Sistema de Archivos Paralelos con Aplicaciones de Machine Learning

Nicolás Benquerenca[1], Román Bond[1], Martín Morales[1,2], Diego Encinas[1,3]

[1]SimHPC-TICAPPS. Universidad Nacional Arturo Jauretche. Florencio Varela, 1888, Argentina.
[2]Centro CodApli. FRLP. Universidad Tecnológica Nacional. La Plata, 1900, Argentina.
[3]Instituto de Investigación en Informática (III-LIDI). Facultad de Informática, Universidad Nacional de La Plata - Centro Asociado CIC. La Plata, 1900, Argentina.

{nbenquerenca, rbond, martin.morales, dencinas}@unaj.edu.ar

Resumen. Se propone la investigación, análisis y evaluación del impacto de aplicaciones del tipo Machine Learning en un sistema de archivos paralelos, a nivel de rendimiento y uso de recursos. Para tal motivo se plantea el estudio del sistema de archivos paralelo BeeGFS, como infraestructura, y el uso de aplicaciones de Machine Learning como herramienta 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.

Palabras clave: Sistemas de Archivos, BeeGFS, Benchmarks, Cloud Computing, Machine Learning.

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, 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 Sistema 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 softwares 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.

3 BeeGFS

Es un sistema de archivos paralelo POSIX (Portable Operating System Interface), independiente del hardware, enfocado en el rendimiento y diseñado para facilitar el uso, la instalación y la gestión. 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 [1], 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 [2].

3.1 Arquitectura

En la figura 1 se visualiza la arquitectura típica de BeeGFS:

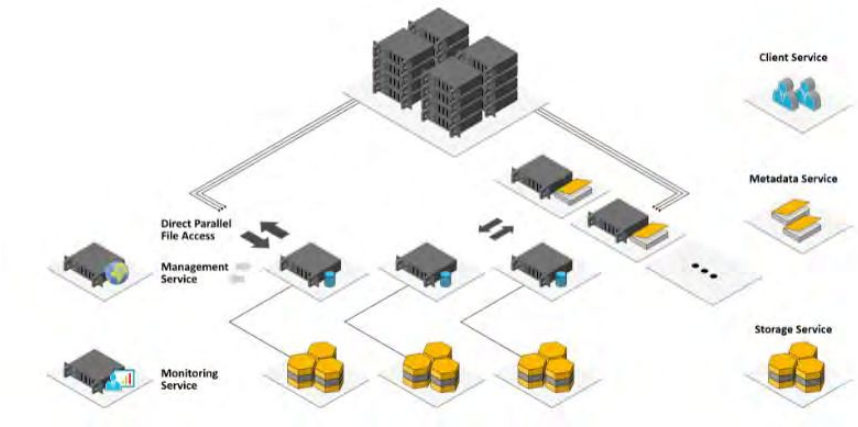


Fig. 1.. Infraestructura BeeGFS típica [3]

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

El despliegue de las arquitecturas se implementó de forma virtual utilizando VirtualBox [4]. Cada máquina virtual cuenta con la siguiente configuración de hardware:

- Servidor metadata/administración/almacenamiento: disco de 60 GB, 1 GB de memoria, 2 cpu y el sistema operativo Ubuntu 20.4.
- Cliente: disco de 60 GB, 2 GB de memoria, 2 cpu y el sistema operativo Ubuntu 20.4.

3.2 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 [5].

Se detallarán los parámetros que resultan relevantes y sobre los cuales se realizarán modificaciones para visualizar el cambio en los comportamientos, en caso de existir.

3.2.1 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.

En la figura 2 se visualizan 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.

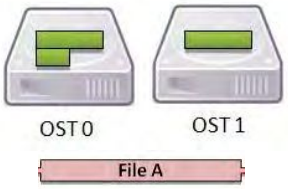


Fig. 2. Stripe [6]

3.2.2 Cache

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 [7]:

- 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.

3.2.3 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 en la arquitectura 3 editando los parámetros de tamaño de stripe en servidores, tipo de caché en cliente y cantidad de memoria RAM en cliente. Los cambios son los siguientes:

- Stripe: BeeGFS establece de manera predeterminada el uso de un stripe de 512K. Se aumentó a 2 MB. 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.
- 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.
- Memoria RAM: Se aumentó la cantidad de memoria en el cliente de 2 GB a 3 GB.

4 Machine Learning

A partir de la lectura de la bibliografía actual [8], 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.

4.1 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.
La gran mayoría de aplicaciones de machine learning necesitan de datos previos para poder entrenar sus modelos. Dependiendo del tipo de problema, el tamaño del dataset puede ser un factor crítico en cuanto operaciones de lectura y escritura para los sistemas de archivos paralelos.

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.

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 [9].

5 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 [10]. En el presente trabajo, se utilizará la siguiente aplicación de Machine Learning para medir el rendimiento:

- FRUITS 360: Es un dataset de imágenes que contiene imágenes de frutas y vegetales de 100x100 píxeles en 131 clases [11]. Está compuesto por 90483 en total, donde de ellas 67692 son de entrenamiento y 22688 de prueba. Un peso aproximado de 1,2 GB.

6 Herramientas útiles

6.1 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 [12][13]. 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.

6.2 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 [14][15].

7 Resultados

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.

7.1 Resultados metadata

7.1.1 Arquitectura 1

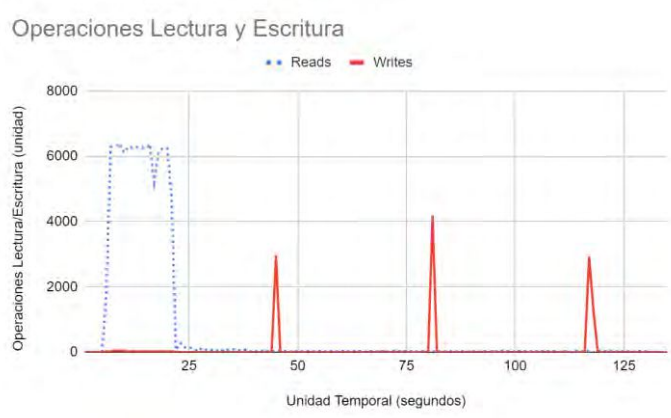


Fig. 3. Operaciones lectura y escritura metadata arq. 1

7.1.2 Arquitectura 3

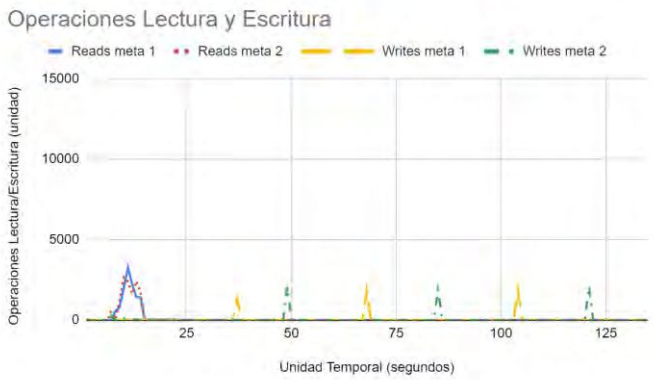


Fig. 4. Operaciones lectura y escritura metadata arq. 3

En la arquitectura 2 no se reflejaron diferencias significativas respecto de la Fig.3.

7.2 Resultados storage

7.2.1 Arquitectura 1

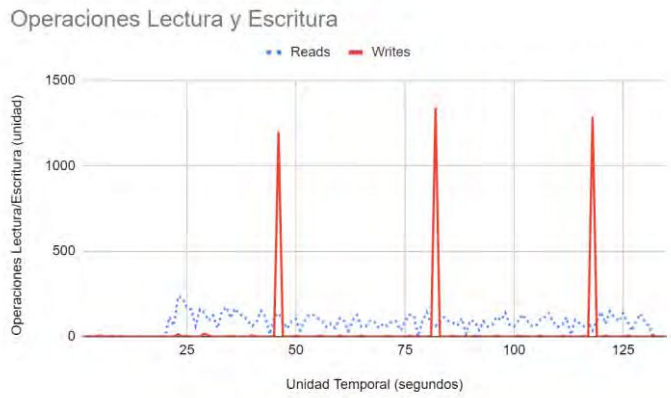


Fig. 5. Operaciones lectura y escritura storage arq. 1

7.2.2 Arquitectura 2

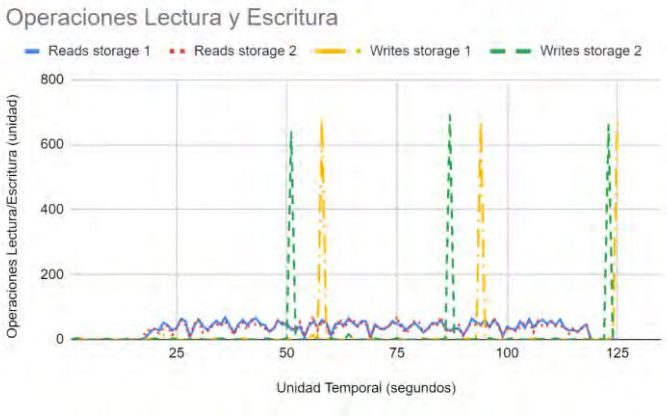


Fig. 6. Operaciones lectura y escritura storage arq. 2

En la arquitectura 3 no se reflejaron diferencias significativas respecto de la figura 6.

7.3 Resultados cliente

7.3.1 Arquitectura 1

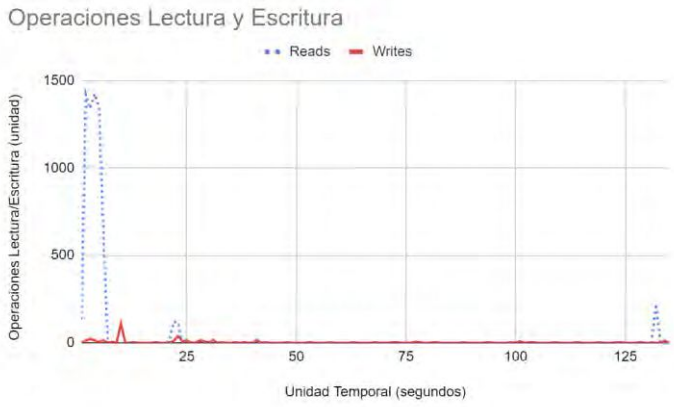


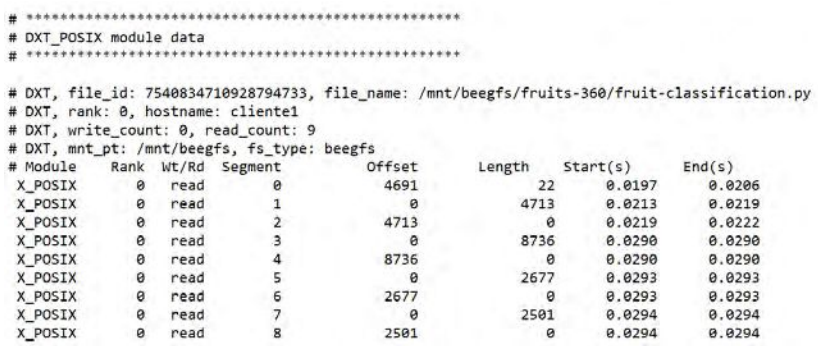
Fig. 7. Operaciones lectura y escritura cliente arq. 1.

En las arquitecturas 2 y 3 no se reflejaron diferencias significativas respecto de la figura 7.

8 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 está 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.



```
# ***************************************************
# 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
```

Fig. 8. Logs darshan.

De la figura 8 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.

9 Conclusiones

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.

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:

Tabla 1. Tiempos de ejecución

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

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%.

Tabla 2. Picos operaciones de lectura Metadata

Max. reads arq.1	Max. reads arq.3	Promedio reducción
6834	3274	48,72%

Verificando los resultados, se entiende 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 producción 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 servicio 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 metadata, el nodo ya se encuentra operativo para recibir solicitudes de los clientes y dividir las cargas de trabajo con los otros nodos que brinden la misma función, casi como un Plug & Play. De igual forma se realiza lo mismo con los otros servicios involucrados, como dato adicional.

Respecto a los storages, a partir de lo visto en los gráficos de los resultados, 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.

Referencias

1. Chowdhury, F., Zhu, Y., Heer, T., Paredes, S., Moody, A., Goldstone, R., Mohror, K., Yu, W.: I/O Characterization and Performance Evaluation of BeeGFS for Deep Learning (2019).
2. Heichler, J.: An Introduction to BeeGFS (2014).
3. BeeGFS Arquitecture, https://doc.beegfs.io/latest/architecture/overview.html
4. Mergen, M., Uhlig, V., Krieger, O., Xenidis, J.: Virtualization of High-Performance Computing (2006).
5. BeeGFS Documentation, https://doc.beegfs.io/latest/index.html
6. High Performance & Scientific Computing, https://oit.utk.edu/hpsc/isaac-open/lustre-user-guide/
7. Client Side Caching Modes, https://doc.beegfs.io/latest/advanced_topics/client_caching.html
8. Yang. Q., Liu, Y., Chen, T., Tong, Y.: Federated Machine Learning: Concept and Applications (2019).
9. Kaggle, https://www.kaggle.com/
10. Benquerenca, N.; Bond, R.; Morales, M.; Encinas, D.: Rendimiento de sistema de archivos en arquitecturas distribuidas y paralelas (2020).
11. Fruits 360, https://www.kaggle.com/datasets/moltean/fruits
12. Collectl, http://collectl.sourceforge.net/

13. Kunkel, J. M., Betke, E., Bryson, M., Carns, P., Francis, R., Frings, W., Laifer, R., Mendez, S.: Tools for Analyzing Parallel I/O (2019)
14. Welcome to the Darshan project, https://www.mcs.anl.gov/research/projects/darshan/
15. Lindi, B.: I/O-profiling with Darshan (2012).