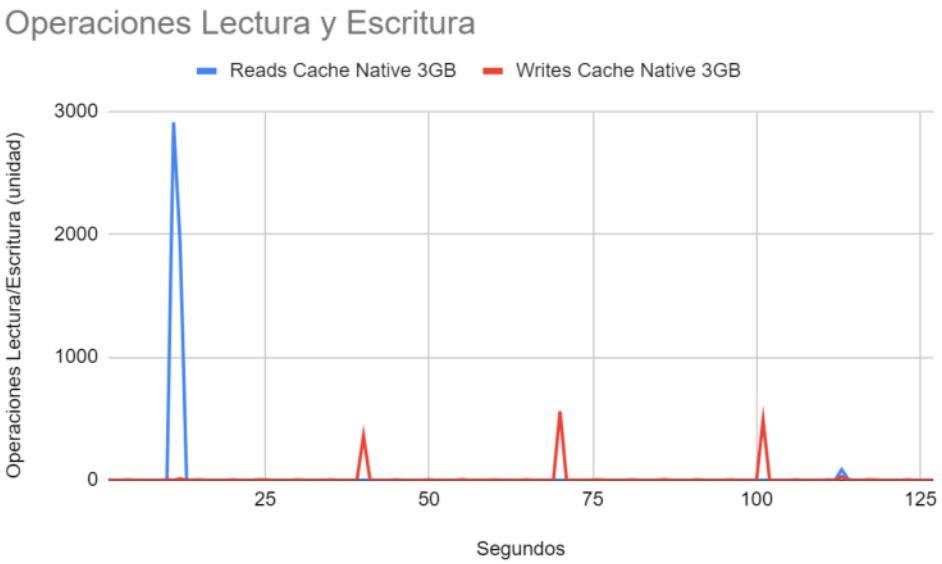

Figura 80. Operaciones Lectura/Escritura caché Native 3 GB Metadata2

Operaciones Totales Lectura caché Native 3 GB	Operaciones Totales Lectura caché Buffered 2 GB	Reduccion operaciones
5783	17710	206.24%

Figura 81. Reducción operaciones lectura en Metadata

Operaciones Totales Escritura caché Native 3 GB	Operaciones Totales Escritura caché Buffered 2 GB	Reduccion operaciones
2898	12022	314.84%

Figura 82. Reducción operaciones escritura en Metadata

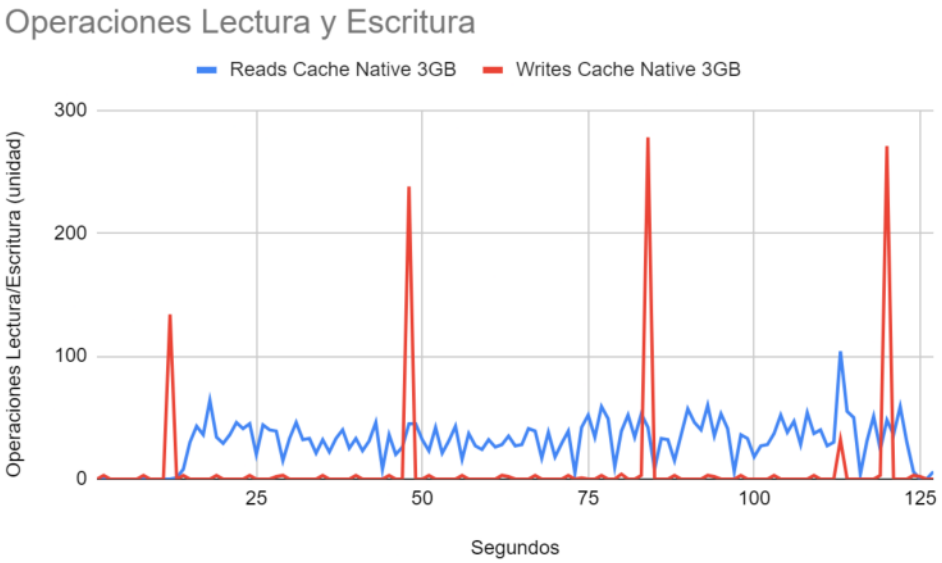

Figura 83. Operaciones Lectura/Escritura caché Native 3 GB Storage1

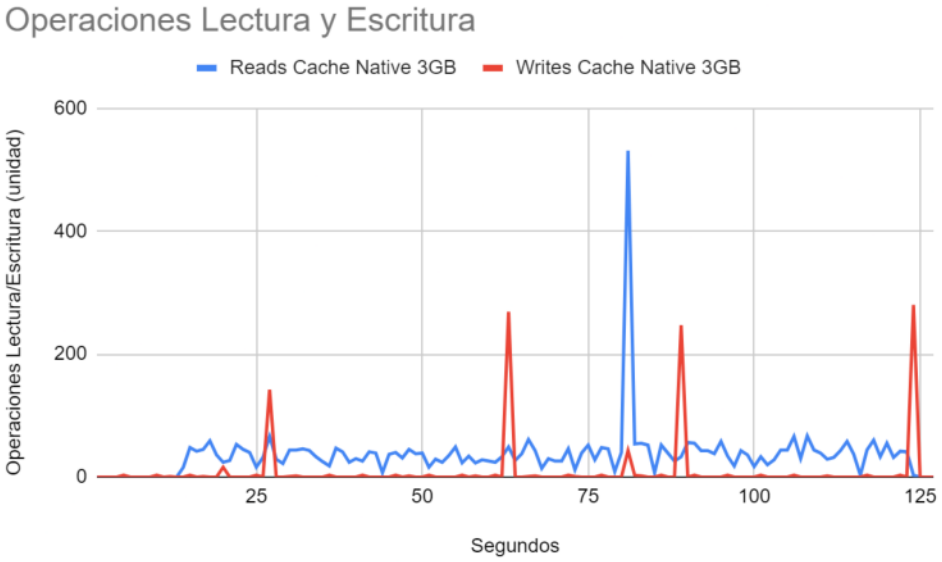

Figura 84. Operaciones Lectura/Escritura caché Native 3 GB Storage2

Operaciones Totales Lectura caché Native 3 GB	Operaciones Totales Lectura caché Buffered 2 GB	Reduccion operaciones
8396	9124	8.67%

Figura 85. Reducción operaciones lectura en Storage

Operaciones Totales Escritura caché Native 3 GB	Operaciones Totales Escritura caché Buffered 2 GB	Reduccion operaciones
2109	4307	104.22%

Figura 86. Reducción operaciones escritura en Storage

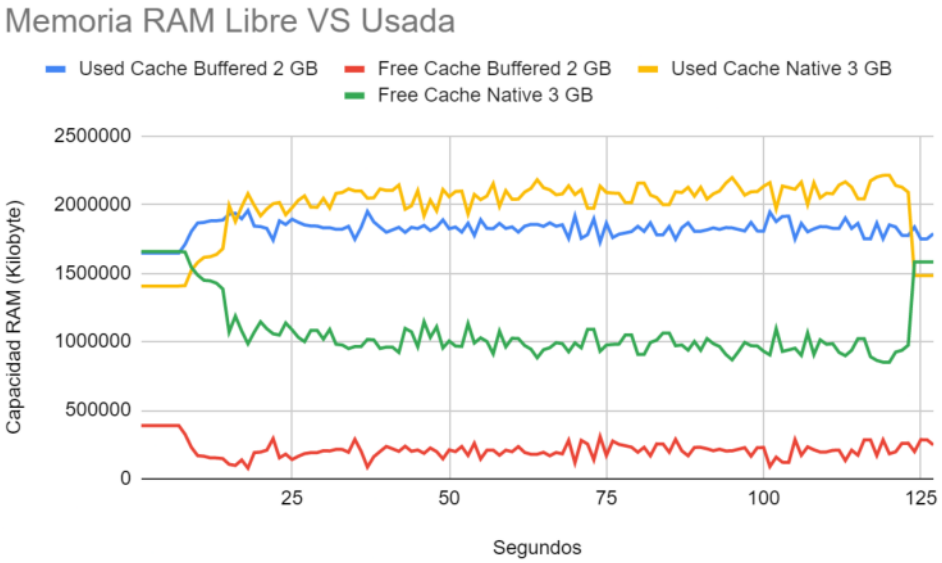

Figura 87. Comparación memoria RAM caché modificada cliente

En la primera ejecución se obtuvieron resultados similares a la caché Buffered. En la segunda se observa un cambio notorio en el comportamiento. Tanto con respecto a caché Buffered como a Native de 2 GB. El aumento de RAM junto al cambio de caché derivó en una reducción sustancial en la carga de los servidores. A su vez, en la tercera ejecución la reducción de operación fue aún más importante.

Por el lado del cliente, se observa en la figura 87, un incremento del uso de la RAM respecto de la caché predeterminada. Esto se debe a que la caché Native permite almacenar GB de datos temporales, siempre que la aplicación lo permita.

14. Consistencia de datos

Para esta prueba se utilizó la herramienta Darshan. El objetivo de la misma es verificar si efectivamente la aplicación de FRUITS-360 y CIFAR10 están trabajando sobre el sistema de archivos paralelo y no sobre una unidad local del nodo.

Esta resulta muy interesante debido a que permite constatar la consistencia de la información obtenida tanto del lado de los servidores, con *collectl*, como del lado del cliente, mediante Darshan.

Luego de instalar Darshan, se procedió a configurarlo para realizar el seguimiento de la aplicación de FRUITS-360. Para esto, se creó un archivo .sh con los parámetros, seguido de la ejecución del software.

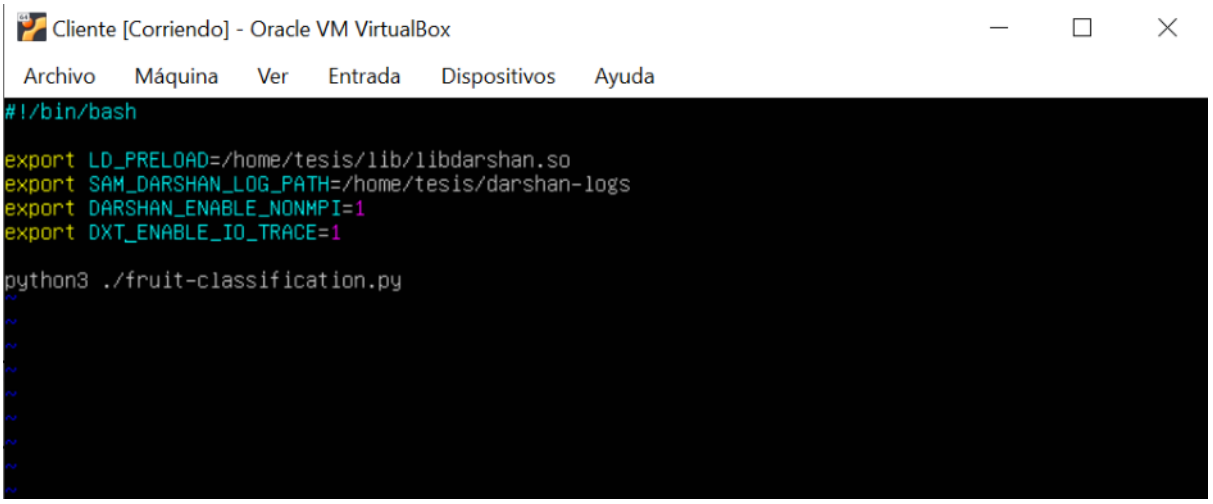

Figura 88. Configuración Trigger Darshan

- DXT_ENABLE_IO_TRACE = 1 : De esta forma se habilita el traceo global de todo tipo de archivos. DXT hace referencia al módulo *Darshan eXtended Tracing*.
- SAM_DARSHAN_LOG_PATCH: Se define la ubicación del directorio de los logs de Darshan
- LD_PRELOAD: Se determina la variable de entorno sobre la aplicación de interés o sobre la carpeta compartida de Darshan.
- DARSHAN_ENABLE_NONMPI=1: Esta variable de entorno es necesario configurarla para que Darshan funcione con aplicaciones no MPI.

A continuación se detallan las pruebas realizadas junto a los resultados obtenidos:

```
# darshan log version: 3.21
# compression method: ZLIB
# exe: python3 ./fruit-classification.py
# uid: 0
# jobid: 12460
# start_time: 1638645376
# start_time_asci: Sat Dec  4 19:16:16 2021
# end_time: 1638645489
# end_time_asci: Sat Dec  4 19:18:09 2021
# nprocs: 1
# run time: 114
# metadata: lib_ver = 3.2.1
# metadata: h = romio_no_indep_rw=true;cb_nodes=4

        # mounted file systems (mount point and fs type)

                # mount entry:  /mnt/beegfs     beegfs
```

Figura 89. Relevamiento Darshan

En la figura 89 se verifica cómo Darshan detecta efectivamente la unidad sistema de archivos BeeGFS. Es importante notar que se visualiza como un espacio externo de montado correspondiente a un sistema de archivo y no a un espacio local del disco.

Respecto de las operaciones se verifica en las siguientes imágenes que la aplicación se encuentra ejecutando correctamente sobre BeeGFS.

```
# ************************************************
# DXT_POSIX module data
# ************************************************

# DXT, file_id: 7540834710928794733, file_name: /mnt/beegfs/fruits-360/fruit-classification.py
# DXT, rank: 0, hostname: cliente1
# DXT, write_count: 0, read_count: 9
# DXT, mnt_pt: /mnt/beegfs, fs_type: beegfs
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX      0   read        0            4691           22      0.0197      0.0206
 X_POSIX      0   read        1               0         4713      0.0213      0.0219
 X_POSIX      0   read        2            4713            0      0.0219      0.0222
 X_POSIX      0   read        3               0         8736      0.0290      0.0290
 X_POSIX      0   read        4            8736            0      0.0290      0.0290
 X_POSIX      0   read        5               0         2677      0.0293      0.0293
 X_POSIX      0   read        6            2677            0      0.0293      0.0293
 X_POSIX      0   read        7               0         2501      0.0294      0.0294
 X_POSIX      0   read        8            2501            0      0.0294      0.0294
```

Figura 90. Logs Darshan

```
# DXT, file_id: 4288638986110284340, file_name: /mnt/beegfs/fruits-360/Training/Orange/r_280_100.jpg
# DXT, rank: 0, hostname: cliente1
# DXT, write_count: 0, read_count: 2
# DXT, mnt_pt: /mnt/beegfs, fs_type: beegfs
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX      0   read        0               0         5228      5.4329      5.4338
 X_POSIX      0   read        1            5228            0      5.4340      5.4342

# DXT, file_id: 1123215251838329278, file_name: /mnt/beegfs/fruits-360/Training/Pomelo Sweetie/r2_114_100.jpg
# DXT, rank: 0, hostname: cliente1
# DXT, write_count: 0, read_count: 2
# DXT, mnt_pt: /mnt/beegfs, fs_type: beegfs
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX      0   read        0               0         5710      5.4394      5.4401
 X_POSIX      0   read        1            5710            0      5.4402      5.4404

# DXT, file_id: 2211352284289533617, file_name: /mnt/beegfs/fruits-360/Training/Pepper Red/r_146_100.jpg
# DXT, rank: 0, hostname: cliente1
# DXT, write_count: 0, read_count: 2
# DXT, mnt_pt: /mnt/beegfs, fs_type: beegfs
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX      0   read        0               0         5386      5.4453      5.4458
 X_POSIX      0   read        1            5386            0      5.4459      5.4461
```

Figura 91. Logs Darshan

De las figuras 90 y 91 se comprueba que realmente la aplicación que se encuentra ejecutando en el cliente está realizando las operaciones de lectura sobre el sistema de archivos analizado. Esto, sumado a la información recolectada por Collectl permite validar la consistencia de datos proporcionada.

De la misma forma, se verificó que la aplicación de CIFAR-10 se esté ejecutando correctamente:

```
# darshan log version: 3.21
# compression method: ZLIB
# exe: python3 /mnt/beegfs/EntreClasific.py
# uid: 0
# jobid: 1572
# start_time: 1616853164
# start_time_asci: Sat Mar 27 10:52:44 2021
# end_time: 1616854037
# end_time_asci: Sat Mar 27 11:07:17 2021
# nprocs: 1
# run time: 874
# metadata: lib_ver = 3.2.1
# metadata: h = romio_no_indep_rw=true;cb_nodes=4

# mounted file systems (mount point and fs type)

# mount entry:  /mnt/beegfs     beegfs
```

Figura 92. Relevamiento Darshan

```
# DXT, file_id: 17012526542315622199, file_name: /mnt/beegfs/EntreClasific.py
# DXT, rank: 0, hostname: cliente-tesis
# DXT, write_count: 0, read_count: 11
# DXT, mnt_pt: /mnt/beegfs, fs_type: beegfs
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX      0   read        0            8533           22     63.1167     63.1174
 X_POSIX      0   read        1               0         8555     63.1179     63.1186
 X_POSIX      0   read        2            8555            0     63.1186     63.1188
 X_POSIX      0   read        3               0        24240     63.1278     63.1278
 X_POSIX      0   read        4           24240            0     63.1278     63.1278
 X_POSIX      0   read        5               0        14422     63.1283     63.1283
 X_POSIX      0   read        6           14422            0     63.1283     63.1283
 X_POSIX      0   read        7               0        24506     63.1287     63.1287
 X_POSIX      0   read        8           24506            0     63.1287     63.1287
 X_POSIX      0   read        9               0         8555     63.7794     63.7803
 X_POSIX      0   read       10            8555            0     63.7804     63.7806
```

Figura 93. Logs Darshan

```
# DXT, file_id: 3822581928163675986, file_name: /mnt/beegfs/cifar-10-batches-py/data_batch_1
# DXT, rank: 0, hostname: cliente-tesis
# DXT, write_count: 0, read_count: 132
# DXT, mnt_pt: /mnt/beegfs, fs_type: beegfs
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX      0   read        0               0      1048576     64.1638     64.1735
 X_POSIX      0   read        1         1048576      1048576     64.1753     64.1835
 X_POSIX      0   read        2         2097152      1048576     64.1853     64.1911
 X_POSIX      0   read        3         3145728      1048576     64.1928     64.1985
 X_POSIX      0   read        4         4194304      1048576     64.2001     64.2040
 X_POSIX      0   read        5         5242880      1048576     64.2057     64.2114
 X_POSIX      0   read        6         6291456      1048576     64.2132     64.2183
 X_POSIX      0   read        7         7340032      1048576     64.2200     64.2249
 X_POSIX      0   read        8         8388608      1048576     64.2266     64.2323
 X_POSIX      0   read        9         9437184      1048576     64.2340     64.2377
 X_POSIX      0   read       10        10485760      1048576     64.2394     64.2444
 X_POSIX      0   read       11        11534336      1048576     64.2461     64.2504
```

Figura 94. Logs Darshan

Como se indicó anteriormente, los logs fueron tomados con el software Darshan. Luego los mismos fueron transformados para ser leídos mediante la herramienta darshan-dxt-parser incluida en el paquete. Esto se realizó mediante la siguiente línea en consola:

```
root@cliente1:/mnt/beegfs/fruits-360# /home/tesis/utils/bin/darshan-dxt-parser /home/tesis/darshan-l
ogs/root_python3_id12460_12-4-69376-4258749990634463318_1638645490.darshan | tee -a ./1234.txt
```

Figura 95. Ejecución darshan-dxt-parser

darshan-dxt-parser permite trasladar la información contenida en el archivo log .darshan a archivos de texto .txt legible por el usuario.

15. Conclusiones

15.1. Generales

Se observa en los resultados obtenidos, que el patrón de trabajo de las aplicaciones de Machine Learning es contraria a los de aquellas que no involucren IA. En programación tradicional, es normal ver gráficos en donde predominen los picos de operaciones de escritura. Por el contrario, este tipo de soluciones presentan grandes volúmenes de

operaciones de lectura, principalmente de mayor estrés en el inicio pero continuas durante la ejecución.

Las operaciones de escritura resultan predictivas respetando las de mayor exigencia una periodicidad idéntica a lo largo de la ejecución de la aplicación. Estos picos de escritura responden a tareas de Checkpoint, una práctica habitual en esta tecnología y que funciona como un backup del estado actual de ejecución. El fin de ello es evitar pérdidas generales en caso de interrupción de la aplicación en algún punto.

A su vez, resulta interesante destacar también cómo el servicio de Metadata en este tipo de aplicaciones cumple un rol crítico para el funcionamiento del sistema. En programación tradicional es costumbre centrar la atención en los storage y cómo responden a operaciones de escritura cada vez más exigentes para el sistema, monitoreando constantemente velocidades y tiempos de respuesta a dichas operaciones, ya que es el servicio más requerido. En Machine Learning, si bien es importante ver qué sucede en los storage y los otros servicios, Metadata cumple un rol fundamental. Este es sometido a una gran cantidad de operaciones de lectura, con accesos aleatorios y concurrentes, a los que debe dar respuesta en tiempo y forma para satisfacer las necesidades de la aplicación. A tal punto que, a partir de visitar la web de varias distribuciones de sistemas de archivos paralelos, se percibe cómo se destacan en el uso de sus productos en aplicaciones del tipo Machine Learning mediante la fácil escalabilidad horizontal de los servicios de Metadata.

Respecto de los tres tipos de arquitecturas propuestas, todas las ejecuciones se realizaron bajo las mismas condiciones de hardware y software, con la aplicación corriendo a 50 épocas. De los resultados se distingue que el incremento tanto de servicios de Metadata como de Storage no aparejó una reducción porcentual significativa en lo que respecta al tiempo de ejecución de la aplicación, de acuerdo a la siguiente tabla:

Ejecución 1 (Segundos)	Ejecución 2 (Segundos)	Ejecución 3 (Segundos)	Reducción 1-2	Reducción 1-3
113	119	125	9%	5%

Figura 96. Tiempos de ejecución

Verificando estos resultados, se entendió por qué las distintas distribuciones de sistemas de archivos paralelos están enfocando la facilidad de adhesión y remoción de nodos de Metadata a entornos de producciones con un simple par de líneas en consola. Esto permite reducir el degradamiento a los que son sometidos los nodos de metadata en aplicaciones de Machine Learning y, además, aumentar el poder de procesamiento de operaciones de ese servicios en aplicaciones que así lo requieran. En este caso en particular, BeeGFS posibilita adherir un nuevo servicio de metadata a la infraestructura simplemente instalando los

paquetes correspondientes y las configuraciones básicas. Lo anterior, junto a un reinicio del servicio de meta, el nodo ya se encuentra operativo para recibir solicitudes de los clientes y dividir las cargas de trabajo con los otros nodos que brinden su mismo servicio, casi como un *Plug & Play*. De igual forma se realiza lo mismo con los otros servicios involucrados, como dato adicional.

Respecto de la caché, debido a que las modificaciones realizadas y probadas mostraron resultados contradictorios, se considera que es necesario evaluar seriamente el incremento de su uso mediante la técnica de caché Native.

Sobre el cambio de stripe, también se estima una estrategia útil para la reducción de operaciones de lectura principalmente en Metadata.

15.2. **Arquitecturas**

15.2.1. **Metadata**

En donde sí se contempla el escalamiento horizontal es en el uso de recursos por parte de los nodos de Metadata. BeeGFS reparte las operaciones entre los nodos disponibles.

Se observa en los resultados una reducción de la carga de trabajo del orden de promedio 50% en la arquitectura 3. Verificando en Metadata, en la arquitectura 1 se obtuvo un pico de operaciones de lectura de 6384 en 1 segundo, mientras que en la arquitectura 3, con la adición de un segundo nodo de metadata, el pico de operaciones de lectura se registró en el nodo Metadata1 con 3274 operaciones, resultando en una reducción de 48,72%.

Max. Reads ARQ. 1	Max. Reads ARQ. 3	Promedio Reducción
6384	3274	48,72 %

Figura 97. Picos operaciones de lectura Metadata

Respecto a la escritura, también se nota una reducción de casi 50% en los nodos. Mientras en el arquitectura 1 tenemos un pico de 4186 operaciones de escritura en 1 segundo, en la arquitectura 3 el pico es de 2095 en el nodo Metadata2.

Max. Writes ARQ. 1	Max. Writes ARQ. 3	Promedio Reducción
4186	2095	49,95 %

Figura 98. Picos operaciones de escritura Metadata

Como se indicó en el apartado de conclusiones generales, se refleja en los resultados fácilmente un patrón de un gran volumen de operaciones de lecturas sobre el servicio de metadata en el inicio de la ejecución de la aplicación, que luego disminuye con el correr de las épocas. Este pico varía de acuerdo al volumen de datos del dataset con el que se encuentre trabajando la aplicación.

Respecto de la memoria RAM, se percibe un incremento en el uso de la misma esperado en los momentos de gran volumen de operaciones de lectura sobre los servidores que luego se restablece a la normalidad. El escalamiento horizontal de este servicio no disminuyó dicho uso en capacidad pero si en duración, casi en un 50%.

15.2.2. **Storage**

A partir de lo visto en los gráficos de los resultados, los storage son sometidos a patrones de lectura de poca exigencia para los nodos, con picos bajos y continuos. En operaciones de escritura - y al igual que sucede en metadata- se advierten picos de operaciones con periodicidades casi idénticas, referentes a checkpoints que realiza la aplicación para salvaguardar la información en caso de detención de la ejecución.

De manera análoga, casi la misma reducción que en Metadata se encuentra en las operaciones de escritura sobre los nodos de Storage. Se tiene un pico en un segundo de 1341 operaciones de escritura en la arquitectura 1, mientras que en la arquitectura 3 el pico ocurre con 698 operaciones en el Storage1.

Max. Writes ARQ. 1	Max. Writes ARQ. 3	Promedio Reducción
1341	698	47,95 %

Figura 99. Picos operaciones de escritora Storage

De modo similar, se advierte una reducción en la exigencia de los nodos con las operaciones de lectura. En la arquitectura 1 se destaca un pico en un segundo de 253 operaciones de lectura, mientras que en la arquitectura 3 el pico ocurre en el nodo storage2 con 74 operaciones.

Max. Reads ARQ. 1	Max. Reads ARQ. 3	Promedio Reducción
235	74	68,51 %

Figura 100. Picos operaciones de lectura Storage

Respecto de la memoria RAM, se aprecia un incremento en el uso en ciertos momentos con periodicidades similares. Al igual que con Metadata, el escalamiento horizontal no redujo este incremento, pero si la cantidad de tiempo en que suceden.

15.2.3. Management

Como se observa en los resultados, el servicio de Management no es relevante o tampoco es afectado por el cambio de arquitecturas o el uso de tecnologías como Machine Learning. Esto resulta esperable ya que se trata de un servicio que se encarga únicamente de las funciones de registro y vigilancia del sistema de archivos.

15.2.4. Cliente

Particularmente, es de destacar del cliente el uso de entre 35% y 40% adicional de memoria RAM en comparación con la utilizada por el sistema operativo para la ejecución total de la aplicación de Machine Learning de Fruits 360. La modificación de las distintas arquitecturas no generó una diferencia notable en estos datos recolectados.

16. Referencias

- Argonne. Darshan-runtime installation and usage. **https://www.mcs.anl.gov/research/projects/darshan/docs/darshan-runtime.html**
- BeeGFS. (5 de noviembre de 2021). BeeGFS Documentation 7.2.5. https://doc.beegfs.io/latest/index.html
- Data Science Association. Parallel vs. Distributed file systems: Time for RAID on Hadoop?. https://www.datascienceassn.org/content/parallel-vs-distributed-file-systems-time-raid-hadoop
- Darshan. (31 de julio de 2009). Welcome to the Darshan project. https://www.mcs.anl.gov/research/projects/darshan/
- Kaggle. (octubre de 2021). Fruits 360. **https://www.kaggle.com/moltean/fruits/activity**
- Mark Seger. (31 de octubre de 2018). Collectl. http://collectl.sourceforge.net/
- Oracle. Oracle VM VirtualBox. https://www.oracle.com/virtualization/virtualbox/
- SAS Institute INC. (2021). Aprendizaje automático. Qué es y por qué es importante.https://www.sas.com/es_ar/insights/analytics/machine-learning.html

17. Bibliografía

- Abramson, David; Jin, Chao; Loung, Justin; Carroll, Jake. (2020). Disponible en https://link.springer.com/content/pdf/10.1007%2F978-3-030-48842-0.pdf. Consultado el 19 de enero de 2022.

- Benquerenca, Nicolás; Bond, Román; Morales, Martin; Encinas, Diego. (2020). Rendimiento de sistema de archivos en arquitecturas distribuidas y paralelas. Disponible en http://conaiisi2020.frsfco.utn.edu.ar/actas.html. Consultado el 25 de noviembre de 2021.

- Bonacoorso, Giuseppe. (2017). Machine Learning Algorithms. Birmingham. Packt Publishing Ltd.

- Chowdhury, Fahim. Zhu, Yue. Heer, Todd. Paredes, Saul. Moody, Adam. Goldstone, Robin. Mohror, Kathryn. Yu, Weikuan. (2019). I/O Characterization and Performance Evaluation of BeeGFS for Deep Learning. Disponible en https://dl.acm.org/doi/pdf/10.1145/3337821.3337902. Consultado el 2 de octubre de 2021

- D. Chitra Devi. V. Rhymend Uthariaraj. (2016). Load Balancing in Cloud Computing Environment Using Improved Weighted Round Robin Algorithm for Nonpreemptive Dependent

- F¨urnkranz, Johannes. (2002). Round Robin Classification. Disponible en https://www.jmlr.org/papers/volume2/fuernkranz02a/fuernkranz02a.pdf. Consultado el 13 de noviembre de 2021.

- Heichler, Jan. (2014). An Introduction to BeeGFS. Disponible en https://m.paperzz.com/doc/402033/an-introduction-to-beegfs-%C2%AE-jan-heichler. Consultado el 15 de septiembre de 2021.

- Lindi, Bjørn. (2012). I/O-profiling with Darshan. Disponible en https://prace-ri.eu/wp-content/uploads/IO-profiling_with_Darshan-2.pdf. Consultado el 15 de septiembre de 2021.

- Mergen, Mark. Uhlig, Volkmar. Krieger, Orran. Xenidis, Jimi. (2006). Virtualization of High-Performance Computing. Disponible en https://www.researchgate.net/profile/Orran-Krieger/publication/220623821_Virtualization_for_high-performance_computing/links/0c960531877aecc328000000/Virtualization-for-high-performance-computing.pdf. Consultado el 15 de noviembre de 2021

- Sale, Ernesto. Rodriguez, Sebastian. (2019). Implementación y Operación de un Cluster HPC utilizando Laboratorios de Computadoras en Horarios de Inactividad. Disponible en https://rtyc.utn.edu.ar/index.php/rtyc/article/download/430/289/1759. Consultado el 1 de diciembre de 2021

- Sterling, T., Becker, D. J., Savarese, D., Dorband, J. E., Ranawake, U. A., Packer, C. V., (1995). BEOWULF: A Parallel Workstation for Scientific Computation. Proceedings of the 24th International Conference on Parallel Processing. Disponible en https://egscbeowulf.er.usgs.gov/geninfo/Beowulf-ICPP95.pdf. Consultado el 20 de noviembre de 2021

- Tasks. Disponible en https://www.hindawi.com/journals/tswj/2016/3896065/. Consultado el 9 de octubre de 2021