

Informe de Practica 2

SISTEMAS DISTRIBUIDOS Y PROGRAMACION PARALELA



Juan Cruz Real – Sofia vazquez

Juancreal1@gmail.com – sofigimevazquez@hotmail.com

# Consignas

**Requisitos a tener en cuenta:** Cada alumno o grupo de alumnos debe desarrollar las actividades prácticas enunciadas.

Además, se debe generar un informe como resultado del trabajo realizado. Este debe incluir:

1. Respuesta a las consultas, evaluaciones de métricas y tiempos, gráficas, conclusiones y, cuando corresponda, propuestas de mejoras sobre los servicios solicitados
2. Pasos a seguir detallados para poder correr los ejercicios a la hora de realizar la corrección.

Opcional: Gestionar y mantener un registro de actividades y operaciones (Logs) en memoria (volátil) y disco (no volátil). Este apartado será contemplado positivamente a la hora de la evaluación.

1) Desarrolle una red P2P de carga, búsqueda y descarga de archivos siguiendo las siguientes pautas:

- Existen dos tipos de nodos, Maestros y Extremos. Los primeros, son servidores centralizados replicados (al menos 2 nodos) que disponen del listado actualizado de los nodos extremos y se encargan de gestionar la E/S de los peers. Los segundos cumplen dos funciones en el sistema: realizan consultas (como clientes) y atienden solicitudes (como servidores).

- Funcionamiento:

-- Cada extremo dispone de un parámetro definido en un archivo de inicialización con las direcciones IP de los nodos Maestros. Al iniciarse se contacta con un maestro el cual funciona como punto de acceso al sistema e informa cuáles son los archivos que dispone para compartir. Luego, está atento a trabajar en dos modos (cliente y servidor), mediante un mensaje de tipo ack, que efectivamente recibió el mensaje que estaba en la cola.

-- Como cliente, deriva consultas al nodo maestro y una vez obtenida la respuesta, seleccionará el/los recursos que desee descargar y se contactará con el par correspondiente para descargar el/los archivo/s.

-- Como servidor, recibe la consulta, revisa si matchea la consulta con alguno de los recursos disponibles y devuelve los resultados al nodo que solicitó resultados.

A partir de los conceptos vistos en la teoría, critique este modelo y presente mejoras en su propuesta.

2) Un banco tiene un proceso para realizar depósitos en cuentas, y otro para extracciones. Ambos procesos corren en simultáneo y aceptan varios clientes a la vez. El proceso que realiza un depósito tarda 40 mseg entre que consulta el saldo actual, y lo actualiza con el nuevo valor. El proceso que realiza una extracción tarda 80 mseg entre que consulta el saldo (y verifica que haya disponible) y lo actualiza con el nuevo valor.

a) Escribir los dos procesos y realizar pruebas con varios clientes en simultáneo, haciendo operaciones de extracción y depósito. Forzar, y mostrar cómo se logra, errores en el acceso al recurso compartido. El saldo de la cuenta puede ser simplemente un archivo de texto plano.

b) Escribir una segunda versión de los procesos, de forma tal que el acceso al recurso compartido esté sincronizado. Explicar y justificar qué partes se deciden modificar

3) Construya una red flexible y elástica de nodos (servicios) la cual se adapte (crece / decrece) dependiendo de la carga de trabajo de la misma. El esquema será el de un balanceador de carga y nodos detrás que atienden los pedidos. Para ello deberá implementar mínimamente:

- Simulación de carga de nodos. Creación dinámica de conexiones de clientes y pedidos de atención al servicio publicado.

- Protocolo de sensado para carga general del sistema. Elija un criterio para detectar esa carga y descríbalo. Ejemplo: cantidad de clientes en simultáneo que el servicio puede atender. Si se excede esa cantidad, el punto de entrada (balanceador de carga) crea dinámicamente un nuevo servicio.

- Definición de umbrales de estado {sin carga, normal, alerta, crítico}

- Creación, puesta en funcionamiento de los servicios nuevos, y remoción de ellos cuando no sean más necesarios. Para esto el balanceador puede contar con una lista de IPs donde los servicios están instalados y pueden correr. De forma tal que arrancando inicialmente con 2 servicios en 2 nodos distintos, el sistema escala dinámicamente en función de la carga del sistema, usando los nodos listados en ese archivo de configuración. Si fuera necesario, puede haber más de un servicio en un mismo nodo. El servicio debe ser multi thread.

4) El operador de Sobel es una máscara que, aplicada a una imagen, permite detectar (resaltar) bordes. Este operador es una operación matemática que, aplicada a cada pixel y teniendo en cuenta los pixeles que lo rodean, obtiene un nuevo valor (color) para ese pixel. Aplicando la operación a cada pixel, se obtiene una nueva imagen que resalta los bordes.

a) Desarrollar un proceso centralizado que tome una imagen, aplique la máscara, y genere un nuevo archivo con el resultado.

b) Desarrolle este proceso de manera distribuida donde se debe partir la imagen en n pedazos, y asignar la tarea de aplicar la máscara a N procesos distribuidos. Después deberá juntar los resultados. Se sugiere implementar los procesos distribuidos usando RMI.

A partir de ambas implementaciones, comente los resultados de performance dependiendo de la cantidad de nodos y tamaño de imagen.

c) Mejore la aplicación del punto anterior para que, en caso de que un proceso distribuido (al que se le asignó parte de la imagen a procesar) se caiga y no responda, el proceso principal detecte esta situación y pida este cálculo a otro proceso.

# Como ejecutar nuestros programas

1. Hacer el git clone de nuestro proyecto
2. Abrir Eclipse con workspace en la raíz del repositorio.
3. Ir a Archivo > Importar proyecto y seleccionar la carpeta *tp2* dentro del repositorio.
4. Click en *Finalizar*.
5. Descargar las dependencias necesarias.
6. Ver las instrucciones para cada ejercicio en particular.

# Respuestas

**EJERCICIO 1**

Primero y principal se deben instalar las dependencias de java, ingresando el comando “mvn install”

### Como ejecutarlo

### **1. Se ejecuta de Server la clase ServerMain:** edu.unlu.sdypp.ej1.Server.ServerMain

### **2. Se ejecuta de Peer la clase PeerMain:**

### edu.unlu.sdypp.ej1.Peer.PeerMain

Luego de esto se abre la consola de Peer para poder observar las distintas opciones, debe ingresar “options” para ver las mismas listadas.

En el caso que se desee correr más de un Peer se debe crear un archivo para cambiar el json por el defecto, ya que c/Peer persiste localmente los archivos compartidos en la red peer to peer y colocarlo en edu.unlu.sdypp.ej1.Peer.Resources para luego correr el comando:

**set [path-file]**

**EJERCICIO 2**

Primero y principal se deben instalar las dependencias de java, ingresando el comando “mvn install” como en el ejercicio mencionado anteriormente.

**Edu.unlu.sdypp.ej2.withoutsynchronization 🡪 Sin sincronización**

**Edu.unlu.sdypp.ej2.withsyncronize 🡪 Con sincronización.**

Para forzar el error de actualización se deben ejecutar de la siguiente manera:

### Como ejecutarlo

**1. ExtraccionServer**

**2. DepositoServer**

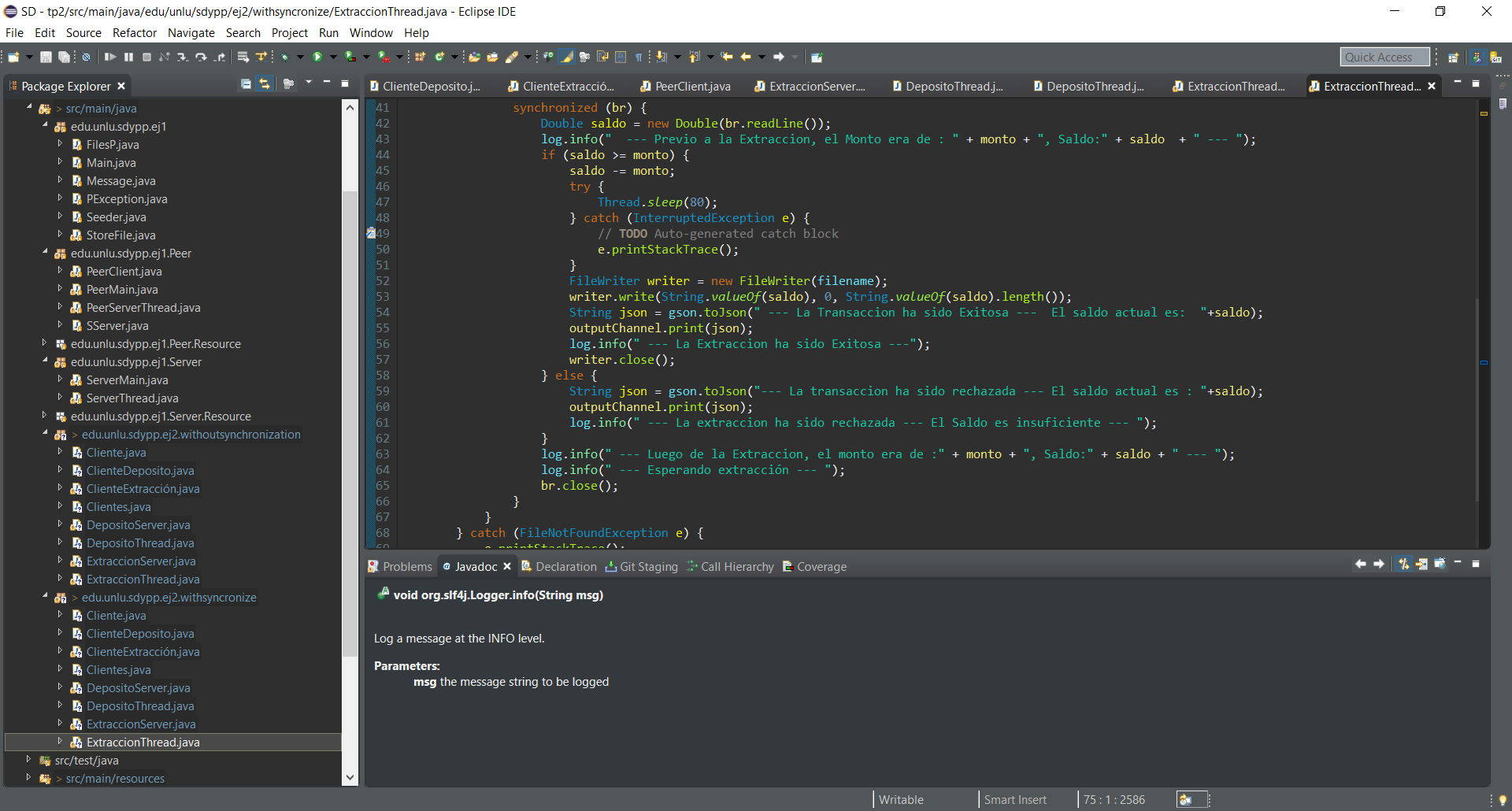
**3. ClienteDeposito**

**4. ClienteExtraccion**

### 

### Para sincronización

Lo que se decide sincronizar son las partes de código en las que se lee, procesa y escribe en el archivo de saldos. De esta manera se fuerza a que las transacciones sean de manera atómica y que no se puedan modificar entre ellas. En la siguiente imagen se puede ver la extracción de manera sincronizada:



## EJERCICIO 3 y 4

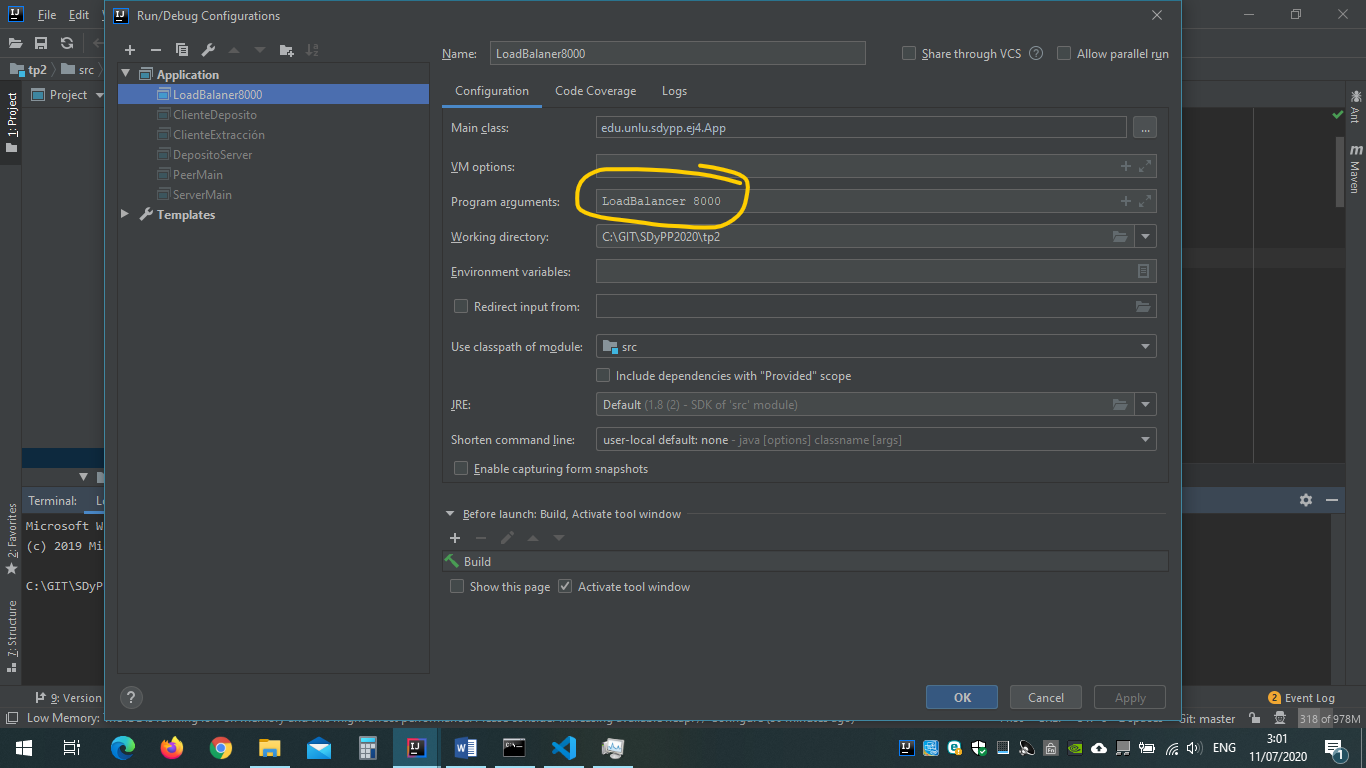
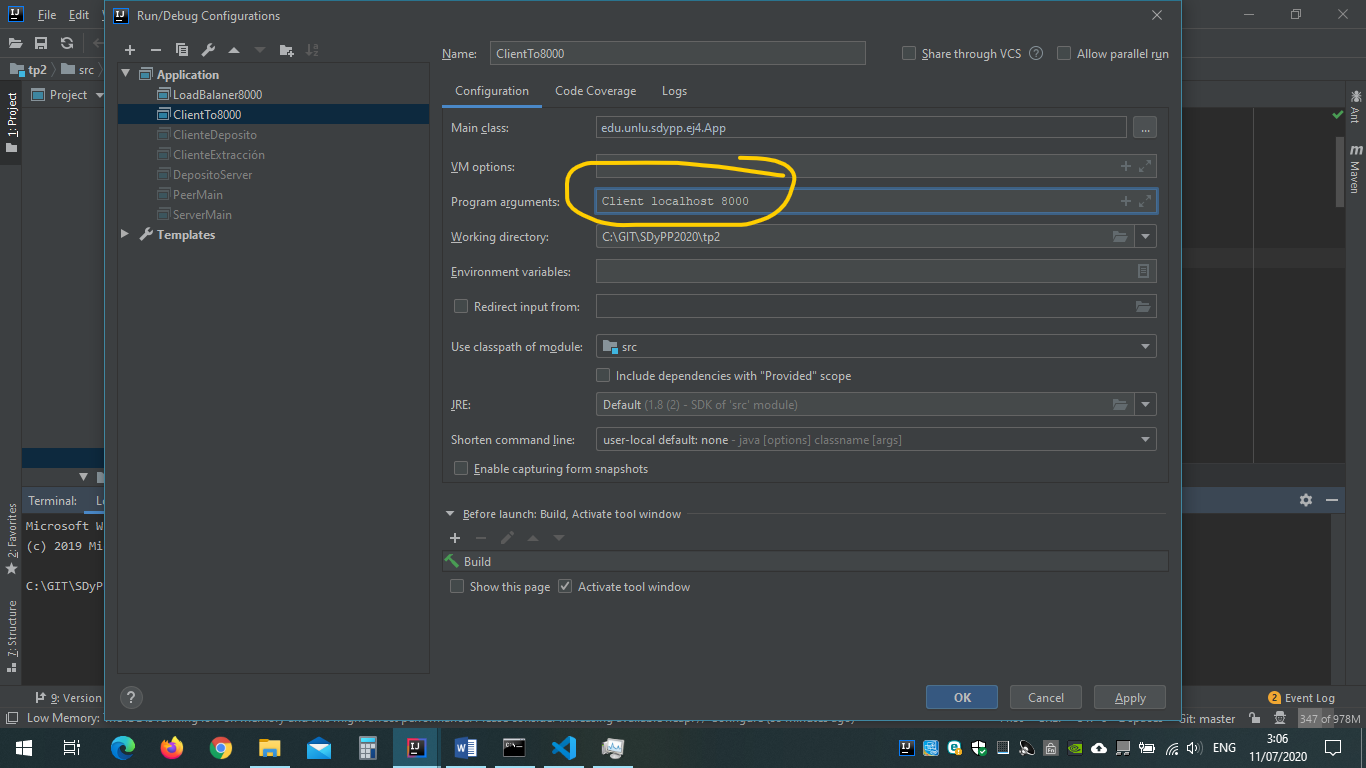
Ambos ejercicios los hice al mismo tiempo pensando que debería poder reutilizar el sistema de nodos a la hora de distribuir la carga en el procesamiento de las imágenes usando Sobel.

Además reutilice la interfaz Computable y Tarea hechas en el TP1 dado que utilice RMI.

### Requisitos

* Programa "fuser" instalado en los nodos (Solo Linux).
* Programa SSH-Server instalado y configurado en los nodos.
* Java 8 instalado en los nodos. **NOTA:** La versión de Java que se usa para ejecutar el proyecto como para levantar los nodos debe ser exactamente la misma. La versión con la que se desarrollo fue OpenJDK Corretto 1.8.0\_252 disponible en: <https://github.com/corretto/corretto-8/releases>

### Como ejecutarlo

1. Abrir el proyecto en su IDE.
2. Ir al archivo *resources/application.conf* y configurar los nodos. (El archivo posee comentarios explicativos)
3. Generar un jar ejecutable (artifact) utilizando la clase ***edu.unlu.sdypp.ej4.App*** como Main.
4. Copiar el jar generado al home del usuario que se va a utilizar como nodo. **NOTA:** darle permisos de ejecución al usuario sobre este archivo.
5. Si no la posee, cree una carpeta *images* a la misma altura (filesystem) donde se va a ejecutar la aplicación.
6. Para levantar el servidor, en su IDE, ejecute el *main()* de App pasándole como parámetro *Loadbalancer <puerto>*. EJ: 
7. Para conectarse al servidor, en su IDE, ejecute el *main()* de App pasándole como parámetro *Client localhost <puerto-del-servidor>*. EJ: 
8. Las imágenes con los bordes detectado seran guardadas en la carpeta ***sobel*** al mismo nivel (filesystem) donde se ejecuta la aplicación cliente.

### Errores conocidos:

#### java.awt.AWTError: Assistive Technology not found: org.GNOME.Accessibility.AtkWrapper

**Error producido por una configuracion de OpenJDK. Solucion:** [**https://askubuntu.com/questions/695560/assistive-technology-not-found-awterror**](https://askubuntu.com/questions/695560/assistive-technology-not-found-awterror)

#### java.io.InvalidClassException: javax.swing.ImageIcon;

**Este error se debe a que no se uso la misma version de java para compilar el proyecto que para ejecutar el jar. Para solucionarlo verificar en el PATH del sistema que la version de java sea la misma que estamos usando para compilar el proyeto en el IDE. Recordar reiniciar el equipo luego de cambiar el PATH.**