Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Licenciatura en Ciencias de la Computación
Sistemas Operativos I

Sistema de archivos distribuído

Alumnos:
BORRERO, Paula (P-????)
IVALDI, Ángela (I-????)
MISTA, Agustín (M-6105/1)

Docentes:
MACHI, Guido
GRINBLAT, Guillermo
DIAZ, José Luis

4 de Julio de 2016

Introducción

Un servidor de archivos distribuido es un componente de software que le ofrece al usuario final las operaciones necesarias para trabajar con un sistema de archivos virtual, aparentemente centralizado, donde todos los archivos parecen estar en una misma ubicación, cuando en realidad es probable que los mismos estén dispersos en varias unidades de disco, o más aun, en varias computadoras.

En éste informe analizaremos las implementaciones tanto en **C** como en **Erlang** de un servidor de archivos distribuido simple, esto incluye profundizar sobre algunas cuestiones de diseño tales como comunicación entre hilos, concurrencia y performance.

Implementación con Posix Threads

Usamos POSIX Messages Queues para la comunicación entre los distintos nodos.

Estructuras de datos usadas:

- Session: representa las sesiones. Contiene datos que relacionan un cliente con su respectivo worker, seleccionado de forma aleatoria por el dispatcher cuando se conecta.
- Request: se usa para modelizar los pedidos a workers. Si el campo external (int) es distinto de 0, el pedido proviene del handler, en otro caso, proviene de otro worker.
- Reply: representa las respuestas del worker al handler. Contiene dos campos uno de tipo Error (representa errores al procesar un comando) si no hay errores es NONE, y el otro campo sirve para agregar informacion extra sobre el resultado.
- File: modela cada archivo.
- Worker_Info: permite guardar datos de cada worker, entre ellos: sus archivos, identificador, cola de mensajes y pool de descriptores de archivos.

Módulos

- Server: inicializa la conexión TCP, el dispatcher y los workers.
- Dispatcher: espera por nuevas conexiones, y cuando ocurre alguna, se ocupa de crear una sesion (session) y lanzar un handler.
- ClientHandler: se encarga de parsear lo que recibe de cada cliente, lo convierte en un external request y se lo manda al worker correspondiente. Cuando recibe una respuesta(reply) de algun worker, la procesa y muestra el resultado a cliente (de manera que lo entienda).
- Worker: recibe del handler o de otros workers diferentes peticiones, las cuales intenta llevar a cabo. Opera sobre sus archivos cuando es necesario. Para optimizar las operaciones, no es necesario que una peticion pase por todos los workers. El primero que pueda llevarla a cabo, lo hará y contestará al handler correspondiente.

Para mostrar los mensajes del servidor se usaron banderas de compilación condicional. Con la bandera DEBUG, se muestran mensajes de estado del servidor. Con DEBUG_REQUEST podremos ver también la comunicación entre workers.

Implementación en Erlang

En este caso los workers forman un anillo similar a la versión de C. La diferencia es que cuando un handler manda una petición a su worker designado sólo le responde el mismo (no hay optimizacion). Si se resuelve la petición antes de llegar al origen (exitósamente o con error) se propaga la respuesta hasta que llegue al origen. Si se da toda la vuelta sin resolver la petición se retorna error. Cada worker conserva sus archivos, su pool de file descriptors y el pid del siguiente worker. Cada uno recibe mensajes de la forma Req, Pid, Count, donde Req representa a los comandos y argumentos, el Pid del cliente solicitante y un contador de saltos, que al llegar a ?N_WORKERS indica que se dió toda la vuelta. El handler se divide en dos funciones, una cuando el cliente todavía no hizo CON handle_client y otra para recibir los comandos una vez que está identificado.
Cada archivo es representado mediante una tupla de la forma {Name, Fd, Opener, Cursor, Size, Content} y el conjunto de archivos de cada worker es una lista de tuplas de ese tipo.

Característica adicional

Implementamos en Erlang el punto adicional de mensajería tolerante a fallas (con time-out). Luego de 300 milisegundos se lanza el error: ERROR 62 ETIME.