Trabajo práctico n. 1

Concurrencia y comunicaciones: Backup server

7574 - Sistemas Distribuidos I

Docentes

- Pablo D. Roca
- Ezequiel Torres Feyuk
- Guido Albarello

Alumno

- Rozanec, Matías 97404 mrozanec@fi.uba.ar

2C 2020
Facultad de Ingeniería - Universidad de Buenos Aires

Índice

Introducción

Nomenclatura

En primer lugar, se aclara la nomenclatura usada en el informe. El **sistema central** que gestiona los backups es llamado **server**. Los comandos de registro, desregistro y query de backups hacia el exterior del sistema son gestionados por el **Admin**. Por último, los sistemas que deben realizar el backup de un path dado son llamados **clientes**.

Esta aclaración se hace porque en las clases de consulta hubo bastante confusión porque cada uno entendía y llamaba estas partes de forma levemente distinta.

Cabe aclarar que se ha tomado la escala de tiempo en segundos en lugar de minutos, con el fin de hacer más ameno el testeo y presentación del trabajo.

Ejecución

Toda la ejecución del sistema puede ser controlada mediante targets que están descritos en el Makefile. Se listan a continuación los targets “base”.

Makefile Targets

En primer lugar hay que correr el comando `make docker-compose-up`.

Con `make docker-compose-logs` se pueden ver los logs del sistema. Esto permite inspeccionar lo que anda pasando en el sistema y asegurarse que el funcionamiento sea el correcto. Notar que todos los procesos del sistema loguean a la misma terminal.

El comando `make new-client NODE=sample_node_name PORT=xxxxx` lanzará un nuevo cliente que se conectará a un mismo volumen de Docker y a la misma red (network) ya creada en los comandos previos. NODE y PORT serán pasadas como variables de entorno a Docker. El valor de NODE es utilizado para

- nombrar al contenedor, y poder hacer así un seguimiento de los contenedores en ejecución.
- mantener todos los backups temporalmente creados, hasta que se manden al servidor central, en un directorio propio dentro del sistema, sin que se mezcle con los de otros clientes.

El valor de PORT es el puerto que debe exponer el cliente para que el sistema se pueda conectar a él para solicitar los backups.

Content creators

Los *content creators* son pequeños programas que se conectan al volumen de Docker y escriben archivos, a razón de una línea por segundo, en el path indicado. Esto sirve para tener automatizada la escritura en vivo de los archivos. Además, su diseño apartado del

resto de las funcionalidades permite un control muy fino sobre la ejecución y pruebas que se quieran realizar.

Hay 3 comandos relacionados con los content creators, a saber:

- `make new-content-creator`: crea un solo creador de contenidos con el nombre name1 que escribirá el archivo a/file1.txt
- `make init-content-creators`: crea 10 content creators que escribirán los siguientes archivos:

```
.
├── a
│   ├── a
│   │   ├── a
│   │   │   └── file1.txt
│   │   └── file1.txt
│   ├── b
│   │   └── file1.txt
│   └── file1.txt
├── b
│   ├── a
│   │   └── file1.txt
│   ├── b
│   │   ├── b
│   │   │   └── file1.txt
│   │   └── file1.txt
│   └── file1.txt
└── c
    ├── b
    │   └── file1.txt
    └── file1.txt
```

- `make kill-content-creators`: detiene la ejecución de los 10 content creators.

Registro, desregistro de nodos

Para el registro y desregistro de nodos se armaron los comandos:

- `make admin-register-node NODE=sample_node_name PORT=xxxxx NODE_PATH=sample_path_to_backup FREQ=xx`
- `make admin-unregister-node NODE=sample_node_name NODE_PATH=sample_path_to_backup`

Para poder registrar varios nodos de prueba a la vez, se creó el comando `make init-nodes` que registra 8 nodos.

Backup Query from Admin

Para realizar la solicitud de algún backup, hay que ejecutar el target

```
make admin-query-node NODE=sample_node PATH=sample_path
SAVE_TO_FILE=requested_backup.tgz
```

que guarda el último backup que hizo el nodo sample_node de sample_path en el archivo requested_backup.tgz.

Restricciones

Path

El path de interés a backupear no debe contener números ni el caracter guiones medios (“-”). Esto se debe a que cada archivo se guarda con un nombre que sigue el siguiente formato: `[node]-[path][num].tgz`. Además:

- las barras (“/”) del path son reemplazadas por guiones al almacenar el archivo comprimido. Si bien esto se podría evitar replicando la estructura del directorio hasta el path deseado en el servidor, esto trae consigo una estructura de directorios más compleja que mantener. Si bien probablemente para un sistema que deba ser desarrollado para salir a producción valga la pena esta inversión, para la entrega se evaluó como un trade-off razonable.
- cada archivo comprimido se guarda con un número de serie incremental, y el próximo número de serie se saca del número que está en el último path. Debido a temas de implementación, tener números extra en el nombre no está soportado. Como en el ítem anterior, esto podría ser optimizado de forma relativamente fácil, pero a efectos de la entrega se prefirió aceptar esta restricción.

Demo

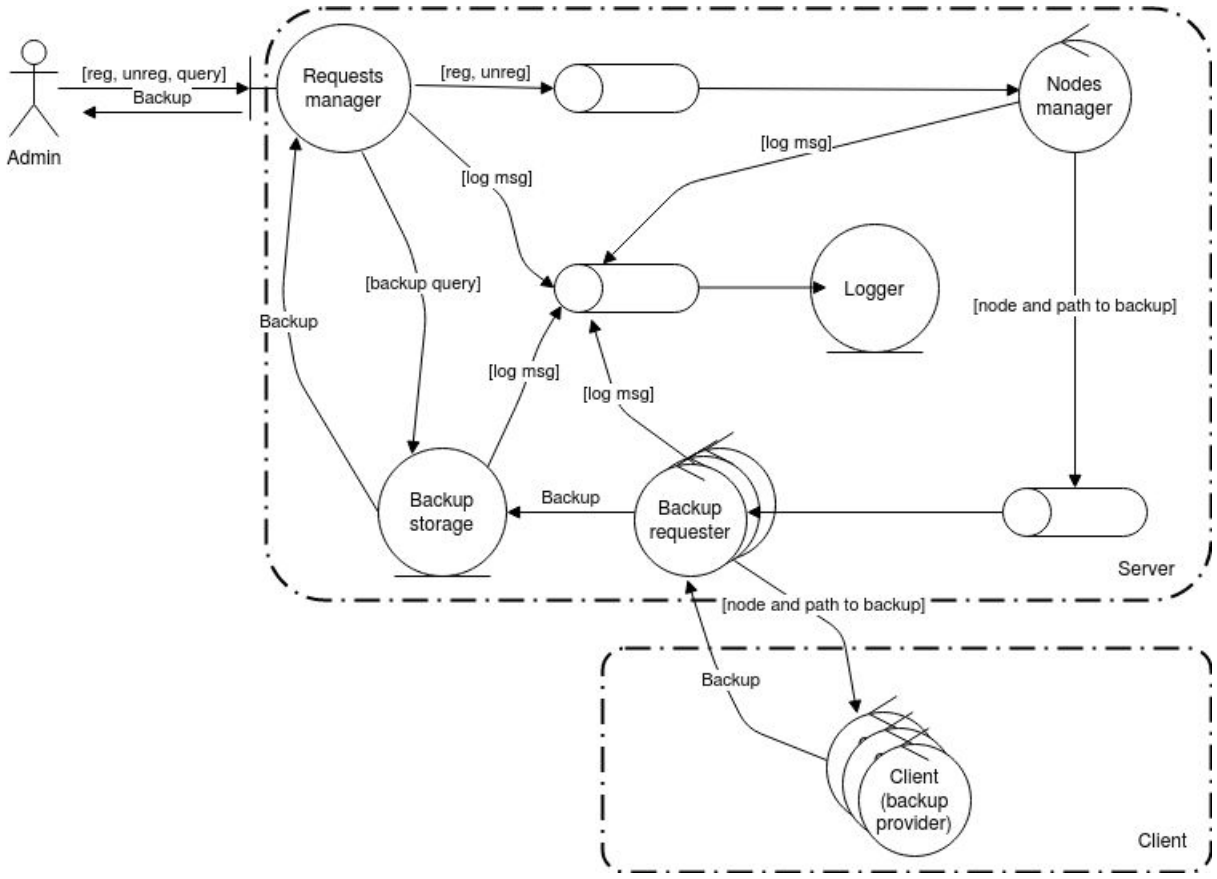
La demo se divide en dos partes: en primer lugar se hace una corrida con un solo cliente, de forma que se pueda hacer un seguimiento claro de los logs, comprobando que todo esté bien y familiarizarse con los outputs esperados. Una vez quede comprobado el funcionamiento con un cliente, se pasa a la segunda parte donde se corre el sistema con múltiples clientes.

En ambos casos se podrán inspeccionar los mensajes que comprueben cómo el sistema cobra vida. Por otro lado, se podrá acceder a los volumes de Docker para inspeccionar la creación de los distintos archivos, y finalmente también hacer retrieve de algún comprimido de interés.

Diagramas

A continuación se presentan los diagramas que permiten entender el sistema desarrollado.

Robustez



En el diagrama adjunto se puede ver que toda la interacción con el sistema se da por medio del Admin que ejecuta comandos de registro, desregistro y query de backups.

Notar que el Server (Requests Manager), Nodes manager, Backup manager y Logger son todos procesos independientes que parten de un mismo main (un main que lanza procesos independientes, no threads). Esto permite que los mismos se intercomuniquen mediante las Queues de IPC de Python, por lo que las colas del diagrama se corresponden totalmente con colas de IPC. Esto es especialmente importante en la conexión entre el Nodes manager y el Backup requester: El Nodes manager es quien controla cuándo hay que pedirle backups a qué nodos, y le pasa dicho request al Backup requester. El Backup requester se comunica luego con cada uno de los clientes necesarios. Esta parte es el cuello de botella en el sistema, porque implica transferencia de archivos, por lo que es donde hay interés de tener flexibilidad para crecer. La presencia de una cola allí hace que pueda haber múltiples consumidores sin cambios en la arquitectura propuesta.

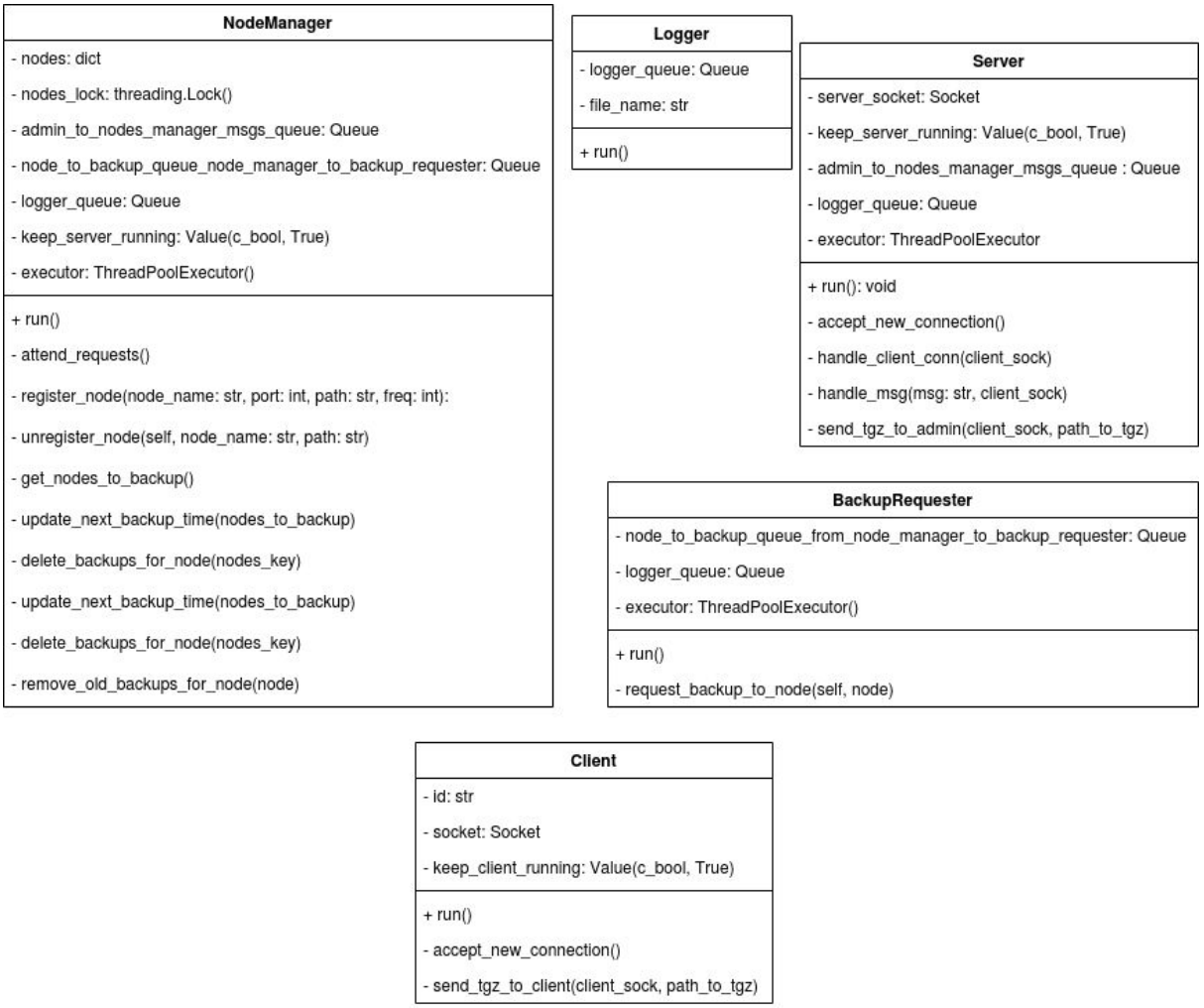
El backup requester corre dos threads:

- por un lado el ya mencionado chron para pedir los backups necesarios. Cada un segundo se chequea qué nodos deben hacer backup y se envía esta info a la cola hacia el backup requester
- por otro lado, se reciben mensajes de registro/desregistro de nodos. Como esta información es accedida por ambos threads, se hizo uso de un Lock para evitar race conditions.

Todos los procesos pueden enviar mensajes de Log al Logger mediante un canal destinado a tal fin.

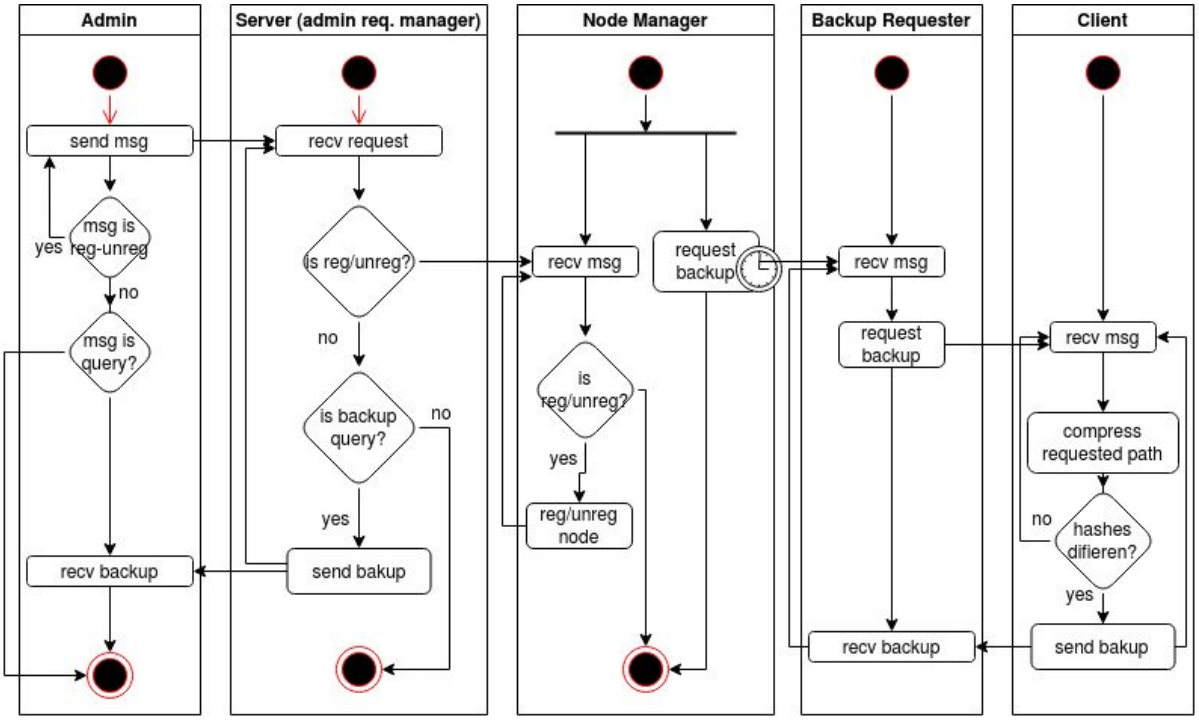
El backup storage es directamente el almacenamiento en disco al que todos los procesos tienen acceso. No hay una entidad que interceda entre el disco y el resto de los procesos. La ventaja es que el acceso es más directo.

Clase



Como se puede observar, no hubo composiciones entre las distintas clases. Cada entidad definida juega más bien un rol bastante independiente, donde la relación entre las distintas partes se da exclusivamente mediante mensajes. Si bien a la hora de realizar el diagrama costó aceptar que esto pueda ser correcto, lo cierto es que es la primera vez que se trabajó con algo de naturaleza distribuida, y si bien seguramente se podría mejorar el aspecto de cada servicio particular, no es del todo raro que haya menos conexiones que las que se acostumbra ver en otras materias o contextos.

Actividades



En este diagrama se puede observar el flujo de los mensajes. En primer lugar cabe notar que el admin puede enviar 4 tipos de mensajes:

- registro
- desregistro
- query
- shutdown

Solamente espera respuesta en el caso del query.

El server solamente responde de forma directa en caso de query; de lo contrario delega la tarea o, en el caso de shutdown, pasa el mensaje para que el resto de los procesos también se apaguen.

El node manager tiene dos threads: uno es el chron que chequea cada segundo qué backups hay que pedir; el otro thread es el que mantiene actualizado el registro de nodos.

El Backup requester solamente espera indicación para pedir backups, y a medida que vayan entrando los pedidos los va pidiendo al cliente que los proporciona solamente en caso que el checksum no coincida con el del que el sistema ya tiene almacenado, en caso que tenga alguno.

Dificultades encotradas

md5 checksum sobre un tar

A la hora de quere realizar el checksum de un tar file, todo anduvo bien hasta que los hashes no deberían haber cambiado y sin embargo esa expectativa no se vio reflejada en los resultados. Finalmente, resultó en que el hash sobre el tar se estaba calculando mal: no se calculó sobre cada uno de los archivos que integraban el tar, sino sobre el tar entero.

Conclusión

El presente trabajo práctico me obligó a enfrentarme a nuevos paradigmas y formas de pensar y resolver los problemas. Siendo la primera vez que trabajo con una arquitectura “plenamente distribuida” que deba ser escalable, muchos conocimientos previos tuvieron que adaptarse de alguna forma a esto.

Si bien finalmente logré implementar una solución que cumple con los requerimientos solicitados, siento que hay mucho lugar para mejorar. ¡Espero que esas potenciales mejoras se vean plasmadas en los siguientes trabajos prácticos!