

Universidad Simón Bolívar

Taller de Sistemas de Operación II

Prof. Yudith Cardinale

Abr/Jul 2011

Proyecto II

Introducción:

Imaginemos una región donde hay una buena cantidad de computadores disponibles pero con muchos problemas en el suministro eléctrico. En estas circunstancias, ejecutar una aplicación puede ser un largo proceso pues las frecuentes caídas de tensión cortan las ejecuciones. Dado que es del interés de todos los nodos poder ejecutar sus aplicaciones, se propuso un esquema de ejecución remota, tolerante a fallas, donde los clientes colocan las aplicaciones que desean ejecutar en varios nodos que juegan el rol de servidores de ejecución, con la idea de tener mayor probabilidad de una ejecución exitosa. El primer servidor que termine la ejecución notifica al cliente y éste enviará un mensaje a los otros servidores solicitando la detención del proceso. Adicionalmente, los resultados de cada ejecución (archivos en disco) deben guardarse también en varios servidores para que puedan ser recuperados en cualquier momento por todos los nodos que tengan interés en esos datos. En consecuencia el esquema propone una arquitectura tolerante a fallas tanto para el uso del CPU como para los archivos resultado de ejecuciones remotas.

Los programas deben estar desarrollados en JAVA ya que el código que viaja es un **.class** lo que implica que debe someterse a las restricciones impuestas por el **security manager** de los JVM de los servidores. Cada aplicación está obligada a respetar las condiciones que imponen los archivos de permiso que prohíben o permiten:

- **Acceder a los recursos locales (disco, unidad de DVD, etc)**
- **Ejecutar programas localmente a partir de la aplicación descargada inicialmente**
- **Conectarse a un servidor diferente al nodo de donde proviene la aplicación, etc.**

Además una de las condiciones impuestas por todos los administradores de los servidores es que se realice autenticación de los nodos mediante certificados digitales y cifrado de los canales.

Cuando un cliente solicite un determinado archivo, se le entregará la última versión de ese archivo. Para esto es necesario asignarle a cada archivo un *timestamp* y ante la solicitud, retornar la versión con mayor *timestamp*. Esto implica que los relojes de los servidores deben estar sincronizados.

La actualización de la información de los datos de cada servidor y su estado se debe realizar periódicamente.

Especificaciones Técnicas:

- Los servidores deberán mantener en todo momento una lista con la información actualizada de los demás servidores: nombre de la máquina, nombres de los archivos que almacena con su versión y cualquier otra información que consideren útil.
- La comunicación de control entre los servidores se realizará a través de un `socket` conectado a una dirección `multicast`, a la cual se suscribirán los servidores al momento de levantarse. Los mensajes que se envían a esta dirección pueden ser `multicast`, de importancia para todos, o `unicast`, dirigidos a un servidor específicamente.
- **La comunicación correspondiente a los servicios que se proveen al cliente, se realizará entre los servidores a través de Java RMI seguro. La certificación digital debe ser en ambos sentidos; de servidor a cliente y de cliente a servidor (la herramienta para crear certificados en Java se llama `keytool`). Por último los `.class` deben estar firmados de preferencia dentro de un `.jar` (en java se realiza con `jarsigner` desde la consola o el método `sign` desde el programa)**
- El cliente, deberá presentar una interfaz gráfica muy sencilla al usuario a través de la cual pueda especificar el nombre del `.class` a ejecutar y **obtener el tipo de permisos que requiere. Para ello existen herramientas que se ejecutan con un `security manager` sin restricciones que monitorea la ejecución y define un perfil de lo que solicita esa aplicación. Este perfil es el que debe ser aceptado por el `security manager` receptor. Esto permite constatar contra el servidor que recibe la aplicación para que éste pueda saber si es posible ejecutar la aplicación.**
- Cada servidor podrá ejecutar sólo una aplicación a la vez.
- Cada servidor es seleccionado para ejecutar una aplicación en función a la carga y bajo el principio de que se requieren al menos 3 servidores para ejecución. El número puede aumentar en función al tiempo promedio en que los servidores estén caídos por falta de suministro eléctrico. Es decir, hasta el 50% del tiempo promedio caídos se seleccionan al menos 3 servidores. Por cada 10% de aumento se asigna un servidor más hasta un máximo de 8 servidores. En consecuencia cada servidor debe monitorear el tiempo en que otros servidores estén caídos (cuando, por supuesto, él esté levantado). Es claro que este número

puede desvirtuarse porque no existe un superservidor siempre levantado que cuente el tiempo pero se hará lo mejor posible ...

- Es claro que debe haber un proceso de sincronización de relojes entre los servidores para manejar las versiones de las ejecuciones y archivos de cada aplicación. Cuando se vaya a guardar un archivo resultado de una ejecución, es necesario que sea guardado en al menos la mitad + 1 servidores.
- El cliente que ejecuta una aplicación debe ser informado de cuáles servidores fueron seleccionados para ejecutar la aplicación. De igual modo debe ser informado de cuáles servidores respaldan los archivos resultados de las ejecuciones. Si algún nodo desea información de algún archivo, se lo solicita al nodo propietario.
- Uds. deben definir la estructura de datos y el protocolo necesario para cumplir con las funcionalidades requeridas. Esto da cierta libertad a la hora de definir el tipo de interacción que tendrán los servidores entre ellos.

NOTA: Lo resaltado en negrita es opcional y tendrá un puntaje adicional al 20% que representa el proyecto.

Entrega:

La entrega será la última semana del trimestre y deben entregar un informe con, al menos, las consideraciones de manejo de sincronización y seguridad más relevantes, protocolo que utilizaron e interfaz de interacción entre los distintos nodos (cliente y servidores).