Programación Distribuida
y
Tiempo Real

Práctica 4

AÑO 2022 – 2° Semestre

Referencias

1. https://www.geeksforgeeks.org/what-is-dfsdistributed-file-system/
2.

1) **Programar un agente para que periódicamente recorra una secuencia de computadoras y reporte al lugar de origen:**
NOTA1: Para almacenar el nombre, la memoria disponible y la carga de procesamiento de cada contenedor usamos una matriz. La misma se imprime cuando el agente da la vuelta completa por todos los contenedores. El contenedor MainContainer es quien la imprime.

```
Time of round 370 milliseconds
Container#              Free memory (bytes)     CPU time (ns)
Container0              248790624               23670462
Container1              248790624               25494144
Container2              248790624               26174164
Container3              247612248               21349507
```

Fig1. Captura de la matriz imprimida por consola

NOTA2: Para ejecutar el experimento hicimos un script bash el cual compila el programa del agente y levanta 4 contenedores junto con el Main Container. El script se llama “init.sh”.

   a) **El tiempo total del recorrido para recolectar la información.**
   Para medir el tiempo total de recorrido usamos el método currentTimeMillis de la clase System, el cual nos devuelve el tiempo actual en milisegundos.
   La marca de tiempo startTime la hacemos en el método Setup, el cual se ejecuta sólo en el último contenedor, ya que nosotros solo asignamos el agente al último contenedor. Luego de pasar por todos los contenedores hasta llegar al Main Container, se calcula el tiempo con la marca de tiempo actual endTime, o sea, el tiempo que tardó en dar la vuelta será endTime - startTime milisegundos.

   b) **La carga de procesamiento de cada una de ellas.**
   Para medir el tiempo de CPU consumido usamos la interfaz ThreadMXBean. La cual mediante el método *getCurrentThread-CpuTime()* nos devuelve el tiempo de CPU consumido en nanosegundos.

   c) **La cantidad de memoria total disponible.**
   Para obtener la cantidad de memoria disponible usamos la clase Runtime de Java, que cuenta con el método *freeMemory()*. Este método devuelve la cantidad de memoria bytes que tiene la JVM disponible para su uso.

   d) **Los nombres de las computadoras.**
   Los contenedores tienen el nombre “Container-X” donde X es el nro. del contenedor, el nro. del contenedor lo almacenamos en la matriz dicha.

Comente la relación entre este posible estado del sistema distribuido y el estado que se obtendría implementando el algoritmo de instantánea.

El estado que obtenemos en este caso es un estado parcial del sistema distribuido, o sea, no estaríamos obteniendo el estado global ya que solo registramos memoria libre, tiempo CPU consumido, etc. por cada proceso. Si aplicáramos el algoritmo de instantánea además de obtener el estado de cada proceso del sistema distribuido, obtendremos todos los mensajes recibidos por cada proceso hasta un determinado momento (el corte debe ser consistente).

2) **Programe un agente para que calcule la suma de todos los números almacenados en un archivo de una computadora que se le pasa como parámetro. Comente cómo se haría lo mismo con una aplicación cliente/servidor. Comente qué pasaría si hubiera otros sitios con archivos que deben ser procesados de manera similar.**

Se utiliza la misma lógica que en el ejercicio anterior, en donde el agente se traslada entre contenedores recopilando la información requerida para posteriormente presentarla en el contexto del contenedor “Main-Container”.

La principal diferencia radica en que se requiere como argumento del programa el nombre del archivo por el cual se leerá la información del mismo en cada uno de los contenedores que cuenten con dicho archivo.

```
java -cp lib/jade.jar:classes jade.Boot -gui -container -host localhost -agents
“mol:AgenteMovil(file.txt)”
```

Código para ejecutar el programa (ejemplo de pasaje de argumentos)

Como en el ejercicio anterior, la solución fue desarrollada en base al escenario donde se tiene como precondición que además del contenedor “Main-Container”, existen otros tres contenedores llamados (“Container-1”, “Container-2”, “Container-3”), que contarán con sus respectivos repositorios simulados con directorios cuyos nombres serán el mismo que el de sus contenedores. Por lo tanto, se desarrolló un script en bash que simula por nosotros este escenario, el script se llama igual al desarrollado en el punto anterior, “init.sh”.

Arquitectura cliente/servidor:

En una solución donde se requiera una arquitectura cliente/servidor. Será el cliente quien realice una petición sobre la información requerida hacia el servidor.

Se puede pensar en dos formas de llevar a cabo una solución, en ambos casos se requiere que el cliente realice una petición mandando como parámetro el nombre del archivo por el cual se quiera operar. En una de las soluciones, el servidor se encargará de sumar los dígitos contenidos en el archivo cuyo nombre es el recibido por parte del cliente en forma de argumento, y una vez finalizada la operación, envía el resultado al cliente. En cambio, en la otra solución

propuesta, el servidor devuelve al cliente directamente el archivo cuyo nombre es brindado como parámetro para que sea el cliente el encargado de sumar los dígitos almacenados en el archivo recibidos para obtener finalmente el resultado esperado.

En el caso donde se encuentren otros sitios con archivos que deben ser procesados de igual forma que la planteada en el ejercicio, el agente debe realizar el proceso de migración, recopilación de información y entrega de resultados tantas veces como sea necesario para cumplir con la operación.

3) **Defina e implemente con agentes un sistema de archivos distribuido similar al de las prácticas anteriores.**
NOTA: Antes de responder los incisos explicaremos el concepto de sistema de archivos distribuido, al cual nos referiremos con el acrónimo DFS.
Un sistema de archivos distribuido es aquel sistema de archivos en el cual los archivos se distribuyen en varias computadoras. Si bien el sistema de archivos está disperso geográficamente el cliente ve un único sistema de archivos.
Un DFS muy conocido es Hadoop DFS el cual se usa para procesamiento Big Data, ya que permite almacenar archivos de gran longitud en partes, las cuales se distribuyen en los distintos nodos.

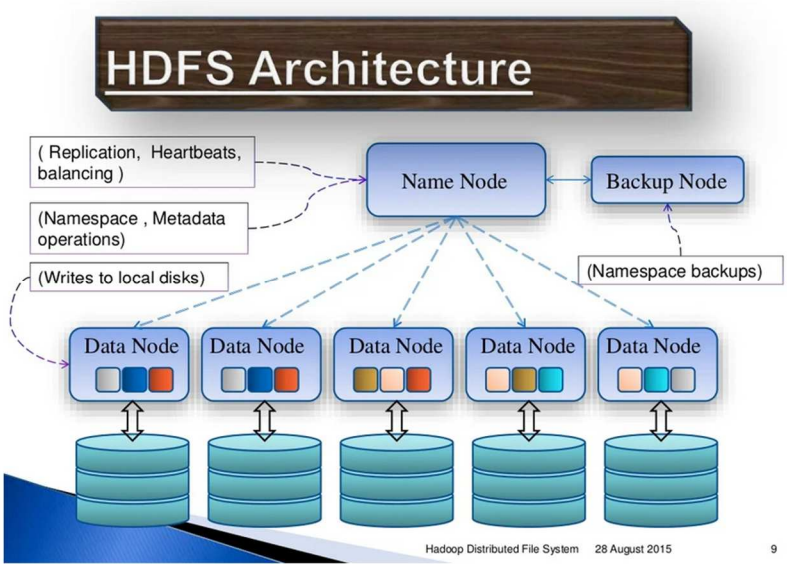


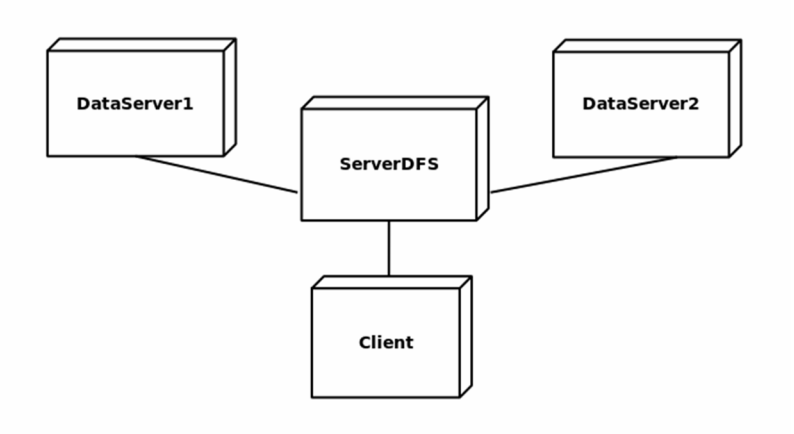
Fig.3.1 Ejemplo de DFS - HDFS

a) **Debería tener como mínimo la misma funcionalidad, es decir las operaciones (definiciones copiadas aquí de la práctica anterior):**
Leer: dado un nombre de archivo, una posición y una cantidad de bytes a leer, retorna 1) la cantidad de bytes del archivo pedida a partir de la posición dada o en caso de haber menos bytes, se retornan los bytes que haya y 2) la cantidad de bytes que efectivamente se retornan leídos.
Escribir: dado un nombre de archivo, una cantidad de bytes determinada, y un buffer a partir del cual están los datos, se

escriben los datos en el archivo dado. Si el archivo existe, los datos se agregan al final, si el archivo no existe, se crea y se le escriben los datos. En todos los casos se retorna la cantidad de bytes escritos.

b) **Implemente un agente que copie un archivo de otro sitio del sistema distribuido en el sistema de archivos local y genere una copia del mismo archivo en el sitio donde está originalmente. Compare esta solución con la de los sistemas cliente/servidor de las prácticas anteriores.**

Para resolver lo pedido en los incisos a y b de este ejercicio hemos implementado un sistema de DFS con Jade. En el cual modelamos los contenedores dependiendo de su rol en el sistema:



Rol de cada contenedor:

- El DataServer es aquel que almacena la información del DFS, pueden ser varios, distribuyendo el almacenamiento por distintas computadoras, tal cual lo hacen los DFS reales.
- El ServerDFS es aquel servidor que atiende las peticiones del cliente respecto al sistema de archivos, es el que recibe el pedido de lectura o escritura de un archivo.
- El Client es aquel que realiza la petición a efectuar (lectura o escritura), es el que ejecuta por primera vez el agente.

Dinámica de lectura:

1. Si la operación es una lectura el código en el contenedor Client migrará al contenedor ServerDFS.
2. Una vez en el contenedor ServerDFS éste determinará en dónde se ubica el archivo que se busca leer, esto lo sabe mediante un hashmap que para cada nombre de archivo (key) contiene el nombre del contenedor DataServer en donde se almacena (value). El agente entonces migra al DataServer que contiene el archivo.
   Si el archivo no existe entonces informa y vuelve al cliente.

3. Una vez en el contenedor DataServer adecuado empieza la transferencia, se leen X cantidad de bytes y luego se migra al contenedor cliente para que este los escriba. El proceso se repite hasta que se terminan de leer todos los bytes involucrados, el archivo resultante se almacena en la carpeta ClientLocalSpace, la cual representa el storage local del cliente.

Dinámica de escritura:

1. Si la operación es una escritura el código en el contenedor Client migrará al contenedor ServerDFS.
2. Una vez en el contenedor ServerDFS éste determinará si el archivo no existe, si no existe elige un contenedor al azar para almacenar el archivo. Una vez elegido el contenedor DataServer se migra al cliente para empezar la transferencia de datos.

   Si el archivo existe entonces se procede normalmente, al archivo existente se le concatena el contenido del archivo a escribir.

3. Una vez en el contenedor Client se comienza a leer el archivo, se leen X cantidad de bytes y luego se migra al DataServer elegido para que este los escriba, el proceso se repite hasta que el archivo se haya transferido por completo. Al final el archivo se agrega al hashmap del ServerDFS, el cual se persiste además en un archivo de texto llamado hashmap.txt.

Para probar este experimento hemos construido un script bash, llamado “dfs.sh” el cual recibe dos parámetros, la operación a realizar y el nombre del archivo a leer o escribir en el DFS. Este se ocupa de chequear los parámetros y de levantar todos los contenedores.

Por ejemplo:

Lectura de un archivo del DFS

```
dfs -r sweet.mp3
```

Escritura de un nuevo archivo en el DFS

```
dfs -w nuevo_archivo.txt
```

Adicionalmente para el caso que pide en el inciso 3b, implementamos una variante en la operación de lectura, que además de efectuar la lectura de un archivo desde el DFS al Cliente, crea además una copia en el DataServer donde se almacena el archivo. Esta copia se creará con el prefijo “copy-” seguida del nombre del archivo original. Esta copia se agrega al hashmap del DFS y se persiste en el archivo hashmap.txt.

```
dfs -r sweet.mp3 -c
```

Comparación con sistemas cliente/servidor previamente estudiados en temas anteriores:

Sistemas cliente/servidor	Sistemas de archivos distribuidos
En la comunicación de esta arquitectura, alcanza con el envío de los datos solicitados, por lo que se puede considerar una transferencia "liviana" ya que no requiere mantener ningún contexto de las variables.	En la comunicación se envía todo el proceso que ejecuta el agente, para poder ejecutarse en el entorno que realiza la petición, por lo que puede considerarse una transferencia "pesada" ya que debe transferirse además el contexto de las variables.
El manejo de las bases de datos se puede ver más sencillo al mantenerla centralizada.	Tanto el manejo como la conexión de las bases de datos en sistemas distribuidos se torna complejo
Toda la red está construida alrededor del servidor, por lo que, si no posee un buen desempeño o directamente deja de funcionar, perjudica a toda la infraestructura.	El sistema de archivos distribuido proporciona disponibilidad de los datos incluso si el servidor o el disco fallan, permitiendo disponibilidad en caso de falla de enlaces, nodos, etc.
Se cuenta con un único servidor que responde a las peticiones que le son realizadas por los procesos clientes.	Para el cliente es transparente la comunicación con el servidor, este desconoce la infraestructura o la ubicación de cada uno de los servidores de archivos
Se puede producir una congestión del tráfico cuando una gran cantidad de clientes envían peticiones simultáneas al mismo servidor.	La distribución de recursos en los diferentes nodos del servidor le permite mantener la comunicación a múltiples peticiones.