Universidad Autónoma de la Ciudad de México

Práctica Difusión de datos basado en chismes P2

Enfoque Pull, Push y Pull-Push

Sistemas Distribuidos

Profesora: Elizabeth López Lozada

ESQUIVEL GUERRERO JANET EDITH

MARTINEZ MÉNDEZ JONATAN IVAN

MIRANDA MORALES EMMA ADELIZ

INDICE

1. Introducción

La difusión de dato basada en chismes se refiere a la propagación de información, a menudo de manera informal o no verificada En el contexto de la programación, podríamos relacionarlo con la forma en que los datos se transmiten o propagan entre diferentes partes de un sistema o red.

En el presente proyecto implementaremos un método de difusión de datos para 'infectar' a varias computadoras. Puede generar un mensaje en pantalla que indique cual nodo ha sido infectado. En este caso usamos los tres enfoques, Pull, Push y Pull-Push. Debemos aclarar que esto solo es una simulación, para nada se busca hacer malas prácticas. Esperando les sea de utilidad para futuros proyectos

2. Difusión de datos basado en chismes

La difusión de datos puede abordar diferentes aspectos que usamos en nuestra vida diaria. Como la difusión de datos en redes sociales, modelos de difusión en sistemas distribuidos, desinformación y difusión de datos incorrectos.

Nuestra idea principal fue apoyarnos de algoritmos que permiten el paso de mensajes, en los cuales mediante dicho mensaje se pueda propagar el “virus”. Se crearon dos clases, Cliente y Servidor.

2.1 Clase Cliente

La función principal de un cliente es solicitar peticiones al servidor.

Primero importamos las librerías necesarias.

```
import java.util.Vector;
import javax.swing.JOptionPane;
import org.apache.xmlrpc.XmlRpcClient;
```

Creamos la clase que se llama Cliente

```
public class Cliente {
```

Creamos el método main y declaramos dos variables tipo string; x y y

```
public static void main(String[] args) {
    //Declaramos dos variables de tipo string ya que los recibe
    String x="", y="";
```

Creamos un objeto donde almacenaremos los resultados de la infección

```
Object resultado;
```

Aquí utilizamos la sentencia try y creamos una instancia de la clase XmlRpcClient con la URL “http:192.168.1.71:8080”. Los que se está intentando es establecer una conexión con un servidor remoto en el puerto 8080. En el cual el objeto cliente se utilizará para comunicarse con ese servidor.
Posteriormente creamos un vector llamado parámetros que almacenará los objetos tipo String y que nos será de utilidad para pasar los parámetros al servidor remoto.

```
try{


    //Establecemos las conexión en la 8080
    XmlRpcClient cliente = new XmlRpcClient ("http:192.168.1.71:8080");
    //Agregamos un vector ya que recibe los parametros
    Vector<String> parametros = new Vector<String>();
```

Primero se muestra una ventana emergente con el mensaje “el cliente se ha conectado”. Posteriormente entra en un bucle while que muestra un menú de opciones para una supuesta “infección de 4 computadoras”. En este caso el usuario puede elegir entre dos opciones; comenzar la infección o salir del bucle.

```
JOptionPane.showMessageDialog(null,"El cliente se ha conectado");

while (true){
    String menu = JOptionPane.showInputDialog(null,"Infección de 4 computadoras"
    +"\n1. Comenzar infección \n"
    +"\n2. Salir", "Infección \n", JOptionPane.DEFAULT_OPTION);
```

En el caso de elegir comenzar la infección entramos en el caso 1. Se solicitara al usuario que ingrese el “ID del nodo” que desea infectar, posteriormente se agrega el ID del nodo, se manda a llamar el método en el servidor llamado “Mi servidor” para iniciar la infección, se muestra un mensaje de confirmación “Infección iniciada en el nodo” + el id del nodo. Si ocurre una excepción durante este proceso, se muestra un mensaje de error, si el usuario cancela o ingresa un ID invalido, se muestra un mensaje de advertencia.

```
switch(menu){

    case "1":
        //Solicitar al cliente que ingrese el ID del nodo a infectar
        String idNodo = JOptionPane.showInputDialog(null, "Ingrese el ID del nodo a infectar:", "Iniciar Infección", JOptionPane.QUESTION_MESSAGE);
        //Verificamos si ingreso un ID valido
        if (idNodo != null && !idNodo.isEmpty()) {
           try {
            //Agregamos el ID del nodo como parámetro
            parametros.add(idNodo);
            //Llamamos al método en el servidor para iniciar la infección
            resultado = cliente.execute("Miservidor", parametros);
            //Mostramos el mensaje de confirmación
            JOptionPane.showMessageDialog(null, "Infección iniciada en el nodo " + idNodo, "Iniciar Infección", JOptionPane.INFORMATION_MESSAGE);
            } catch (Exception ex) {
             //Mostramos un mensaje de error si ocurre una excepción
            JOptionPane.showMessageDialog(null, "Error al iniciar la infección: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
           }
        } else {
         //Mostramos un mensaje si se cancela o no se ingresa un ID válido
        JOptionPane.showMessageDialog(null, "No se ingresó un ID válido.", "Iniciar Infección", JOptionPane.WARNING_MESSAGE);
        }
        break;
```

En caso de que se seleccione el Caso 2, se muestra una ventana emergente con el mensaje “Saliendo”. El argumento JOptionPane.WARNING_MESSAGE, será el que especifique el mensaje de emergencia. Finalmente con System.exit el programa se detiene inmediatamente y se cierra.

```
        case "2":
            JOptionPane.showMessageDialog(null, " Saliendo",null, JOptionPane.WARNING_MESSAGE);
             //Salimos del programa
            System.exit(0);
            break;
    }
}
```

La sentencia catch trabaja en conjunto con la sentencia try declarada casi al inicio del código. Si ocurre algún problema durante la lectura (por ejemplo, el archivo no existe o hay un error de lectura), El bloque catch captura esta excepción y muestra un mensaje de error con el texto “Error: Archivo no encontrado”.

```
        } catch(Exception e) {
            //En caso de problema
            JOptionPane.showMessageDialog(null, "Error" + e.getMessage());

        }
    }
}
```

2.2 Pom.xml-Cliente

El archivo POM (Project Object Model) en formato XML nos permitirá transmitir los datos entre el cliente y el servidor, de este modo ambas partes podrán entender la estructura y el contenido. En el caso el cliente puede enviar solicitudes al servidor en formato XML.

Estas dos primeras líneas especifican la versión de XML y la codificación utilizada UTF-8 en este caso y el Project es el elemento que contiene toda la información relacionada con el proyecto.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://ma
```

En la siguiente parte del código nos muestra la versión del modelo POM, groupId es el dominio inverso de la organización. Posteriormente muestra el nombre del proyecto, en este caso Cliente_Servidor seguido de la versión actual del proyecto, en este caso 1.0-SNAPSHOT. Por consiguiente especifica el tipo de empaquetado JAR (Java Archive)

```
<modelVersion>4.0.0</modelVersion>
<groupId>com.mycompany</groupId>
<artifactId>Cliente_Servidor</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
```

En estas líneas de código indica las dependencias del proyecto, en este caso tenemos una dependencia. El groupId muestra el identificador de la independencia, en este caso “xmlrpc”, por consiguiente muestra el nombre del módulo y la versión especifica de la dependencia. Finalmente, el tipo de archivo de la dependencia, en este caso un archivo JAR.

```
<dependencies>
    <dependency>
        <groupId>xmlrpc</groupId>
        <artifactId>xmlrpc</artifactId>
        <version>1.2-b1</version>
        <type>jar</type>
    </dependency>
</dependencies>
```

El siguiente bloque define las propiedades específicas del proyecto. La primera línea establece la codificación de origen del proyecto en UTF-8. Maven define la versión de Java utilizada para compilar el código fuente, posteriormente se muestra la versión de destino para la compilación. Finalmente exec especifica la clase principal (main que se

ejecutará) cuando se inicie el proyecto.

```
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <exec.mainClass>cliente_servidor.Cliente_Servidor</exec.mainClass>
    </properties>
</project>
```

2.3 Clase Servidor

La función principal de un servidor es proporcionar servicios a dispositivos clientes en una red.

Creamos la clase Servidor, pero al crearla debemos importar los chivos necesarios para la conexión.

```
import javax.swing.JOptionPane;

import org.apache.xmlrpc.WebServer;

//import org.apache.xmlrcp.WebServer;

public class Servidor {
```

Creamos el método main, aquí se inicia la ejecución del servidor.

```
public static void main (String[] args){
```

El bloque try-catch maneja excepciones. Si ocurre algún error dentro del bloque try, se captura y se maneja en el bloque catch (ese se mostrará al final del bloque). Posteriormente se muestra el mensaje en una ventana emergente “Iniciando el servidor” y se crea un objeto WebServer que escucha el puerto 8080. Aquí es donde el servidor aceptará conexiones entrantes.

```
try{
    //enviamos un mensaje
    JOptionPane.showMessageDialog(null, "Iniciando el servidor");

    //Creamos un objeto de tipo weserver
    WebServer server = new WebServer(8080);
```

Se crea un objeto SimuladorInfeccion, este objeto contiene la lógica para simular la

infección.

```
//Hacemos una relacion con nuestra parte simuladorInfección
SimuladorInfeccion simuladorinfecion = new SimuladorInfeccion();
```

En este caso se asocia el objeto simuladorinfecion con un identificador llamado “Miservidor”. Esto significa que cuando se reciba una solicitud con el nombre “Miservidor”, se invocará la lógica del simulador de infección.

```
server.addHandler("Miservidor", simuladorinfecion);
```

Inicializamos el servidor y se muestra una ventana emergente con el mensaje “Servidor en línea” para indicar que el servidor se ha iniciado correctamente.

```
server.start();

//Enviamos otro mensaje para ver si fue exito el inicio
JOptionPane.showMessageDialog(null, "Servidor en linea");
```

Si ocurre una excepción, por ejemplo, si el puerto está ocupado o hay un problema al iniciar el servidor, se captura y se muestra un mensaje de error que concatena el mensaje de la excepción (e.getMessage()).

```
        } catch (Exception e){
            //En caso que no se conecte el servidor se envia un mensaje y lo conectamos
            JOptionPane.showMessageDialog(null, "Error de conexión" + e.getMessage());
        }
    }

}
```

2.4 Clase Nodo

Esta clase hará la función del nodo

Creamos la clase Nodo

```
class Nodo
```

En el siguiente bloque de código se le pusieron comentarios a cada línea de código especificando la funcionalidad de cada uno.

```
{
    int id;//Almacenara nuestro identificador unico para cada nodo
    boolean infectado; //Declararemos una variable de instancia esto para indicar si el
                       //nodo esta infectado o no (true - false )
    List<Nodo> vecinos;//Declaramos otra variable instancida para almacenar las listas de nodos
                        //vecinos conectados a este nodo,se especifica que solo puede conoectar
                        //objetos de tipo 'Nodo'
    Nodo(int id){//Establecemos el constructor de la clase 'Nodo' tomando el argumento 'id'
        this.id=id;//Asignamos el valor parametro 'id' al campo 'id' del nodo
        this.infectado=false;//inicializamos el nodo en falso significando que aun no esta infectado
        this.vecinos= new ArrayList<>();//Le asignamos que inicialice la lista de vecinos como una nueva
                                        //instancia
    }
}
```

Creamos una clase llamada SimuladorInfeccion. Creamos un método llamado simularInfeccion que recibe dos parámetros (nodos y enfoque). Posteriormente se crea una instancia de la clase Random. Esto lo utilizamos para generar números aleatorios. Con la siguiente línea Nodo infectado = nodos.get(rand.nextInt(nodos.size())); Se selecciona un nodo aleatorio de la lista de nodos y se genera un índice aleatorio dentro del rango de tamaño de la lista de nodos donde el nodo infectado se almacena en la variable infectado. Posteriormente se marca el nodo seleccionado como “infectado” estableciendo su campo infectado en true. Por consiguiente se crea una nueva lista llamada infectados que inicialmente se encuentra vacía en la cual se agregaran la lista de nodos infectados.

```
public class SimuladorInfeccion {
    //Este metodo nos permitira tomar dos parametros de una lista de  nodos
       static void simularInfeccion(List<Nodo> nodos, String enfoque) {
        Random rand = new Random();//se instanc
        Nodo infectado = nodos.get(rand.nextInt(nodos.size()));
        infectado.infectado = true;
        List<Nodo> infectados = new ArrayList<>();
        infectados.add(infectado);
```

Posteriormente entramos al bucle while, que se ejecutará mientras la lista infectados no está vacía. Continuando se declara una variable llamada nodoActual y se inicializa con null.

Después de eso ingresamos a la condicional. Dependiendo del valor de enfoque, se selecciona un nodo de la lista infectados para procesarlo. Si el enfoque es “push” o “pull-push” se selecciona el primer nodo de la lista infectados utilizando infectados.remove(0). El nodo seleccionado se asigna a la variable nodoActual. Si el enfoque es “pull” se selecciona el último nodo de la lista infectados utilizando infectados.remove(infectados.size() - 1). El nodo seleccionado se asigna a la variable nodoActual

```
while (!infectados.isEmpty()) {
    Nodo nodoActual = null;
    if (enfoque.equals("push") || enfoque.equals("pull-push")) {
        nodoActual = infectados.remove(0);
    } else if (enfoque.equals("pull")) {
        nodoActual = infectados.remove(infectados.size() - 1);
    }
```

La instrucción assert se utiliza para verificar una condición y lanzar una excepción. En este caso, verifica si el nodoActual no sea nulo antes de continuar con el procesamiento

```
assert nodoActual != null;
```

La siguiente instrucción muestra una ventana emergente con el mensaje Nodo infectado + el id del nodo

```
JOptionPane.showMessageDialog(null,"Nodo"+nodoActual.id+"infectado.");
//System.out.println("Nodo " + nodoActual.id + " infectado.");
```

El bucle for Itera a través de los vecinos del nodoActual para cada vecino, si el vecino no está infectado (!vecino.infectado), se marca como infectado (vecino.infectado = true) y se agrega a la lista de nodos infectados (infectados.add(vecino)).

```
for (Nodo vecino : nodoActual.vecinos) {
    if (!vecino.infectado) {
        vecino.infectado = true;
        infectados.add(vecino);
    }
}
```

Posteriormente tenemos el if (enfoque.equals("pull-push"). Si el valor de enfoque es “pull-push”, se realiza un bucle similar para los vecinos del nodoActual. Esto significa que, en el caso de “pull-push”, los vecinos también se marcan como infectados y se agregan a la lista de nodos infectados.

```
    if (enfoque.equals("pull-push")) {
        for (Nodo vecino : nodoActual.vecinos) {
            if (!vecino.infectado) {
                vecino.infectado = true;
                infectados.add(vecino);
            }
        }
    }
}
```

Después declaramos una variable llamada totalInfectados y la inicializamos en. Esta variable se utilizará para contar cuántos nodos están infectados. El bucle for va a iterar a través de cada no en la lista nodos, si el nodo está infectado se incrementa el contador totalInfectados

```
int totalInfectados = 0;
for (Nodo nodo : nodos) {
    if (nodo.infectado) {
        totalInfectados++;
    }
}
```

Finalmente se muestra una ventana emergente con un mensaje que incluye el enfoque y la cantidad total de nodos infectados concatenados con el valor de enfoque para indicar qué enfoque se utilizó.

```
        JOptionPane.showMessageDialog(null,"Enfoque"+enfoque+"- Nodos infectados: "+totalInfectados);
        //System.out.println("Enfoque: " + enfoque + " - Nodos infectados: " + totalInfectados);

    }
}
```

2.5 Pom.xml-Servidor

Como se mencionó anteriormente el archivo POM (Project Object Model) en formato XML nos permite transmitir los datos entre el cliente y el servidor, de este modo ambas partes podrán entender la estructura y el contenido. En el caso del servidor procesa estas solicitudes y devuelve respuestas también en el formato XML.

La primera línea específica la versión de XML y la codificación utilizada UTF-8. Project es

elemento raíz del archivo POM. Contiene toda la información relacionada con el proyecto. En este caso se utiliza la versión 4.0.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

Posteriormente se muestra el groupId que es el dominio inverso de la organización. Posteriormente muestra el nombre del proyecto, en este caso simulador-infeccion seguido de la versión actual del proyecto. Finalmente se especifica el tipo de empaquetado JAR (Java Archive)

```
<groupId>com.example</groupId>
<artifactId>simulador-infeccion</artifactId>
<version>1.0.0</version>
<packaging>jar</packaging>
```

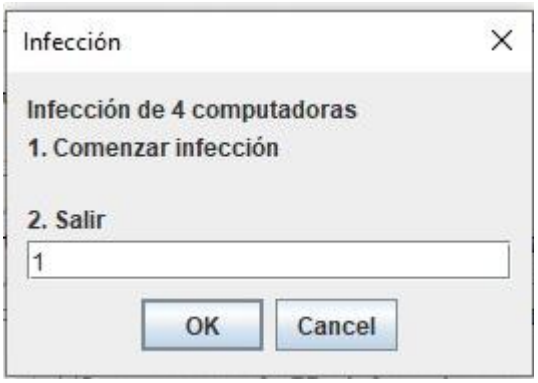
Después definimos las dependencias, en este caso dos. La primera la biblioteca XML-RPC cliente de Apache con la versión 3.1.3. y la segunda biblioteca XML-RPC con la versión 1.2.

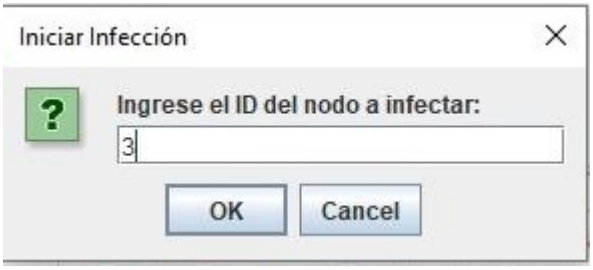
```
<dependencies>
    <dependency>
        <groupId>org.apache.xmlrpc</groupId>
        <artifactId>xmlrpc-client</artifactId>
        <version>3.1.3</version>
    </dependency>
    <dependency>
        <groupId>xmlrpc.server</groupId>
        <artifactId>xmlrpc</artifactId>
        <version>1.2</version>
        <type>jar</type>
    </dependency>
</dependencies>
```

Finalmente build nos permitirá la configuración para el proyecto. Se especifica un complemento plugin para el empaquetado JAR y para incluir la clase principal Main.

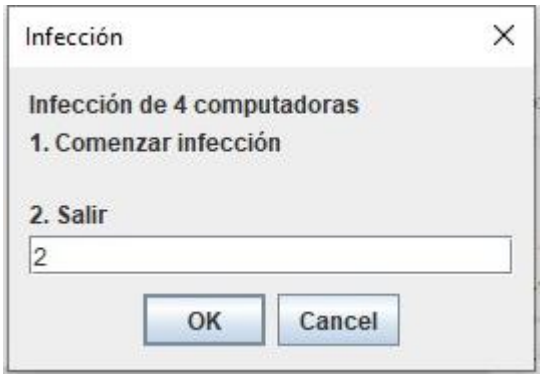
```
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <version>3.2.0</version>
                <configuration>
                    <archive>
                        <manifest>
                            <addClasspath>true</addClasspath>
                            <mainClass>com.example.Main</mainClass>
                        </manifest>
                    </archive>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

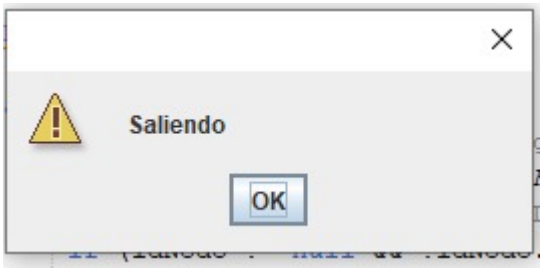
2.6 Resultados





En esos momentos fallo la infección, sin embargo si se estaban infectando los nodos.





3. Conclusiones

Se estuvieron considerando diferentes formas de llevar a cabo esta práctica, sin embargo consideramos que esta fue una de las más viables y sencillas para poder entender la funcionalidad de la difusión de datos basado en chismes. En nuestro caso pudimos hacer la infección de manera local recordando que solo es una simulación. Se hizo uso de dos clases principales, una que hace la funcionalidad del cliente y la otra la del servidor, además se creó una clase nodo, que funge como los nodos a infectar. Finalmente se crearon los archivos Pom XML que permitirán que se lleve a cabo la comunicación entre el cliente y el servidor. La infección de nodos es importante ya que esto puede representar la propagación de la información.

4. Referencia

Microsoft Build. (Mayo 23, 2024). Arquitectura de aplicación de formato XML en el cliente y en el servidor (SQLXML 4.0). Arquitectura de XML de cliente y servidor (SQLXML) - SQL Server | Microsoft Learn

Hostinger. (Abril 27, 2023). ¿Qué es un servidor web y cómo funciona?. ¿Qué es un servidor web y cómo funciona? - Guía completa (hostinger.mx)

Avalion. (Enero 22, 2023). ¿QUÉ ES UN SERVIDOR Y CUÁL ES SU FUNCIÓN?. ¿Qué es un servidor y cuál es su función? - Avalion

Mdn web docs. (s.f). try...catch. try...catch - JavaScript | MDN (mozilla.org)