UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA:

INGENIERÍA DE SOFTWARE

NRC:

2546

ASIGNATURA:

APLICACIONES DISTRIBUIDAS

DOCENTE:

DARIO MORALES

TEMA:

SINCRONIZACIÓN CON JAVA

NOMBRE:

RAMOS ADRIAN

FECHA:

22/01/2025

Informe sobre Algoritmos de Sincronización de Relojes: Berkeley, Cristian y NTP

1. Introducción

La sincronización de relojes es crucial en sistemas distribuidos para garantizar la consistencia de datos, la coordinación de eventos y la correcta ejecución de procesos concurrentes. Este informe explora tres algoritmos clave utilizados para este propósito: el algoritmo de Berkeley, el algoritmo de Cristian y el Protocolo de Tiempo de Red (NTP). A través del análisis teórico y la implementación de estos algoritmos, se busca comprender cómo funcionan, sus ventajas, desventajas y aplicaciones.

2. Marco Teórico

- **Sincronización de relojes en sistemas distribuidos**
  En sistemas distribuidos, cada nodo o máquina tiene su propio reloj local, que puede desviarse de los demás debido a factores como la deriva del hardware y las variaciones en la latencia de la red. La sincronización busca minimizar estas diferencias para garantizar una visión coherente del tiempo en todos los nodos.

- **Algoritmo de Berkeley**
  Propuesto por Bruce J. Lubachevsky, el algoritmo de Berkeley es utilizado en entornos donde no se dispone de un reloj central de referencia. El nodo maestro recopila los tiempos de los nodos esclavos, calcula un promedio y ajusta los relojes de todos los nodos en función de este promedio.

- **Algoritmo de Cristian**
  Este algoritmo asume la existencia de un servidor confiable que actúa como referencia de tiempo. Los clientes envían solicitudes al servidor, que devuelve su hora actual. Los clientes ajustan su reloj considerando la latencia estimada.

- **Protocolo de Tiempo de Red (NTP)**
  NTP es un estándar ampliamente adoptado para sincronización de relojes en redes. Utiliza una jerarquía de servidores para proporcionar una referencia de tiempo precisa y minimizar el error causado por la latencia.

3. Objetivo General

Implementar y analizar el comportamiento de los algoritmos de Berkeley, Cristian y NTP, evaluando sus ventajas y desventajas en la sincronización de relojes en sistemas distribuidos.

4. Objetivos Específicos

- Implementar los tres algoritmos de sincronización de relojes: Berkeley, Cristian y NTP, utilizando herramientas y bibliotecas en Java.
- Desarrollar una aplicación para cada algoritmo que permita realizar la sincronización en un entorno simulado.
- Comprender el funcionamiento de cada algoritmo y sus diferencias.

5. Desarrollo

5.1. Algoritmo de Berkeley

- **Descripción:**

El algoritmo de Berkeley utiliza un nodo maestro que calcula un tiempo promedio basado en las horas enviadas por los nodos esclavos. Luego, ajusta los relojes locales de los nodos para alinearlos con este tiempo promedio.

- **Código Implementado:**
  Se diseñó un nodo maestro y dos nodos esclavos. El maestro calcula el tiempo promedio y envía las diferencias necesarias para ajustar los relojes locales.

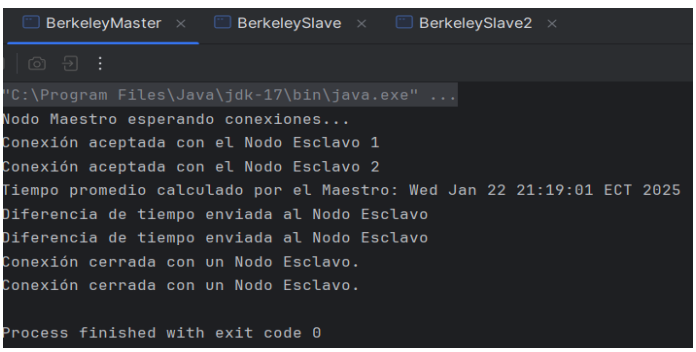
Maestro:

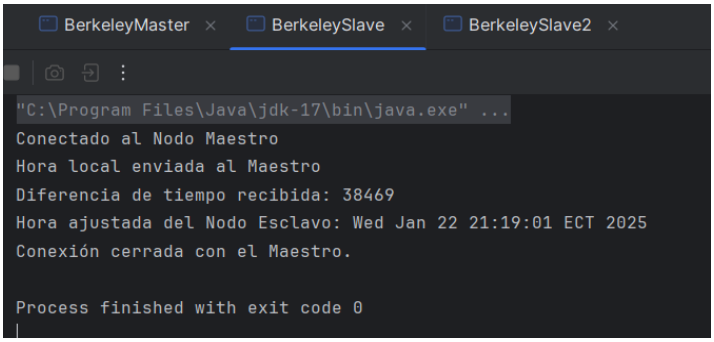
```java
package com.espe.Berkeley;

import ...

public class BerkeleyMaster {
    public static void main(String[] args) {
        int port = 12345; // Puerto de comunicación
        int numSlaves = 2; // Número de nodos esclavos esperados (ajustado a 1)
        List<Long> slaveTimes = new ArrayList<>();
        List<Socket> slaveSockets = new ArrayList<>();

        ExecutorService executor = Executors.newFixedThreadPool(numSlaves);

        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Nodo Maestro esperando conexiones...");

            // Aceptar conexiones de los nodos esclavos
            for (int i = 0; i < numSlaves; i++) {
                Socket slaveSocket = serverSocket.accept();
                slaveSockets.add(slaveSocket);
                System.out.println("Conexión aceptada con el Nodo Esclavo " + (i + 1));

                // Leer el tiempo del esclavo en un hilo separado
                executor.submit(() -> {
                    try {
                        DataInputStream dataIn = new DataInputStream(slaveSocket.getInputStream());
                        long slaveTime = dataIn.readLong(); // Leer el tiempo enviado por el esclavo
                        synchronized (slaveTimes) {
                            slaveTimes.add(slaveTime);
                        }
                    } catch (IOException e) {
                        System.err.println("Error leyendo datos del esclavo: " + e.getMessage());
                    }
                });
            }

            // Esperar a que todos los esclavos envíen su tiempo
            while (slaveTimes.size() < numSlaves) {
                Thread.sleep( millis: 1000); // Espera breve para evitar bloqueos
            }

            // Calcular el tiempo promedio
            long totalTime = slaveTimes.stream().mapToLong(Long::longValue).sum();
            long averageTime = totalTime / numSlaves;
            System.out.println("Tiempo promedio calculado por el Maestro: " + new Date(averageTime));

            // Enviar la diferencia de tiempo a los nodos esclavos
            for (int i = 0; i < slaveSockets.size(); i++) {
                Socket slaveSocket = slaveSockets.get(i);
                long slaveTime = slaveTimes.get(i);

                executor.submit(() -> {
                    try {
                        DataOutputStream dataOut = new DataOutputStream(slaveSocket.getOutputStream());
                        long timeDifference = averageTime - slaveTime;
                        dataOut.writeLong(timeDifference); // Enviar la diferencia al esclavo
                        dataOut.flush();
                        System.out.println("Diferencia de tiempo enviada al Nodo Esclavo");
                    } catch (IOException e) {
                        System.err.println("Error enviando datos al esclavo: " + e.getMessage());
                    }
                });
            }

            // Cerrar las conexiones con los nodos esclavos
            executor.shutdown();
            executor.awaitTermination( timeout: 5, TimeUnit.SECONDS);
            for (Socket slaveSocket : slaveSockets) {
                slaveSocket.close();
                System.out.println("Conexión cerrada con un Nodo Esclavo.");
            }

        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

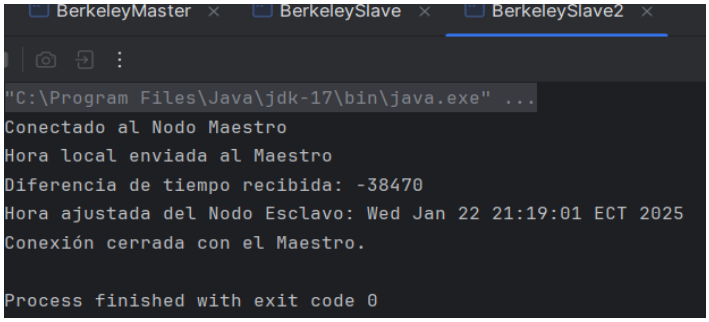
Esclavo:

```java
package com.espe.Berkeley;

import ...

public class BerkeleySlave {
    public static void main(String[] args) {
        String masterAddress = "localhost"; // Dirección del nodo maestro
        int port = 12345; // Puerto del nodo maestro

        try {
            Socket socket = new Socket(masterAddress, port);
            System.out.println("Conectado al Nodo Maestro");

            // Enviar la hora local al maestro
            long localTime = System.currentTimeMillis();
            DataOutputStream dataOut = new DataOutputStream(socket.getOutputStream());
            dataOut.writeLong(localTime);
            dataOut.flush();
            System.out.println("Hora local enviada al Maestro");

            // Mantener el socket abierto y recibir la diferencia de tiempo
            DataInputStream dataIn = new DataInputStream(socket.getInputStream());
            while (true) {
                try {
                    // Leer la diferencia de tiempo del maestro
                    long timeDifference = dataIn.readLong();
                    System.out.println("Diferencia de tiempo recibida: " + timeDifference);

                    // Ajustar el reloj local con la diferencia recibida
                    long adjustedTime = localTime + timeDifference;
                    Date adjustedDate = new Date(adjustedTime);
                    System.out.println("Hora ajustada del Nodo Esclavo: " + adjustedDate);

                    // Salir del bucle si no se espera más comunicación
                    break;
                } catch (EOFException e) {
                    // Manejar el cierre del flujo por parte del maestro
                    System.err.println("El maestro cerró la conexión.");
                    break;
                }
            }

            // Cerrar recursos manualmente
            dataOut.close();
            dataIn.close();
            socket.close();
            System.out.println("Conexión cerrada con el Maestro.");

        } catch (IOException e) {
            System.err.println("Error en el Nodo Esclavo: " + e.getMessage());
        }
    }
}
```

Ejecución:







Ventajas:

- No requiere un reloj central externo.
- Es útil en redes privadas y sistemas sin acceso a servidores NTP.

Desventajas:

- Sensible a la latencia de red.
- Menos preciso que NTP en grandes redes.

5.2. Algoritmo de Cristian´s

- **Descripción:**

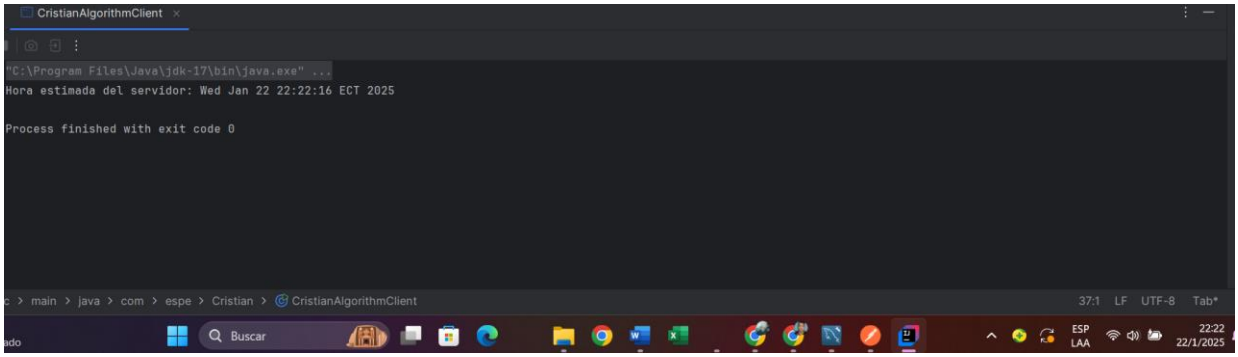
Este algoritmo se basa en un servidor confiable que responde con su hora actual. El cliente calcula la latencia y ajusta su reloj en consecuencia.

Código Implementado:

```
public class CristianAlgorithmClient {
    public static void main(String[] args) {
        // Dirección del servidor NTP al que se conectará para obtener la hora
        String serverAddress = "time.google.com"; // Servidor NTP público

        // Cliente NTP que se encargará de comunicarse con el servidor
        NTPUDPClient timeClient = new NTPUDPClient();

        try {
            // Obtener la dirección IP del servidor NTP a partir de su nombre de dominio
            InetAddress timeServerAddress = InetAddress.getByName(serverAddress);

            // Solicitar la hora al servidor NTP y obtener la respuesta
            TimeInfo timeInfo = timeClient.getTime(timeServerAddress);

            // Registrar el tiempo actual en el cliente antes de enviar la solicitud al servidor (T1)
            long T1 = System.currentTimeMillis();
```

```java
            // Obtener el tiempo del servidor cuando procesó la solicitud (T2 y T3 son iguales para el servidor NTP)
            long T2 = timeInfo.getMessage().getTransmitTimeStamp().getTime();
            long T3 = T2; // Tiempo del servidor transmitido en la respuesta (lo mismo que T2)

            // Registrar el tiempo actual en el cliente después de recibir la respuesta del servidor (T4)
            long T4 = System.currentTimeMillis();

            // Calcular el tiempo de ida y vuelta (Round-Trip Time, RTT)
            long RTT = T4 - T1;

            // Calcular la latencia (delay) como la mitad del RTT
            long delay = (T3 - T2) / 2;

            // Calcular la hora estimada del servidor ajustada con el retraso
            long estimatedServerTime = T3 + delay;

            // Calcular el tiempo sincronizado en el cliente ajustando también el RTT
            long synchronizedTime = estimatedServerTime + (RTT / 2);

            // Convertir el tiempo sincronizado a un objeto Date para su representación
            Date date = new Date(synchronizedTime);

            // Mostrar la hora sincronizada estimada del servidor
            System.out.println("Hora estimada del servidor: " + date);

        } catch (Exception e) {
            // Imprimir cualquier excepción ocurrida durante la conexión o sincronización
            e.printStackTrace();
        }
    }
}
```

Ejecución:



Ventajas:

- Sencillo de implementar.
- Adecuado para sincronizar pocos nodos.

Desventajas:

- Dependencia de un servidor externo.
- Sensible a la variabilidad en la latencia.

5.3. Protocolo de Tiempo de Red (NTP)

- Descripción:

NTP sincroniza el tiempo entre nodos utilizando una jerarquía de servidores. Considera la latencia de la red y ajustes de error acumulativo.

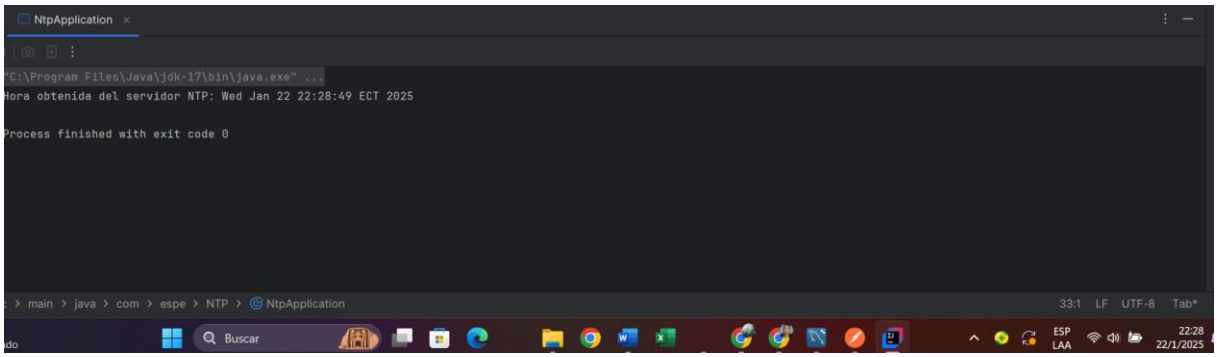
Código Implementado:

Se usó un cliente NTP para conectarse a un servidor público (time.google.com).

```java
@SpringBootApplication
public class NtpApplication {
    public static void main(String[] args) {
        // Dirección del servidor NTP al que se conectará para obtener la hora actual
        String timeServer = "time.google.com"; // Servidor NTP público reconocido

        // Cliente NTP que se encargará de enviar la solicitud y recibir la respuesta
        NTPUDPClient timeClient = new NTPUDPClient();

        try {
            // Obtener la dirección IP del servidor NTP utilizando su nombre de dominio
            InetAddress timeServerAddress = InetAddress.getByName(timeServer);

            // Enviar la solicitud al servidor NTP y obtener la respuesta
            TimeInfo timeInfo = timeClient.getTime(timeServerAddress);

            // Extraer el tiempo transmitido por el servidor NTP en formato de milisegundos
            long currentTimeMillis = timeInfo.getMessage().getTransmitTimeStamp().getTime();
```

```java
            // Extraer el tiempo transmitido por el servidor NTP en formato de milisegundos
            long currentTimeMillis = timeInfo.getMessage().getTransmitTimeStamp().getTime();

            // Convertir el tiempo obtenido a un objeto Date para facilitar su visualización
            Date time = new Date(currentTimeMillis);

            // Mostrar la hora obtenida del servidor NTP
            System.out.println("Hora obtenida del servidor NTP: " + time);

        } catch (Exception e) {
            // Capturar y mostrar cualquier excepción ocurrida durante la conexión o procesamiento
            e.printStackTrace();
        } finally {
            // Asegurarse de cerrar el cliente NTP para liberar los recursos
            timeClient.close();
        }
    }
}
```

Ejecución:



Ventajas:

- Alta precisión.
- Escalable a grandes redes.

Desventajas:

- Requiere acceso a servidores NTP confiables.

6. Conclusión

La sincronización de relojes es fundamental para garantizar la coherencia temporal en sistemas distribuidos. Cada algoritmo tiene aplicaciones específicas:

- **Berkeley:** Ideal para redes privadas y sistemas sin acceso externo.
- **Cristian:** Útil para sincronización sencilla entre pocos nodos.

- **NTP:** Opción robusta y escalable para sistemas de gran escala.

La elección del algoritmo dependerá de factores como el tamaño de la red, la necesidad de precisión y la disponibilidad de recursos externos.

Referencias Bibliográficas

1. Lubachevsky, B. J. (1985). Berkeley Algorithm for Clock Synchronization.
2. Cristian, F. (1989). Probabilistic Clock Synchronization.
3. Mills, D. L. (1985). Network Time Protocol (NTP). RFC 958.
4. Apache Commons Net Library. https://commons.apache.org/proper/commons-net/

Link GitHub
https://github.com/adrianmanu/Sincronizacion_Java