Programación Declarativa Avanzada

Curso 2011/201

Juan Rodríguez Hortalá

Modelos de práctica para Erlang

1. Chat

Implementa un sistema de chat sencillo con interfaz de consola. Se basará en un proceso servidor central en el que se registrarán (login) los usuarios asociando un nick a un pid. Cada usuario se representará con un proceso servidor usuario al que se le mandarán órdenes de consola, usando funciones que encapsulen el formato de los mensajes, para iniciar el servidor de usuario y hacer login en el servidor central, mandar un mensaje, hacer logout … El proceso servidor central y cada uno de los procesos usuarios se ejecutará en un nodo diferente, de forma que cada cliente tenga su propia consola. Se puede emplear un nombre predefinido para el nodo central y tener todos los nodos dentro del mismo host.

- Cada vez que un usuario mande un mensaje el servidor central este lo reenviará a todos los demás usuarios, que escribirán en mensaje en su consola con un formato [“nick - <hora>] <mensaje>” o similar.
- El servidor central también mandará mensajes a todos los usuarios cuando un usuario haga login o logout.
- También se implementará un comando para pedir al servidor central la lista de nicks de usuarios registrados, que se escribirá en la consola.

Se valorará, aunque no es imprescindible:

- Que se utilice supervisión, por ejemplo para que cuando el proceso de un usuario se caiga entonces se haga logout del usuario en el chat, se envíe un mensaje informativo a los demás usuarios, y se borren todos sus mensajes pendientes.
- Que se permita establecer sesiones de chat privadas entre un grupo de clientes, que pueden acceder o rechazar la invitación, así como abandonar la sesión privada, durante las cuales sólo recibirán y enviarán mensajes a los clientes de la sesión privada.

2. Servidor RPC

Implementa un servidor RPC (remote procedure call) como un servidor Erlang. El servidor permitirá:

- Registrar funciones asociándolas a un átomo que las identifica.
- Pedir que se ejecute una función registrada anteriormente, proporcionando un átomo que identifica a la función, una lista de argumentos y un timeout para la ejecución.
- Encapsular el formato de los mensajes que maneja el servidor mediante funciones `init/0`, `register_fun/2`, `call_fun/3`, `unregister_fun/1`, ...

Se valorará, aunque no es imprescindible:

- Que se utilice supervisión, para matar al proceso que esté ejecutando una llamada concreta en caso de que se de el timeout.
- Que se proporcionen funciones adicionales para facilitar el ejecutar el servidor RPC en un nodo distinto de los procesos clientes.

3. Chat P2P

Variante del problema anterior en el que no hay un nodo central sino que todos los servidores usuarios, que ahora llamaremos pares, tienen la lista del resto de pares. En este caso es más natural identificar a los pares por el nombre del nodo en vez de por el pid, usando el mismo nombre de proceso registrado para todos los procesos usuario par en cada nodo. Se manejarán dos listas:

- Listas de nodos a los que nos hemos conectado alguna vez: al iniciarse un par se intentará conectar a todos los nodos de esta lista.
- Lista de nodos conectados: los nodos de la lista anterior a los que estamos conectados actualmente.

La lista de nodos a los que nos hemos conectado alguna vez se guardará en un archivo al salir del chat. Al conectarse a cualquier nodo de la red el nodo mandará a los demás nodos conectados el nombre del nuevo nodo para que actualicen sus listas. Por lo demás el chat funcionará de la misma manera.

Se valorará, aunque no es imprescindible:

- Que se utilice supervisión
- Que se utilicen otros modelos de distribución de los mensajes: por ejemplo en vez de que todos los pares tengan la lista de los nodos de los demás pares podríamos hacer que sólo tengan algunos, y que al mandarse un mensaje este se distribuya transitivamente hasta un cierto nivel par-de-par. Esto podría implicar que los mensajes no siempre lleguen a todos los nodos, pero se considerará aceptable (basándose en la idea de recibir mensajes de nodos “amigos” de “amigos” sólo hasta cierta distancia). En cualquier caso hay que tener cuidado con que no haya ciclos infinitos de reenvío de mensajes

4. MapReduce en Erlang

Implementa en Erlang el esquema algorítmico MapReduce (http://en.wikipedia.org/wiki/MapReduce) como una función de orden superior, y aplícalo a resolver algún problema sencillo, el ejemplo típico es contar el número de apariciones de cada palabra en un carpeta que contiene varios archivos de texto.

5. Sistema de sincronización de datos

Implementa un servidor que sincronice el contenido de las carpetas de distintas máquinas. En una primera versión simple sólo sincronizaremos los archivos de texto que se encuentren en una carpeta. Se basará en un proceso servidor que tendrá registrados varios procesos que quieren sincronizarse, que indicarán la carpeta que contiene los archivos de texto a sincronizar. Además de hacer login y logout del servidor, los clientes podrán solicitar una sincronización, lo que resultará en que el servidor comprobará la antigüedad de los archivos de cada cliente y mandará mensajes a los clientes que haga falta para que actualicen sus archivos, añadiendo los posibles archivos nuevos que haya creado el cliente que solicitó la sincronización
Se recomienda que en el estado del servidor se guarden los datos acerca de los archivos que tiene cada cliente.
Se valorará, aunque no es imprescindible:

- Que se implemente de forma distribuida.
- Que se sincronicen carpetas recursivamente y/o archivos de cuaquier tipo.
- Que se haga una versión P2P sin un servidor central que coordine la sincronización.

6. Sistema de hosting de documentos P2P

Implementa un sistema de almacenamiento distribuido de documentos de texto basado en un servidor central que actúa de índice, y que contiene una lista de documentos junto con los nodos que los están guardando. Cuando un nodo se una a la red indicará la carpeta que contiene los documentos de texto que está hosteando. Se supone que en la red no puede haber dos documentos con el mismo nombre.
Implementa operaciones para:

- Buscar un documento: se localiza el nodo que almacena el documento y se escribe en la consola el contenido del documento.
- Actualizar un documento: se proporciona un string que será la nueva versión del documento, y se envía la nueva versión al nodo o nodos que tengan copia del documento para que la actualicen, y se añade el nodo modificador a la lista de nodos con copia del documento.

Se valorará, aunque no es imprescindible:

- Que se almacenen recursivamente carpetas que contengan archivos de texto, que ahora se identificaran por la ruta en la carpeta.
- Que se haga una versión sin un servidor central que actúe de índice, teniendo el índice distribuido entre los pares, de forma similar al chat P2P del ejemplo 3.

7. Chat persistente: clon de twitter

Un chat con almacenamiento de los mensajes antiguos puede verse como una versión muy simplificada de twitter. Extiende el chat con almacenamiento de mensajes en tablas DETS y permitiendo búsqueda de mensajes según palabras clave.
Este clon de twitter será necesariamente limitado porque las tablas DETS tienen un límite de capacidad de 2GB. Se valorará una versión P2P.