Implementación visual de los metadatos generados por las operaciones de escritura del GFS

David Edgardo Castillo Rodríguez
ITAM, México
david.castillo@itam.mx

I. Introducción

El *Google File System* [1], es un sistema semi descentralizado de administración de archivos. En terminos generales, este sistema está compuesto de un nodo maestro conocido como *master*, de un conjunto de servidores conocidos como *chunkservers*, y de una aplicación de cliente conocida como *client*, mediante la cual un usuario (client) puede comunicarse con el *master* y con los *chunkservers*. Cada uno de estos componentes tiene sus propias funcionalidaddes, las cuales, de manera resumida, son:

1. *Master:* es de suma importancia, ya que entre sus funciones, este es el encargado de administrar todos los metadátos del sistema. También, le indica al *client* qué *chunkservers* debe contactar para llevar a cabo una operación de lectura, escritura o actualización de archivos. También es el encargado de monitorear el estado de los *chunkservers*, así como de contar con protocolos de recuperación.
2. *Chunkservers:* es donde toda la información está guardada de manera particionada, es decir, cada vez que un archivo es generado, este es particionado, donde cada elemento de la partición es guardado en algún *chunkserver* (el *master* decide en qué *chunkserver*).
3. *Client:* le permite a un usuario leer, escribir o actualizar algún archivo en particular. Para ello, el *client* primero se comunica con el *master*, y este segundo le indica en qué *chunkservers* debe consultar, guardar o actualizar la información, según sea el caso.

La arquitectura de estos tres componentes puede verse en la figura 1.

Dicho lo anterior, en este proyecto se ha realizado una implementación visual de los metadatos generados por las operaciones de escritura del sistema de administración de archivos *Google File System* [1]. Esta implementación consta de tres componentes, las cuales son una simulación del *master*, una simulación de los *chunkservers*, y un simulación de la aplicación para el *client*, las cuales se describen a más detalle a continuación.

II. Componentes del sistea implementado

Esta implementación se realizó en el lenguaje de programación *Phyton* mediante la librería *rpyc* (*Remote Python Call*), esta permite invocar procedimientos de manera remota, con lo cual es posible simular la interacción de cada uno de los componentes del *Google File System*.

Los componentes implementados y sus respectivos métodos son los siguientes:

II-A. master

Este componente se instancia al invocar la clase *MasterService*, la cual cuenta con los siguientes atributos y métodos:

Los siguientes atributos son parámetros de configuración:

1. *num_chunks:* es el número de servidores *chunk* que se instancian cada vez que se crea un objeto de la clase *MasterService*
2. *partitons:* es el número de particiones en que se particiona un archivo.

Los siguientes atributos son necesarios para la simulación del *master*:

1. *filedict:* es un diccionario donde se guardan los metadatos, es decir, se guarda el archivo y su respectiva ubicación.
2. *chunksize:* es un diccionario que le indica al master el tamaño de cada *chunkserver*.

Los siguientes métodos pueden ser accesados de manera remota por el *client*

1. *exposed_restart:* este método reestablece los parámetros de configuración del *master*, por parámetros dados por el usuario. En realidad, el cliente no debería tener acceso a los parámetros de configuración del *master*, pero para nuestra implementación visual esto es necesario.
2. *exposed_read:* este es un procedimiento externo ejecutado por el *client*, mediante este procedimiento el *client* le pregunta al *master* por la existencia de un archivo, en caso de que el archivo exista, el master responde con su ubicación así como con el id del archivo.
3. *exposed_check:* este método es invocado de manera remota por el *client* cuando este quiere escribir un archivo. En caso de que el archivo ya exista, el master le informa al *client* que el archivo ya existe, en caso contrario, el *master* le indica al *client* en qué *chunkservers* debe guardar cada parte del archivo, y con qué id.

Los siguientes métodos ayudan a la operación del *master*:

1. *start_chunks_servers:* con este método el *master* crea instancias de los *chunkservers*.
2. *create_handle:* con este método, el *master* asigna un único id a cada partición de un archivo dado. También con este método, el master guarda los metodatos de las particiones de un archivo.

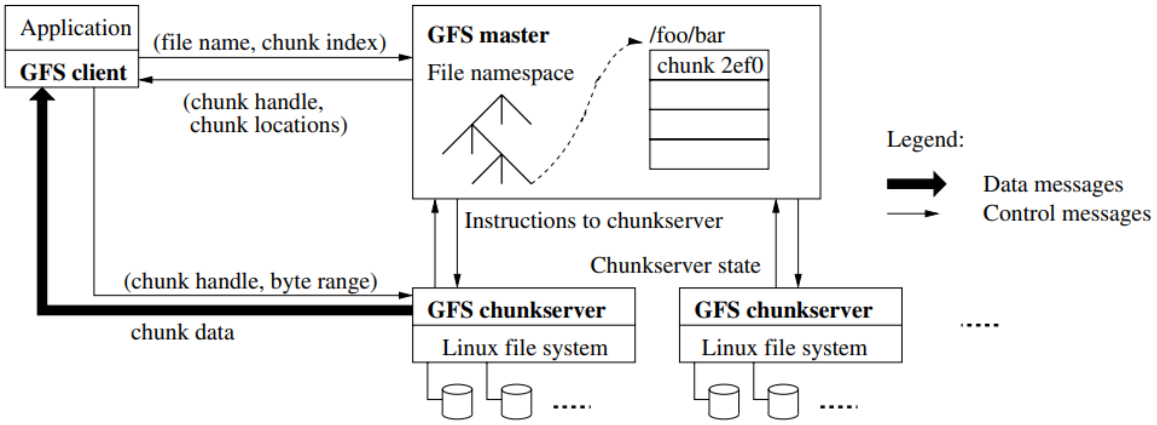

Figura 1. Arquitectura del GFS, fuente [1].

II-B. chunk

Este componente se instancia al invocar la clase *ChunkService*. Una vez instanciado este componente, podemos simular cualquier cantidad de *chunkservers* al crear objetos de la clase *exposed_Chunkmini*. Estos objetos pueden ser creados de manera remota ya sea por el *master* o por el *client*. Cada *chunkserver* instanciado cuenta con los siguientes atributos:

1. *name:* es el id asignado al *chunkserver*, es un atributo numérico mayor a cero.
2. *storage:* aquí se guarda el espacio ocupado por los archivos escritos.
3. *capacity:* es la capacidad de almacenamiento total del *chunkserver*.
4. *local_filesystem_root:* es la ubicación donde el *chunkserver* escribe la información mandada por el *client*.

Cada *chunkserver* instanciado cuenta con los siguientes métodos:

1. *exposed_write:* es un método remoto ejecutado por el *client*, mediante el cual el *cliente* realiza una operación de escritura.
2. *exposed_read:* es un método remoto ejecutado por el *client*, mediante el cual el *cliente* realiza una operación de lectura.
3. *exposed_upcapacity:* es un método remoto mediante el cual el cliente puede incrementar la capacidad de memoria del *chunkserver*. El *client* no debería tener acceso a esta información, pero para nuestra implementación nos ayudará a visualizar la memoria ocupada de cada *chunkserver*.

II-C. client

Este componente lo simulamos al llamar de manera remota los métodos o atributos de las clases *MasterService* y *ChunkService*, para ello definimos los siguientes métodos:

1. *master_read:* este método lee un archivo, para ello primero le manda al *master* el nombre del archivo y la partición a leer, mediante la ejecución remota del método *exposed_read* (método del *master*). El *master* responde con el id asignado a la respectiva partición del archivo y con el id del *chunkserver* donde está guardada tal partición. Después el *client* ejecuta el método *ask_chunk*, el cual se describe a continuación.
2. *ask_chunks:* Este método requiere el id de archivo dado por el *master*, así como el id, del *chunkserver* que contiene el archivo, para así, invocar de manera remota el método *exposed_read* del respectivo *chunkserver*. El *chunkserver* responde mandando el archivo solicitado.
3. *write_to_chunk:* este método particiona un archivo y cada partición la guarda en el respectivo *chunkserver* que el *master* le indique. Para ello, primero el *client* ejecuta de manera remota el método *exposed_check* del *master*, el cual responde sí el archivo ya existe en sus metadatos. En caso de que el archivo no exista, el *master* genera un id para cada partición del archivo, asigna de manera aleatoria qué partición del archivo debe guardarse en qué *chunkserver*, esta información es enviada al *client*. Para cada partición del archivo, el *client* contacta al *chunkserver* indicado por el *master* y ejecuta de manera remota el método *exposed_write* de cada *chunkserver*, para así escribir cada partición del archivo en el respectivo *chunkserver*.

II-D. Implementación gráfica

Para realizar la implementación gráfica se utilizó la librería *dash* de *Python*, la cual nos permite realizar *dashboards* dinámicos. Para ello, todos los métodos del *client*, así como los parámetros de configuración del *master* y de los *chunkservers* pueden ser accedidos mediante una interfaz gráfica, la cual se muestra en la figura 2.

Esta interfaz gráfica consta de los 4 botones (de izquierda a derecha):

1. *Read File:* este botón nos permite leer un archivo.
2. *Write File:* este botón nos permite escribir un archivo.
3. *Restart Server* este botón nos permite reiniciar el *master* server, así como configurar el número de *chunkservers* y el número de particiones por archivo.

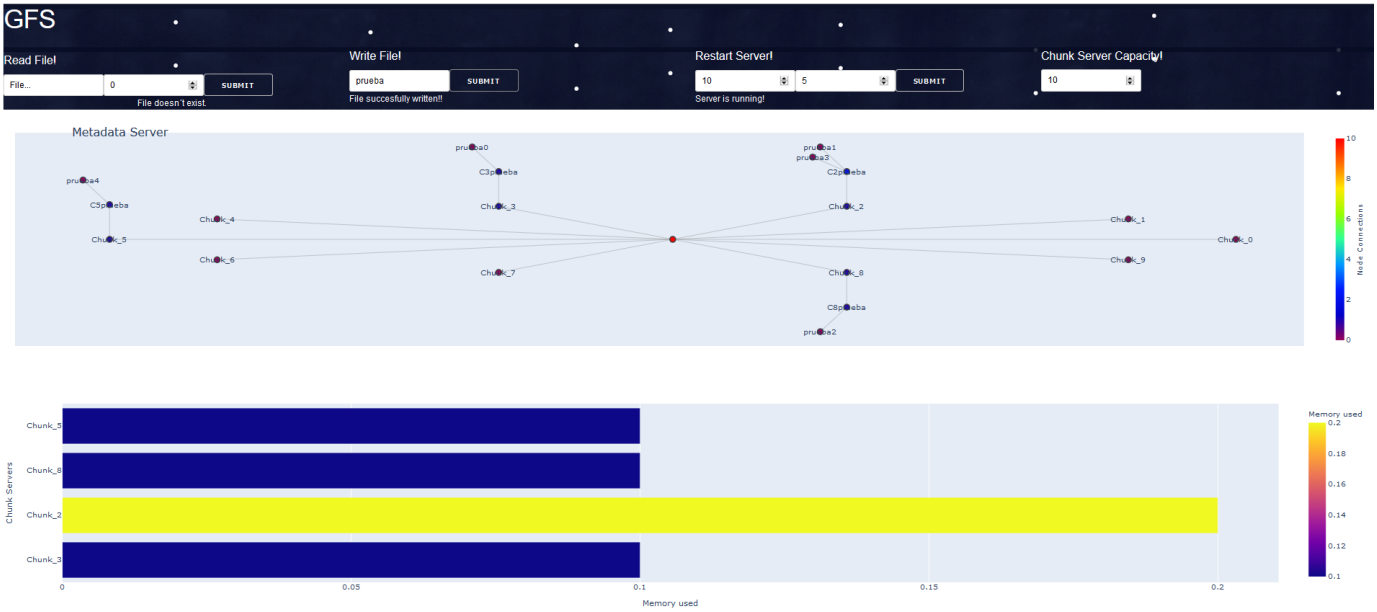

Figura 2. Interfaz gráfica, fuente propia.

4. *Chunk Server Capacity:* este botón permite configurar la capacidad de memoria de los *chunkservers*.

También es posible observar los metadatos representadso como un grafo. En el centro se presenta el nodo *master*, cada uno de los nodos hijos de este representa a un *chunkserver*, y cada hoja de un *chunkserver* representa una partición de algún archivo guardado en este *chunkserver*.

Por último se muestra un gráfico de barras, el cual representa la cantidad de memoria ocupada en cada *chunkserver*.

III. Conclusiones

En este proyecto se realizó una simulación de los metadatos generados por las operaciones de escritura del sistema de administración de archivos *Gogle File System*, para ello se implementó una interfaz gráfica la cual nos permite jugar con algunos parámetros de configuración, como lo son el número de *chunkservers*, la capacidad de memoria de cada *chunkserver* y el número de particiones por archivo. Tanto el número de particiones por archivo como el número de *chunkservers* permiten explorar qué tanta es la concentración y distribución de archivos entre los distintos *chunkservers*, en general esta debería ser uniforme pues el *master* selecciona de manera aleatoria a los *chunkservers*. En la misma línea, en [1] se asume que los *chunkservers* son infinitos, es decir, nunca se saturarán, pero para el caso en que se cuente con un conjunto limitado de recursos, esta implementación permite explorar cual es la saturación de estos ante distintas cargas de trabajo (tamaños y número de archivos), con lo cual podría ser interesante el desarrollo de nuevos protocolos de rebalanceo. Por ejemplo, un portocolo podría ser la asignación de una distribución de probabilidad uniforme ponderada por el porcentaje de uso de la memoria de cada *chunkserver*, de esta manera se le asignaría menor probabilidad de seleccionar a los *chunkservers* con menos memoria disponible. Por último, cabe señalar que el *Google File System* es un sistema de administración de archivos simple, pero poderoso, es decir, la idea de que en el nodo *master* estén guardados únicamente los metadatos, y que este sea un intermediario entre los *clients* y los *chunkservers* otorga una gran escabilidad para el sistema, ya que una gan parte de la latencia de las operaciones de escritura, lectura o actualización recaen en los *chunkservers*.

Referencias

1 Ghemawat, S., Gobioff, H., and Leung, S.-T., "The google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 29–43. [Online]. Available: https://doi.org/10.1145/945445.945450