Programación de Objetos Distribuidos **2012**

Trabajo práctico especial

Objetivo

Distribuir un calculo complejo con capacidad de **escalar horizontalmente de forma dinámica** en la cantidad de computadoras utilizadas, y soportando la caída hasta un nodo.

Enunciado

Un problema asociado común al que se enfrentan los servicios de seguridad de cualquier país es la detección de envíos clandestinos de información escondidos en señales de radiofrecuencia.

Con el fin de automatizar dicha detección, se ha construido un sistema que hace una clasificación previa intentando hallar patrones de señales repetidos que permitan clasificar las señales como ruido o como señales interesantes para ser analizadas en más detalle.

La idea es simple: considerando al ruido como una señal aleatoria con un distribución uniforme, se espera que la correlación (grado de similitud) entre dos señales de ruido sea muy baja. Entonces, el mecanismo de clasificación consiste en comparar señales contra un grupo de patrones almacenados previamente, para determinar si hay cierta repetición estructural que justifique la hipotesis de que la señal no es ruido electrónico.

El problema es que a medida que aumenta el numero de señales a comparar el problema se vuelve muy demandante para poder ser ejecutado en una única computadora. Por eso se ha comisionado la construcción de un sistema distribuído que pueda crecer dinámicamente según sea necesario.

Descripción del trabajo

Junto al enunciado, el alumno recibirá una implementación corre en un único thread. El objetivo es construir una implementación que ofrezca la misma funcionalidad y de los mismos resultados, pero que funcione de forma concurrente (aprovechando todos los nucleos disponibles) y distribuída (pudiendo aprovechar varias computadoras en paralelo).

El tipo de problema tiene una tasa de escalabilidad muy alta, así que se espera un crecimiento de performance asintoticamente lineal (duplicar la cantidad de procesadores debería aproximadamente reducir el tiempo a la mitad).

El trabajo práctico del sistema consiste en la implementación de dos interfases remotas, **que no podrán ser modificadas**:

SPNode, que permitirá gestionar el ciclo de vida de los nodos en el sistema (conectar un nodo a un cluster, desconectarlo) y ver estadísticas.

SignalProcessor, que será la interfaz utilizada normalmente por los clientes y contiene métodos para agregar señales y comparar señales con las registradas en el sistema.

Ambas interfases son remotas, de tal forma que las implementaciónes deberán poder ser invocadas vía RMI.

Se proveerán ejemplos de uso de las interfases, así como también casos de prueba para verificar el correcto funcionamiento del sistema. En cualquier caso, los resultados obtenidos deberán coincidir con la implementación provista.

El sistema resultante deberá ser 1-tolerante ante caídas. Es decir que cualquier cliente deberá poder completar un pedido realizado contra un nodo si otro nodo cae durante el procesamiento (si cae el nodo al que esta conectado el cliente, se espera que el cliente maneje el reintento conectandose a otro nodo).

Para ello se establecen los siguientes requisitos mínimos:

- Todos los nodos expondrán las dos interfases mencionadas. **Deberá ser indistinto utilizar SignalProcessor desde cualquier nodo**. SpNode, por supuesto, afectará solo al nodo que la implementa.

- Los nodos **no deberán implementar durabilidad**. Es decir, todas las señales quedarán almacenadas en memoria. Cuando el último nodo de un cluster termine, se perderan todas las señales.

- **La cantidad de señales por nodo deberá mantenerse lo más pareja posible**. Las señales deberán poder ser agregadas desde cualquier nodo. En particular deberá evitarse que un nodo tenga una única copia de las señales en caso de que dicho nodo caiga. Para el problema en cuestión es suficiente que exista una copia por cada señal almacenada, y que se determine de forma aleatoria que nodo/s la almacenaran dentro del cluster.

- Deben poder **agregarse nodos mientras la aplicación está funcionando**. La aparición de un nuevo nodo implicará la redistribución de las señales existentes para **mantener balanceada la cantidad de las mismas por nodo**.

- Deben poder **quitarse nodos mientras la aplicación está funcionando**. Las señales que se

encontraban en dicho nodo deberán ser migrados a otros nodos manteniendo el principio de **ecualizar la cantidad de señales por nodo**. Si el nodo que se quita es el único presente, se finalizará la aplicación.

- **El sistema deberá reaccionar correctamente ante la caída de un nodo**, rebalanceando la carga de trabajo en los nodos restantes al detectar la situación. Si hubiese consultas en curso, originadas por toros nodos, las mismas deberán ser reintentadas o corregidas usando las replicas de las señales. En ningún caso deberan fallar.

- El sistema operará en **dos modos:** normal y degradado**.** En modo **normal** el sistema será 1-tolarante a caídas. En modo **degradado** el sistema no será tolerante a fallas. Cuando un nodo esté funcionando solo o sea el único de un cluster deberá considerarse que opera en modo degradado**.** Dos o mas nodos en un cluster deben pasar a modo normal luego de un pequeño tiempo de sincronización. Cualquier cambio en la topología del cluster (nuevos nodos, caida o salida de un nodo) puede dejar temporalmente al cluster en modo degradado, pero mientras que queden al menos dos nodos, el sistema debe retornar a modo normal lo antes posible.

Sobre el código fuente

Todas las implementaciones del alumno deben ubicarse en el siguiente paquete o paquetes que lo tengan como prefijo:

ar.edu.itba.pod.legajoXXX

Donde XXX es el número de legajo del alumno. Cualquier clase, interfaz o archivo fuera del paquete será ignorado.

La implementación del deberá incluír una clase llamada Server con un metodo main que **debe instanciar el RMI Registry de manera embebida, y registrar la implementación de las dos interfases, en el puerto** . Es decir, no debe hacer falta levantar el RMI Registry mismo desde afuera de la aplicación.

El resultado final deberá ser un jar ejecutable, que será invocado de la siguiente manera:

java -jar <nombre del jar> <puerto> <threads>

Donde <puerto> es el puerto donde escuchará el RMI Registry y <threads> es el número de hilos de ejecución de procesamiento que deberían instanciarse en el nodo (puede haber hilos de ejecución extra adminsitrativos o de limpieza).

Se permite el uso de las siguinetes librerías:

jGroup - versión 3.0.3

Google Guava – versión 11.0.1

El uso de cualquier otra librería o versión deberá ser autorizado por la cátedra previamente.

Los proyectos deberán compilar y correr usando la version 7 del JDK.

Modalidad y Calificación

El trabajo será desarrollado individualmente. Cualquier **plagio** detectado será penalizado con una calificación de 0 - no recuperable – reservándose la cátedra el derecho de tomar medidas administrativas adicionales.

La fecha límite de entrega será **12/11/2010 a las 19:00 horas**. Quienes no entreguen en dicha fecha dispondrán de una fecha tardía, que vence el **19/11/2010 a las 19:00 horas.**

La entrega será calificada con una nota entre 0 y 10. **La nota mínima para considerar aprobado el trabajo práctico será de 7 puntos**. En el caso de las entregas en primera fecha que estén aprobadas, la nota definitiva es la obtenida en el trabajo. En el caso de **entregas aprobadas directamente en segunda fecha, la nota definitiva es de 5 puntos** independientemente del resultado. Recordar que siempre se considera un trabajo práctico como aprobado si su nota es igual o superior a los 7 puntos. No hay posibilidad de volver a entregar el trabajo si el mismo no se encuentra aprobado, tanto en la primera como en la segunda fecha.

Junto con la implementación de la aplicación single thread, la cátedra proveerá a los alumnos de 9 casos de pruebas a ser utilizados en la evaluación. **Cualquier implementación entregada por algún alumno que no funcione con cualquiera de dichos casos de prueba o arroje un resultado diferente a la implementación de referencia provista, será directamente desaprobada**.

Entregables

Se deberá entregar los siguientes ítems:

- Código fuente completo del trabajo práctico. El mismo debe ser enviado por mail al mail de la cátedra.

- Documentación sobre como crear el jar ejecutable. Incluirla en un archivo Readme.txt dentro del proyecto. El proceso puede ser utilizar **Maven (confirmar con la cátedra antes de utilizar otra herramienta)**. En ambos casos, la ejecución **no debe contener errores** (classpath seteados a mano dependientes de la ubicación, etc). Toda la ejecución de prueba se va a realizar por línea de comando. **No asumir la existencia de algun IDE** (Eclipse o similares). El jar ejecutable debe contender dentro todas las dependencias necesarias (incluído el jar provisto por la cátedra).

Distribución de la nota / criterios de calificación

A la hora de calificar el trabajo práctico se tendrán en cuestión los siguientes aspectos:

- ¿La aplicación funciona correctamente en un solo nodo? (1 puntos)
- ¿La aplicación funciona correctamente en varios nodos? (2 puntos)
- ¿La cantidad de señales por nodo permanece balanceada al agregar un nuevo nodo? (1,5 punto)
- ¿ La cantidad de señales por nodo permanece balanceada al quitar un nodo? (1 puntos)
- ¿ La cantidad de señales por nodo permanece balanceada al caerse un nodo, y no se pierden señales? (1,5 puntos)
- ¿La aplicación es capaz de escalar de forma asintoticamente lineal tanto en un procesador como en varios? (3 puntos)

Se penalizará el uso de pooling no bloqueante (ciclos preguntando por algun evento hasta que llegue uno – con alto consumo de procesdor) y el uso de esperas (Thread.sleep o similares) con el objeto de reducir la frecuencia de condiciones de carrera.

La cátedra se reserva el derecho de otorgar puntos adicionales, así como también de otorgar puntuación parcial de acuerdo a su entendimiento de cómo se han cumplido los requisitos de la simulación en cada caso.

Apendice - Comparación de señales

Se proveera del algoritmo de comparación de señales, pero para un mejor entendimiento, a continuación se explica el funcionamiento del mismo, en caso de que sea modificado para la implementación concurrente.

Cada señal está representada como un arreglo de bytes.

La comparación entre dos señales resulta en un número de tipo *double* que indica la desviación de una señal con respecto a la otra. El resultado de una comparación es 0 si las señales son iguales o una es un desplazamiento pequeño de la otra, o un número positivo que cuantifica cuan diferentes son.

El calculo de la diferencia se realiza de la siguiente manera:

- Comparar las señales entre si con un desplazamiento de +/- un 10%. Si las señales tuviesen 100 bytes de longitud, esto sería comparar:

  s1[10, 11, …, 98, 99] con s2[0, 1, 2, …, 89] ← señal 1 desplazada 10 posiciones a izq.

  s1[9, 10, …, 98, 99] con s2[0, 1, 2, …, 89, 90]

  ...

  s1[1, 2, …, 98, 99] con s2[0, 1, 2, …, 97, 98] ← señal 1 desplazada 1 posición a izq.

  s1[0, 1, …, 98, 99] con s2[0, 1, 2, …, 98, 99] ← señales sin desplazar

  s1[0, 1, …, 97, 98] con s2[1, 2, …, 98, 99] ← señal 1 desplazada 1 pos. a derecha

  …

  s1[0, 1, …, 89, 90] con s2[9, 10, …, 98, 99] ← señal 1 desplazada 9 pos. a derecha

  s1[0, 1, …, 88, 89] con s2[10, 11, …, 98, 99] ← señal 1 desplazada 10 pos. a derecha

- En cada comparación, calcular la suma de cuadrados de la diferencia entre los bits de las señales, y dividir por la cantidad de muestras comparadas.
- Quedarse con el mínimo valor de todas las comparaciones.

El objetivo de esta comparación es detectar dos señales muy parecidas, pero que fueron capturadas con un desplazamiento diferente.

Como ejemplo muy reducido, las siguientes señales:

[0, 2, 1, 1, 4, 5, 0, 2, 6, 8, 3, 5, 7, 0, 3] y [2, 1, 1, 4, 5, 0, 2, 6, 8, 3, 5, 7, 0, 3, 4] tienen una diferencia igual a 0

Apendice 2 - Interfaces remotas

A continuación se copian las interfases remotas que gobiernan la interacción del sistema con los clientes:

```
/**
 * Signal processor that finds similarities between signals
 */
@DoNotChange
public interface SignalProcessor extends Remote {

	/**
	 * Adds a new {@link Signal} to the database.
	 * This method may return before the signal is actually added, but calling get
<code>findSimilarTo(...)</code> from the same client after adding
	 * some signals MUST take them into account (read your writes consistency model).
	 *
	 * If the same signal is added twice to the system, the behaviour is unespecified.
Implementations may discard the duplicate signal or store it twice.
	 *
	 * @param signal The signal to add.
	 */
	void add(Signal signal) throws RemoteException;

	/**
	 * Returns the top 10 {@link Signal}s that best match the given signal.
	 * If the system has less than 10 signals registered, it may return less than 10
results, but it will never return more.
	 * If there are several signals with the same comparison value, any of them may be in
the results in an unespecified order.
	 * @param signal The signal to compare against stored signals
	 */
	Result findSimilarTo(Signal signal) throws RemoteException;
}
```

```
/**
 * Administrative interface for SignalProcessing nodes
 */
@DoNotChange
public interface SPNode extends Remote {

	/**
	 * Joins a cluster.
	 * If there is no cluster found under the given name, a new cluster will be started.
	 * In order to join a cluster the node must not have any signal registered.
	 *
	 * @param clusterName Name of the cluster to create or join
	 * @throws IllegalStateException if the node joins a cluster but already has some
values stored
	 * @throws IllegalStateException if the node is already part of a cluster
	 * @throws RemoteException
	 */
	void join(String clusterName) throws RemoteException;

	/**
	 * Cleanly disconects a node the the cluster.
	 * Note that this command doesn't stop the node, but detaches it from a cluster and
returns it to it's initial state: no signals
	 * stored, and not part of any cluster.
	 * If the node is the only node in the cluster, the cluster will be shutdown.
	 */
	void exit() throws RemoteException;
```

```
        /**
         * Returns the running {@link NodeStats} for this node.
         * @return Statistics for this node
         * @throws RemoteException
         */
        NodeStats getStats() throws RemoteException;
}
```