

## Tarea 5 : Programación Distribuida

|                       |                     |
|-----------------------|---------------------|
| Lluch, Pablo          | Fuentes, Tomás      |
| pablo.lluch@gmail.com | tafuentes@gmail.com |

7 de diciembre de 2013

## 1. Problema Distribuido

El tipo genérico de problemas que se decidió atacar corresponde al planteado por el enunciado. Específicamente, el de una red adhoc en que se tiene uno de los procesos como administrador del problema, y en que se tiene una matriz de adyacencia que define cuáles son los elementos conexos. El proceso principal se encarga de crear y distribuir las tareas, así como unir las una vez han sido resueltas. Todos los procesos participan del proceso. El problema es, entonces, el de sincronizar a estos distintos procesos y eventualmente ver cómo se puede usar la información de topología para distribuir la carga de las tareas.

## 2. Solución

Para solucionar el problema planteado, se optó por utilizar el código base en Java entregado a los alumnos. La solución está en dos carpetas distintas, al igual que el código base: En demo se encuentra la clase `Executor`, que fue ligeramente modificada para poder trabajar con un parámetro extra, que especifica el problema que se quiere resolver (bonus del enunciado). En `demo_base` se encuentra el código correspondientes a los problemas y trabajadores de la red adhoc.

El problema se carga por reflection y se le pide al usuario por consola, inmediatamente después de que se le pide elegir si quiere cargar una matriz de adyacencia aleatoria o bien una prefabricada. Lo único que se debe especificar es el nombre de la clase que se desea utilizar, el cual debe estar en la misma carpeta que los otros archivos del código de `demo_base`. El único requisito es que la clase debe derivar de la interfaz `Task`, la cual contiene declaraciones de métodos genéricos. Como ejemplo, se implementó la clase `QuicksortTask`, la cual implementa un quicksort distribuido. Retorna una instancia del problema resuelto que será serializado y enviado de vuelta al administrador.

Los métodos que requiere la interfaz `task` son:

1. `int getAnswerCount();` // Corresponde al número de respuestas que debería esperar a recibir el proceso principal. En el caso de `QuicksortTask`, corresponde al número de elementos del arreglo.

2. `Task executeTask();` // Método ejecutado por los objetos de la clase `Worker` para trabajar sobre una instancia de un problema concreto.
3. `public ArrayList < Task > getNextTasks(Object[] currentResults);`  
// Es el principal método de ejecución. Recibe como parámetro un arreglo de `Objects` correspondientes a la solución actual, el cual puede contener cualquier tipo de objeto. En el caso de `QuicksortTask`, son enteros. Puede tener un número arbitrario de elementos. Retorna un `ArrayLists` de `Tasks` que corresponden a los próximos subtasks que deben ejecutar otros procesos en paralelo. Nuevamente, para el caso de `QuicksortTask`, es un `ArrayList` de dos elementos correspondientes a las dos llamadas recursivas de `Quicksort`.

El resto de los miembros de `Task` pueden ser a gusto del usuario y la clase del problema puede definir lo necesario para resolverlo. `Quicksort` define un arreglo arbitrario y algunas cosas como pivotes e índices.

### 3. Ejecución

Para ejecutar la solución, se hace de exactamente el mismo modo que la solución base, salvo el punto mencionado previamente en que se debe especificar la clase que define y soluciona el problema.

Notas adicionales: El programa fue probado en Linux, pero como la solución base fue probada en Mac OS X no deberían haber problemas de compatibilidad.