Memoria técnica del proyecto

Programación de Sistemas Concurrentes y Distribuidos

2º curso, Grado de Ingeniería en Informática, 2021-2022

Número de equipo: *6*

Integrantes:

- *Axel Isaac Pazmiño Ortega, 817627*
- *Alicia Lázaro Huerta, 820574*
- *Francisco Javier Pizarro Martínez, 821259*
- *Leonor Murphy Anía, 798368*
- *Pablo López Mosqueda, 779739*
- *Inés Román Gracia, 820731*

Índice de contenidos

1. Introducción

El problema a resolver se compone de una arquitectura master-worker para analizar una colección de tweets y extraer información relevante para el usuario. El sistema planteado se compone de un servicio de streaming que filtra los tweets y los manda a los procesos master worker, el master crea tareas en una cola del gestor de colas y los worker, reciben y procesan los tweets de la cola. Después, los worker van encolando la información procesada en el gestor de colas y por último los analizadores reciben la información desde las colas, la analizan y la muestran al usuario.

Nuestro equipo se compone de 6 personas: Axel Isaac Pazmiño, Alicia Lázaro, Francisco Javier Pizarro, Leonor Murphy, Pablo López e Inés Román.

Este documento se compone de las siguientes partes:

- Un diseño en alto nivel en el que se explican los distintos servicios que componen el proyecto
- Las decisiones de diseño más importantes y críticas a la hora de implementar los servicios
- La forma de evaluar el resultado final del proyecto
- Como se ha planificado y gestionado tanto los contenidos del proyecto como el tiempo para los mismos
- Las conclusiones y posibles mejoras futuras para optimizar el proyecto o añadir nuevas funcionalidades

2. Diseño de alto nivel de la solución

En la figura 2.1 se puede apreciar el diseño de alto nivel que se ha seguido por el equipo.

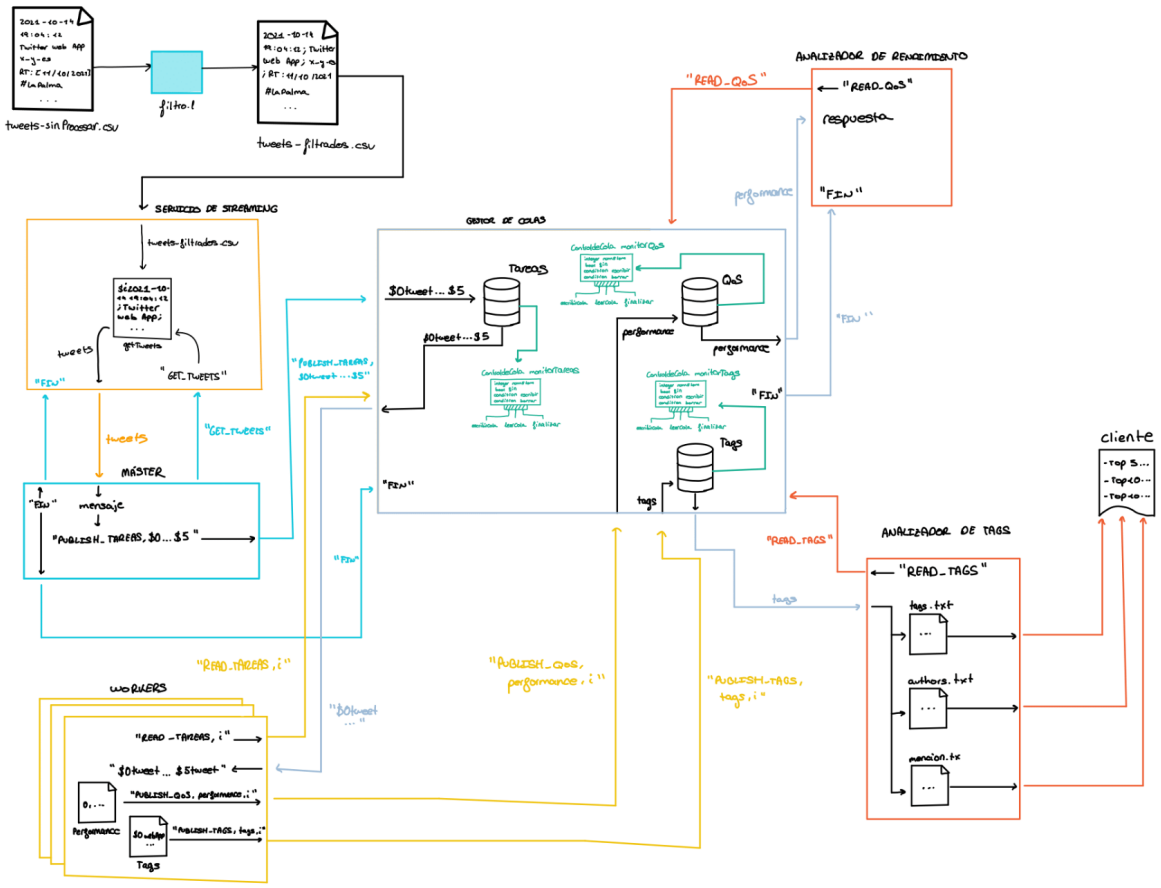

Figura 2.1

2.1 El servicio de Streaming

En primer lugar, en la parte superior izquierda de la imagen nos encontramos con el servicio de streaming. Primero se filtra el archivo tweets-sinProcesar.csv, y devuelve otro fichero, tweets - filtrados.csv que es un fichero como el original pero con los emoticonos o caracteres no esperados borrados. Después, el servicio de streaming lee 25 tweets del fichero tweets-filtrados, procesa los tweets eliminando los caracteres extraños y aquellos tweets que no contengan los campos necesarios y los envía al proceso master hasta que reciba un mensaje de fin de parte del mismo.

Su diseño a alto nivel es el siguiente:

```
channel of string operation
Process streaming
        string pedido operation => (pedido)
        while(pedido != “FIN”)
                if (pedido = “GET_TWEETS”)
                string mensaje
                getTweets(mensaje)
                        tweets <= (mensaje)
                end if
                operation => (pedido)
        end while
end process
```

2.2 Los procesos master-worker

En la parte inferior izquierda de la figura nos encontramos con los procesos master-worker.

Este modelo consta de dos tipos de proceso: el proceso master que se encarga de enviar las tareas y un conjunto de procesos workers que recibirán las tareas y se encargarán de llevarlas a cabo. El proceso master se encarga de pedir los tweets a procesar y los recibe directamente del servicio de streaming, luego los envía al gestor de colas que los almacenará en la cola de tareas hasta que los workers empiecen a procesarlos.

Como resultado de este procesamiento de mensajes, los workers enviarán la información obtenida sobre la calidad de servicio y de los tags al gestor de colas y este los almacenará en sus colas de QoS y de Tags respectivamente. Además, será el proceso master el que se encargará de cerrar la comunicación. Se enviará un mensaje de fin al servicio de streaming y otro al gestor de colas, después este se encargará de informar a los workers y a los analizadores del cierre de la comunicación.

Los procesos workers estarán continuamente pidiendo nuevos tweets para procesar al gestor de colas y enviarán la información obtenida hasta que reciban el mensaje de fin del gestor de colas. Finalmente enviarán al gestor de colas la información sobre el rendimiento de los propios procesos worker e información sobre los tweets que servirá a los procesos analizadores.

Su diseño a alto nivel se detalla a continuación:

```
channel of (string) [1..N_WORKERS]worker_gestor
channel of (string) master_gestor, master_streaming

Process worker i := (1..N_WORKERS)
        string recibido, datosTags, datosQoS
        while(!finalizado)
                worker_gestor(i) <- “READ_TAREAS, i”
                worker_gestor(i) -> recibido
                if(recibido = “FIN”)
                        finalizado := true
                else
                        datosTags := procesar_tags()
                        datosQoS := procesar_QoS()
                        worker_gestor(i) <- “PUBLISH_TAGS, datosTags”
                        worker_gestor(i) <- “READ_QOS, datosQoS”
                end if
        end while
end process

Process master
        string bloqueTweets, tweets
        for i := 0..N_VECES
                master_streaming <- “GET_TWEETS”
                master_streaming -> bloqueTweets
                tweets := procesar_tweets()
                master_gestor <- tweets
        end for
master_streaming <- “FIN”
master_gestor <- “FIN
end process
```

2.3 El gestor de colas

El siguiente bloque es el del gestor de colas, representado en la figura 2.1 con un rectángulo gris. El gestor de colas proporcionará las herramientas necesarias para almacenar y gestionar todos los datos del resto de procesos. Primero, es necesario almacenar los datos que el resto de procesos envían. Para ello, se utiliza la cola genérica BoundedQueue ya dada. Para evitar el solapamiento de entrada y salida de los datos en las colas, se usa un monitor cuyo esquema en alto nivel es el siguiente:

```
Monitor ControldeCola
        integer numElementos := 0
        boolean fin := false
        condition estaEscribiendo
        condition estaBorrando
        operation escribirCola( BoundedQueue cola, string elemento )
                while (numElementos >= MAX_ELEMENTOS)
                        waitC( estaEscribiendo )
                end while
                cola.encolar(elemento)
                numElementos := numElementos + 1
                signalC (estaBorrando)
        end operation
        operation leerCola(BoundedQueue cola, string elemento)
                boolean leido = false
                while (numElementos <= 0 && !fin)
                        WaitC(estaBorrando)
                end while
                if (numElementos > 0)
                        numElementos - -
                        cola.desencolar(elemento)
                        signalC(estaEscribiendo)
                        leido= true;
                end if
                return leido
```

```
        end operation
        operation finalizar()
                fin := true
                signalC_all( estaBorrando )
        end operation
end monitor
```

El monitor está diseñado de tal forma que para cada cola que se necesite, se invoque un monitor. Este cuenta con dos variables condición y dos operaciones. Si se quiere escribir en la cola primero se comprueba que la cola no esté llena y si no lo está se añade el elemento deseado al final de la cola. Para leer un elemento de la cola se comprueba primero que la cola no es vacía y si no lo es se guarda el primer elemento y después se desencola. Si fuera vacía, se devuelve falso en un booleano de control. El programa principal crea dos sockets: uno para el master y los workers y otro para los analizadores. A continuación lanza N + 1 procesos master/worker (siendo N el número máximo de procesos que se pueden conectar) y dos procesos analizadores. A continuación se detalla el funcionamiento de ambos tipos de proceso:

```
Channel of string socTareas, socAnalizadores
ControldeCola controlTareas, controlTags, controlQoS
BoundedQueue colaTareas, colaTags, colaQoS
Process masterWorker
        string mensaje
        bool out := false
        while (!out)
                socTareas => mensaje
                if (datos = “FIN”)
                        out := true
                else if (mensaje = “PUBLISH_TAREAS,datos”)
                        controlTareas.escribirCola(colaTareas, datos)
                else if (mensaje = “READ_TAREAS”)
                        if (controlTareas.leerCola(colaTareas, datos) )
                                socTareas <= datos else mensaje := “FIN” socTareas <= mensaje
                        end if
                else if (mensaje = “PUBLISH_QoS”)
```

```
                controlQoS.escribirCola(colaQoS, datos)
        else if (mensaje == “PUBLISH_TAGS”)
                controlTags.escribirCola(colaTags, datos)
        end if
    end while
end process
```

El proceso masterWorker es el encargado de gestionar la comunicación con los procesos master/worker dependiendo de lo que escuche por su canal. Por ejemplo si un proceso worker quiere leer tareas pendientes, se desencolará de la cola de tareas los datos correspondientes y se le enviarán estos. Si no quedaran tareas pendientes se le envía un mensaje de fin y se finaliza la comunicación.

```
Process analizadores
    string mensaje
    boolean out := false
    while(!out)
        socAnalizadores => mensaje
        if (mensaje = “READ_QoS”)
            if (controlQoS.leerCola(colaQoS, mensaje))
                socAnalizadores <= mensaje
            else
                mensaje := “FIN”
                socAnalizadores <= mensaje
                out := true
            end if
        else if (mensaje =“READ_TAGS”)
            if (controlTags.leerCola(colaTags, mensaje)
                socAnalizadores <= mensaje
            else
                mensaje := “FIN”
                out := true
            end if
        end if
    end while
end process
```

El proceso analizadores se encarga de recibir peticiones de lectura por parte de los analizadores y enviar los datos correspondientes. Por ejemplo: si un analizador quiere leer un dato de la cola de QoS, enviará a través del canal socAnalizadores el mensaje “READ_QoS” y el gestor de colas desencolar el primer elemento que haya en la cola de QoS y se lo enviará al analizador.

2.4 Los analizadores

El analizador de Tags proporcionará información de los tweets desde diferentes perspectivas.

-> Visión global:

. Top 5 Clientes más utilizados para acceder a twitter

· Top 5 Clientes más utilizados para escribir tweets

· Top 5 Clientes más utilizados al escribir tweets con tags

· Top 5 Clientes más utilizados al escribir tweets con menciones

· Top 5 Clientes más utilizados para hacer retweet

· Porcentaje de ocupación del cliente

-> Visión desde los tags:

· Top 10 Hashtags más utilizados durante la vida del sistema

· Porcentaje de apariciones del Hashtag del número total de tweets

· Porcentaje de apariciones del Hashtag del número total de Hashtags

-> Visión desde los usuarios:

· Top 10 usuarios con más actividad - Visión desde las menciones:

· Top 10 autores con más menciones

· Porcentaje de apariciones de la mención del número total de tweets

· Porcentaje de apariciones de la mención del número total de Hashtags

El analizador de tags se comunica con el gestor de colas y recibe la información a analizar y la trata mediante el siguiente proceso:

```
channel of string socGestor
Process analizadorTags
        string mensaje
        boolean out := false
        while(!out)
                mensaje := “READ_TAGS”
                socGestor <= mensaje
                socGestor => mensaje
```

```
            if(mensaje = “FIN”)
                out := true
            else
                procesar()          // trocea el mensaje recibido, produce los resultados
                                    // de la información respecto a cada cliente
                                    // (información numérica) guardándolo en variables.
                almacenarTxt()      // trocea el mensaje dejando los hashtags, los manda a
                                    // su fichero. Lo mismo para el autor y las menciones.
            end if
        end while

        tagsInformation()           // cliente que procesa su fichero de texto y muestra la
                                    // información  relativa a los hashtags
        authorsInformation()        //cliente que procesa su fichero de texto y muestra la
                                    // información relativa a los autores.
        mencionInformation()        // cliente que procesa su fichero de texto y muestra la
                                    // información relativa a las menciones. end process
end process
```

El analizador de rendimiento pretende mostrar los siguientes resultados: - tiempo medio que le cuesta resolver a cada worker 200 bloques de tareas - gráfico temporal (para cada worker) tiempo de ejecución cada 200 bloques - worker más lento y el más rápido (rango en que oscila el tiempo de ejecución medio) - bloque más lento y más rápido El analizador de rendimiento se comunica con el gestor de colas y recibe la información a analizar y la trata mediante el siguiente proceso:

```
channel of string socGestor
Process analizadorQoS
        string mensaje boolean out := false
        while(!out) mensaje := “READ_QoS”
            socGestor <= mensaje
            socGestor => mensaje
            if(mensaje = “FIN”)
                out := true
            else
```

```
                procesarRendimiento()
            end if
        end while
end process
```

Al cerrar la conexión el analizador de rendimiento mostrará toda la información procesada por pantalla.

3. Decisiones de diseño relevantes

Para el masterworker se ha usado un semáforo para gestionar un pequeño problema de concurrencia que se presentó al realizar la comunicación en la función connect() con el gestor de colas.

En el gestor de colas se ha usado un monitor. Se presentaban dos posibles soluciones o un monitor grande que gestionará la concurrencia de las tres colas a la vez. Se decidió usar un monitor general para las colas ya que así se supuso que sería más simple y más fácil a la hora comprobar los posibles errores que pudieran aparecer. Este monitor se compone de las siguientes operaciones:

- escribirCola(), que encola un elemento en la cola y actualiza el número de elementos en esta. Si no hay espacio en la cola, se añade una señal de wait a la condición estaEscribiendo.
- leerCola(), que desencola un elemento de la cola y actualiza el número de elementos de esta. Se añade una señal de signal a la condición de estaEscribiendo. Devuelve true o false dependiendo de si ha leído un elemento. Si no hay elementos en la cola, se añade una señal de wait a la condición estaBorrando.
- finalizar(), asigna true a la variable fin y le da señal de signal a todos los estaBorrando. Esta operación tuvo que ser añadida debido a un problema con los monitores.

El monitor se usa para que dos procesos worker no puedan escribir a la vez en la cola, ya que podría provocar que se escribieran más datos en la cola de las que esta puede soportar, o, en el caso de que dos worker desencolaran un dato a la vez, la cola podría tener un número negativo de elementos u otro tipo de problemas.

Nuestro proyecto presenta dos servidores distintos, por un lado, el proceso Streaming consta de un proceso auxiliar que envía los datos al proceso master. Su ciclo de vida es mandar datos hasta que el proceso master le envía “FIN”.

El otro servidor es el gestor de colas. Este consta de dos procesos auxiliares y su ciclo de vida funciona de forma similar al servicio de Streaming:

Por un lado el proceso master worker se mantiene vivo hasta que no quedan datos en la cola, entonces manda “FIN” al proceso worker y muere.

Por otro, el proceso analizadores hace algo parecido, envía datos a los analizadores hasta que no quedan datos en la cola. En ese caso envía “FIN” y muere.

Para el analizador de tags, se emplean de manera auxiliar ficheros de texto para almacenar datos y después procesarlos, para procesar estos datos se emplea memoria dinámica mediante el uso de una estructura genérica de tipo árbol binario, para que sea eficiente en tiempo y memoria.Además dado que todo el sistema depende de que las colas del gestor tengan hueco, de esta forma evitamos que el analizador de tags ralentice todo el sistema procesando datos sobre la marcha.

Para el analizador de rendimiento al ser datos simples y sencillos se pueden procesar sobre la marcha sin necesidad de memoria dinámica sin ralentizar el sistema.

4. Evaluación del sistema final

Para probar el sistema se ha hecho de la siguiente forma: se han usado los ordenadores del laboratorio de manera remota. 4 equipos distintos, en hendrix01 servidor de Streaming, en hendrix02 el gestor de colas, en lab000 el masterworker y en central los analizadores, cada uno en una terminal distinta para visualizar mejor los resultados. Todas estas máquinas son sistemas basados en unix y están conectadas mediante la red de unizar.

Las condiciones de ejecución del sistema consisten en que se debe lanzar el sistema en el siguiente orden:

servicio de streaming -> gestor de colas -> masterWorkers -> analizadores

La pruebas de carga realizadas han sido con la siguiente carga de tweets 100, 2500 y 5000, funcionando en todos los casos correctamente.

5. Planificación y organización del trabajo

5.1 La gestión del proyecto

El proyecto se ha dividido en distintos grupos de trabajo:

Por un lado, el servicio de streaming y los analizadores los programaran Axel Pazmiño y Alicia Lázaro.

El servicio masterworker lo realizará Ines Román y Javier Pizarro.

Por último el gestor de colas lo programaran Leonor Murphy y Pablo López.

Además de programar el sistema correspondiente, cada grupo realizará la documentación de su parte. Se ha decidido separar de esta forma el proyecto porque se ha considerado que cada grupo tendría una carga de trabajo similar.

En un primer momento se pensó en tener acabada una primera versión funcional antes del día de Navidad, pero, debido a falta de tiempo, al final se decidió posponer esta primera versión al día de Reyes.

5.2 La dedicación individual

Los esfuerzos dedicados por cada miembro del equipo son los siguientes:

Nombre: Alicia Lázaro Huerta

Fecha dd/mm/aa	**Tiempo dedicado ***	**Tipo de tarea ****
15/12/2021	2 horas	análisis y diseño del sistema
18/12/2021	1h15m	diseño del sistema y programación
19/12/2021	1 hora	reuniones
02/01/2022	3h15m	programación
06/01/2022	30m	reuniones
07/01/2022	5 horas	programación
08/01/2022	1 hora	reuniones
09/01/2022	1 hora	programación
11/01/2022	2h30m	programación
13/01/2022	2 horas	programación
14/01/2022	1 hora	programación
15/01/2022	5h	programación
15/01/2022	30m	documentación y pruebas
Tiempo total dedicado		26h

Nombre completo: Axel Isaac Pazmiño Ortega

Fecha dd/mm/aa	**Tiempo dedicado ***	**Tipo de tarea ****
10/12/21	1h15m	Diseño del sistema
15/12/21	2h35m	Programación
18/12/21	1h30m	Diseño del sistema
22/12/21	1h15m	Programación
28/12/21	4h	Diseño del sistema
29/12/21	4h	Diseño del sistema
02/01/22	4h	Programación
04/01/22	3h	Programación
06/01/22	3h	Programación
07/01/22	7h	Programación
08/01/22	6h	Programación
08/01/22	1h	Reuniones
09/01/22	7h	Programación
11/01/22	10h	Programación
12/01/22	8h	Programación
13/1/22	8h	Programación
14/1/22	12h	Programación
15/01/22	5h	Programación, Diseño del sistema y pruebas
Tiempo total dedicado		85,35h

Nombre completo: Inés Román Gracia

Fecha dd/mm/aa	**Tiempo dedicado ***	**Tipo de tarea ****
10/12/2021	1h15m	diseño del sistema
11/12/2021	1h15m	programación
19/12/2021	1h	reunión
20/12/2021	30m	reunión
21/12/2021	2h	programación
22/12/2021	2h	programación
03/01/2022	30m	programación
06/01/2022	45m	reunión
07/01/2022	15m	programación
10/01/2022	2h	documentación
Tiempo total dedicado		11h

Nombre completo: Leonor Murphy Anía

Fecha dd/mm/aa	**Tiempo dedicado ***	**Tipo de tarea ****
10/12/21	45m	Reunión
17/12/21	1h	Diseño del sistema
18/12/21	45m	Diseño del sistema
19/12/21	1h 30m	Programación
19/12/21	45m	Programación
20/12/21	30m	Reunión
21/12/21	1h 15m	Programación
03/01/22	1h	Documentación
04/01/22	1h	Programación
06/01/22	1h	Reunión
08/01/22	45m	Reunión
10/01/22	30m	Reunión
10/01/22	45m	Documentación
12/01/22	1h	Diseño del sistema
13/01/22	1h	Documentación
14/01/22	1h	Preparar defensa
Tiempo total dedicado		14h 30m

Nombre completo: Pablo López Mosqueda

Fecha dd/mm/aa	**Tiempo dedicado ***	**Tipo de tarea ****
10/12/21	45m	Reunión
17/12/21	1h	Diseño del sistema
18/12/21	45m	Diseño del sistema
19/12/21	1h 30m	Programación
19/12/21	45m	Programación
20/12/21	30m	Reunión
21/12/21	1h 15m	Programación
03/01/22	1h	Documentación
04/01/22	1h	Programación
06/01/22	1h	Reunión
08/01/22	45m	Reunión

10/01/22	30m	Reunión
10/01/22	45m	Documentación
12/01/22	1h	Diseño del sistema
13/01/22	1h	Documentación
14/01/22	1h	Documentación
Tiempo total dedicado		14h

Nombre completo: Javier Pizarro Martínez

Fecha dd/mm/aa	**Tiempo dedicado ***	**Tipo de tarea ****
10/12/21	1h15m	diseño del sistema
11/12/21	2h	diseño del sistema
11/12/21	1h15m	programación
18/12/21	1h	programación
19/12/21	1h	reunión
20/12/21	1h30m	programación
20/12/21	30m	reunión
21/12/21	2h	programación
22/12/21	2h	programación
22/12/21	30m	programación
06/01/22	45m	reunión
07/01/22	15m	programación
08/01/22	45m	reunión
08/01/22	1h	programación
10/01/22	3h30m	programación
11/01/22	5h30m	programación
13/01/22	4h	programación
14/01/22	3h30m	programación
15/01/22	5h	programación
Tiempo total dedicado		37h15m

6. Conclusiones y posibles mejoras futuras

Se considera que haciendo el proyecto en general se han tomado buenas decisiones ya que no se han tenido problemas de forma excesiva. Uno de los mayores problemas que hemos podido tener ha sido por ejemplo la estructura excesivamente compleja de los analizadores en un primer momento, debido a ello se tuvieron que destinar muchos recursos al mismo para acabar rehaciéndolo en los últimos días antes de la entrega. Otro fallo que se encuentra fue el posponer tanto el despliegue en línea del proyecto, esto causó que se descubrieran problemas que en local no se daban y hubiera que modificar partes de la comunicación.

Para mejorar nuestro proyecto, se podría añadir en un futuro una función para cronometrar el tiempo que tardan los worker en realizar sus tareas. Así se podría medir mejor el impacto de las diferentes variables. También se podría añadir un protocolo para que la memoria utilizada en el envío de mensajes por socket sea dinámica. Por último se podría optimizar el analizador de tags para reducir el tiempo que se emplea en este.