Práctica Final 1: MapReduce en Erlang

Se trata de simular una versión del modelo de procesamiento distribuido MapReduce, adaptado a las características de Erlang. Este modelo funciona en sistemas distribuidos en los que la información está distribuida entre nodos, que representaremos como procesos. A continuación describimos los procesos que participan y los mensajes que envían/reciben

Proceso Master

Para construir el sistema de nodos usaremos un proceso inicial, el máster, que inicializaremos con dos parámetros:

- ***Info***: una lista de valores.
- **N**: Número de nodos a crear.

Con esta información el máster:

1. Repartirá la lista ***Info*** en **N** trozos (de longitud similar).
2. Creará **N** nodos (procesos), cada uno recibiendo un trozo de ***Info***. También recibirá el *pid* del máster y el número de nodo (entre 1 y **N**).

El máster guardará la lista de Pids de los nodos creados, y entrará en un bucle sin fin que se dedica a recibir mensajes *mapreduce* y procesarlos. Los mensajes *mapreduce* que espera el máster tienen la estructura {*mapreduce*, **Parent**, **Fmap, Freduce}** donde:

- **Parent**: pid del proceso que le está enviando el mensaje
- **Fmap**: Una función que realiza funciones de filtrado y ordenación. Recibe como entrada una lista (el fragmento de ***Info*** contenido en un nodo) y la posición del nodo (el valor entre 1 y **N** que lo distingue) y devuelve una lista de parejas {clave, valor}.
- **Freduce**: La función que procesa los resultados obtenidos de los nodos, ya combinados. Recibe una clave y dos valores, y produce una salida. Difiere de la versión habitual de reduce que se supone que recibe la lista completa de valores asociada a una clave; aquí los procesamos según llegan para aumentar el paralelismo aunque a costa de perder generalidad.

Cuando el máster recibe un mensaje *mapreduce*, crea un proceso que será el encargado de repartir las tareas map a los nodos, recibirlas, crear los procesos reduce y finalmente devolver el resultado. A continuación el proceso máster vuelve al bucle, donde esperará la llegada de un nuevo mensaje *mapreduce*.

El proceso creado para tratar la petición mapreduce recibirá los pids de los nodos y la información del mensaje ({*mapreduce*, **Parent**, **Fmap, Freduce}).** A continuación este proceso:

1) Les pasa un mensaje {*startmap*, self(), **Fmap**} a los **N** nodos.

2) Crea un nuevo diccionario (*dict*, ver las referencias más abajo), que permitirá saber los pids de los nodos reduce asociados a cada clave, y entra en un modo de espera a recibir mensajes para reunir los resultados de ***Fmap***. En este estado se reciben dos tipos de mensajes

- {Clave,Valor} → son mensajes enviados por un nodo que está procesando su trozo de información. Lo que hace es:
    1. Si la clave ya está en el diccionario simplemente enviar el mensaje {newvalue, Valor} al proceso reduce asociado.
    2. Si la clave no existe en el diccionario crear un nuevo proceso reduce al que se pasa inicialmente el valor {self(), **Freduce**, Clave, Valor}.

- {end} → Un nodo ha terminado de procesar el **Fmap**.

Este estado sigue hasta que todos los nodos han acabado (se han recibido N señales end).
En ese momento se envía a todos los procesos *reduce* el mensaje *end*.

3) Bucle de recepción de mensajes análogo al de la fase 2, pero ahora recolectando los mensajes {Clave, Valor} que devuelven los procesos *reduce* y acumulando en una lista los resultados. Acabará cuando se hayan recibido R mensajes, con R el número de procesos *reduce*. Al final se obtiene un nuevo diccionario que es el que se envía al padre.

Procesos Nodo

Contienen un trozo de información representado como una lista. Tras ser creados quedan a la espera de recibir un mensaje (*startmap*,**IdParent**, **Fmap**) que indica que deben aplicar la función **Fmap** a cada elemento de su lista, pasándole además la posición del nodo (este valor se usa rara vez). Cada vez que aplica la función a un elemento se obtiene una lista L de elementos de la forma {Clave, Valor}. El nodo le envía al **IdParent** tantos mensajes como elementos tenga la lista L. Cuando ha aplicado la función a todos los elementos de la lista, envía a **IdParent** el mensaje *end*. En este momento el nodo, que ha terminado su tarea, vuelve a pasar a la espera de recibir un nuevo mensaje (*startmap*, **IdParent**, **Fmap**).

Procesos Reduce

Cuando se crean reciben como valores iniciales {**IdParent**, **Freduce**, **Clave**, **Valor**}. Cada proceso reduce actúa de la siguiente forma:

1) Al crearse apunta los valores de entrada. En particular apunta que el **ValorActual** que se tiene de momento es **Valor**.

2) Entra en un bucle a la espera de mensajes, que pueden ser de dos formas:
- {*newvalue*,ValorNuevo} → Aplica la función **Freduce** a (**Clave**,ValorActual,ValorNuevo). El valor que se obtiene será el nuevo ValorActual, y se continúa en el bucle.
- {end} → En este caso se envía al **IdParent** el mensaje {Clave,ValorActual} y el proceso se termina.

Un ejemplo

En este ejemplo tenemos distintas temperaturas en distintas ciudades [{madrid,34},{barcelona,21}, {madrid,22},{barcelona,19},{teruel,-5}, {teruel, 14}, {madrid,37}, {teruel, -8}, {barcelona,30}, {teruel,10}]. Queremos calcular la temperatura máxima de las ciudades que han sobrepasado en algún momento los 28 grados.

Al crear el máster le pasamos la lista y le indicamos que lo divida por ejemplo en 3 trozos.
Por tanto el proceso máster crea (Fase 1) los siguientes procesos:

Nodo1
Trozo = [{madrid, 34},{barcelona,21},{madrid,22}]

Nodo2
Trozo= [{barcelona, 19}, {teruel,-5}, {teruel, 14}]

Nodo3
Trozo= [{madrid, 37}, {teruel, -8}, {barcelona,30},{teruel,10}]

En este momento al máster le llega un mensaje *mapreduce* con una función **Fmap** que se limita a filtrar los valores que tienen temperatura mayor de 28. La función reduce calculará el máximo de las temperaturas para cada clave. De esta forma si una ciudad tiene todas sus temperaturas por debajo de 28 no pasará a la siguiente fase.

Entonces el máster crea un nuevo proceso P que se encargará de este *mapreduce*. Este nuevo proceso envía el mensaje {*startmap*, self(), **Fmap**} a los 3 nodos. Los nodos contestarán enviando a P los siguientes mensajes:

Nodo 1: m1={madrid,34} y m2=end.
Nodo 2: m3=end.
Nodo 3: m4={madrid,37}, m5={barcelona,30} y m6=end.

Supongamos para simplificar que los mensajes llegan en orden al proceso P (aunque cualquier otra secuencia conducirá al mismo resultado).

- m1= {madrid, 34} P mira a ver si tiene en su diccionario de procesos reduce la clave madrid. Como no la tiene (el diccionario está vacío):
  - Crea un proceso reduce al que le pasa {initreduce, self(), **Freduce**, madrid, 34}.
  - Llamemos al pid de este proceso *reducemadrid*. En el diccionario insertamos (madrid, *reducemadrid*)
- m2=end. El máster apunta que 1 nodo ya ha acabado.
- m3=end. El máster apunta que 2 nodos ya han acabado.
- m4= {madrid, 37}. El máster busca en el diccionario madrid. Y lo encuentra con valor asociado *reducemadrid*. Envía a *reducemadrid* el mensaje {newvalue, self(), 37}. En ese momento el proceso *reducemadrid* aplica la función **Freduce**, que hace el máximo entre 34 y 37, obteniendo un nuevo máximo, 37.

- m5= {barcelona,30}. Como no existe en el diccionario creamos un proceso *reducebarcelona* al que se le pasa { initreduce, self(), **Freduce**, barcelona, 30}
- m6= end. El máster apunta que el tercer nodo ya ha acabado. Por ello se envía a los procesos *reducemadrid* y *reducebarcelona* el mensaje *end*.

Al recibir los mensajes *end*, *reducemadrid* envía al máster el mensaje m7= {madrid, 37} y *reducebarcelona* envía el mensaje m8= {barcelona, 30}. Ambos procesos terminan.

En la siguiente fase el proceso P inicializa la lista resultados a vacío y se pone a esperar. Recibe los dos mensajes:

- m7= {madrid, 37}. Lo añade a la lista y apunta que ya tiene un resultado.
- m8= {barcelona, 30}. Lo añade a la lista y apunta que ya tiene dos resultados.

Como ya tiene 2 resultados y el número de procesos reduce es 2 envía al proceso llamante Padre, `{self(), Lista}` y acaba.

Lo que hay que hacer (8 puntos sobre 10)

- Implementar el modelo mapReduce siguiendo estas ideas.
- Buscar algún otro ejemplo sencillo de mapReduce (que no sea el del ejemplo anterior) y probarlo con el modelo definido.

Para facilitar la corrección, se requiere especificar y comentar bien el código, incluyendo una descripción del ejemplo usado.

Extensiones (2 puntos sobre 10)

Elegir 3 de los siguientes puntos:

- En lugar de usar dict probar a usar orddict, gb_trees, proplists. o estructuras ETS
- Permitir al máster añadir/borrar nodos para tener datos variables (añadir o eliminar trozos)
- Definir una topología distinta a la del enunciado (siempre y cuando esté bien motivada, no se pierda distribución y admita problemas del tipo del ejemplo).
- Hacer que los nodos a su vez creen procesos, uno por cada elemento de su lista para realizar en paralelo los procesos map.
- Se puede implementar el máster como servidor mediante un comportamiento gen_server.

Más información

Sobre diccionarios en Erlang:
http://www.erlang.org/doc/man/dict.html
http://www.techrepublic.com/article/working-with-dictionaries-in-erlang/

Sobre *mapreduce*:
MapReduce: Simplified Data Processing on Large Clusters
Dean and Ghemawat (Google)
OSDI 2004

Sobre gen_server:
http://www.erlang.org/doc/man/gen_server.html