Memoria técnica del proyecto

# Programación de Sistemas Concurrentes y Distribuidos

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

**Número de equipo**: *15*

**Integrantes**:

* *Sául Daniel Soriano, 815743*
* *Ernesto Bielsa,*

*798799*

* *Adrián Fortea Valencia, 815177*
* *Elshad Feyzili*

*848508*

# Índice de contenidos

1.Introducción 3

2.Diseño de alto nivel de la solución 3

4.Evaluación del sistema final 10

5.Planificación y organización del trabajo 11

6.Conclusiones y posibles mejoras futuras 12

# 1.Introducción

El problema por resolver en este trabajo es crear un sistema distribuido que analice colecciones de tweets y extraiga conocimiento de interés utilizando una arquitectura master-worker y un servicio responsable de la gestión de las colas de mensajes.

Para ello se han separado las funcionalidades del sistema en 4 máquinas, una por cada integrante del grupo:

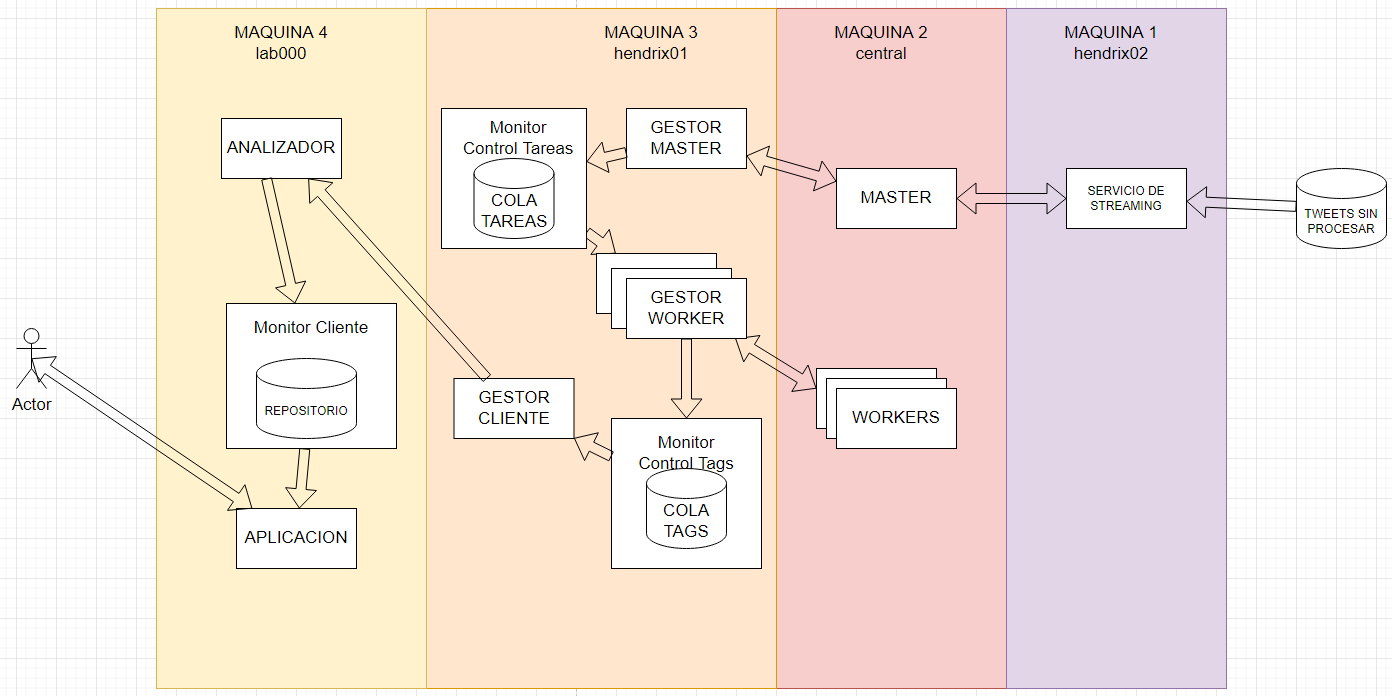
Maquina 1: Saul se encargará del Servicio de Streaming, el cual procesará una colección de tweets en tiempo real y obtendrá por separado la fecha, el cliente, el autor y el contenido de cada tweet.

Maquina 2: Ernesto será el encargado del proceso Master y los procesos Workers responsables de la extracción de los tags de los tweets.

Maquina 3: Adrián será el responsable de la gestión de las colas de Tareas y de Tags mediante procesos que servidores que se conectaran con el Master, los Workers y el Analizador.

Maquina 4: Elshad se ocupará del servicio analizador de tags y la aplicación del cliente para guardar y consultar el historial de tags de los tweets.

# 2.Diseño de alto nivel de la solución



Diseño a alto nivel de los procesos, monitores, bases de datos y comunicaciones de cada máquina.

Cada rectángulo representa un proceso.

La máquina 1 tiene el proceso de servicio de Streaming, el cual recoge tweets de una base de datos y los envía al Master., y el proceso Master, el cual se encarga de enviar los tweets recibidos del Servicio de Streaming para mandarlos de 5 en 5 al Gestor Master, para que este los introduzca en la cola de Tareas.

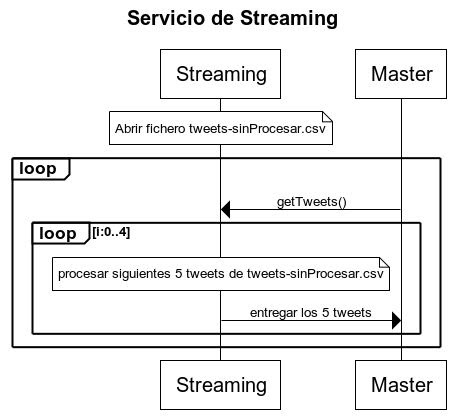
En la maquina 2 se encuentra el proceso Master que se encarga de enviar los tweets recibidos del servicio de Streaming al Gestor de Master para que lo introduzca en la cola de Tareas.

También se encuentran en la maquina 2 los N procesos Workers, que se encargan de recibir tweets de la cola de Tareas mediante los N gestores Workers, de filtrar los tags en los tweets y reenviárselo a los gestores Workers para que los introduzcan en la cola de Tags.

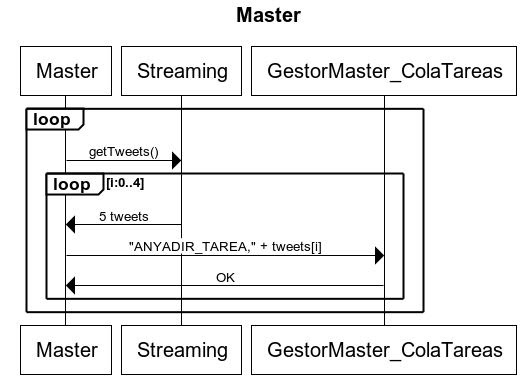
En la maquina 3 además de los 2 gestores mencionades anteriormente también está el gestor Cliente, que se encarga de mandarle al proceso Analizador de la maquina 4 los tags de la cola de Tags.

Finalmente, en la maquina 4 el proceso Aplicación lee los datos que introduce el proceso Analizador en el repositorio, para mostrarlo por pantalla al cliente.

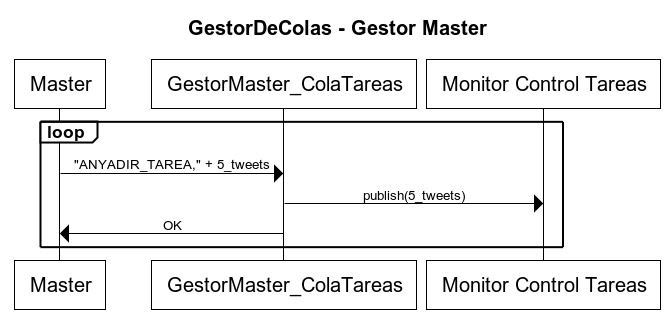
# 3.Decisiones de diseño relevantes



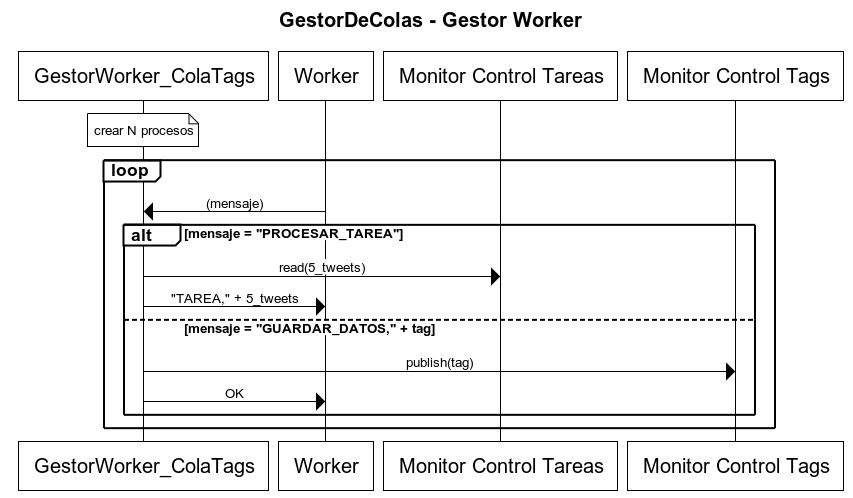
**Diseño del servicio de streaming**. Se ha decidido procesar los tweets en tiempo real según iba siendo requerido debido a que es un comportamiento más real que el de primero procesar todos los tweets y luego atender a las peticiones. También se ha decidido mandar los tweets de 5 en 5 en vez de los 25 de golpe para evitar posibles errores en la red debido a la comunicación de muchos bytes a la vez.



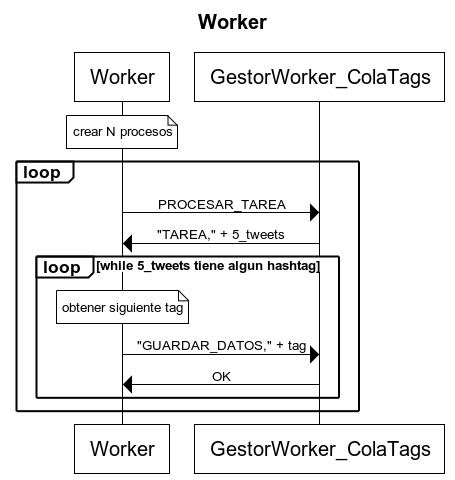
**Diseño del proceso Master**. Debido a que el servicio de Streaming se encarga de mandar los tweets de 5 en 5 no hace falta que el Master los separe, por lo que nada más recibir 5 tweets los reenvía al Gestor Master. Para que el Gestor Master introduzca los tweets en la cola de tareas se debe mandar “ANYADIR\_TAREA,” al principio de cada mensaje. Si se introduce correctamente se recibirá un “OK” a continuación de parte de Gestor Master.



**Diseño del Gestor Master**. Este gestor solo accede a la cola de Tareas. El comportamiento corresponde con el descrito en el Master.



**Diseño del Gestor Worker**. Hay N procesos, siendo N el número de procesos Workers que hay también. Cada proceso del Gestor Worker está esperando mensajes de su Worker asociado, y según el mensaje que reciba hará una cosa u otra. Si recibe “PROCESAR\_TAREA” le enviará los 5 siguientes tweets almacenados en la cola de Tareas. Si por el contrario recibe “GUARDAR\_DATOS” junto a un tag, guardará este tag en la cola de Tags.

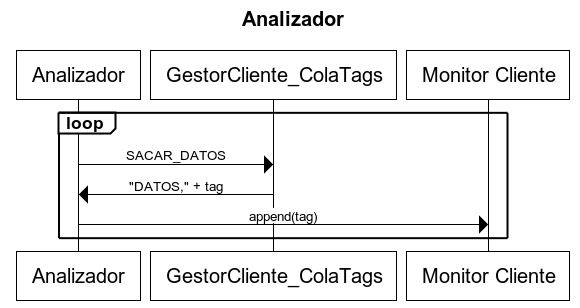


**Diseño del proceso Worker**. Al igual que el Gestor Worker, existen N procesos iguales actuando al mismo tiempo. Cada proceso sigue la siguiente secuencia de instrucciones: primero recibe 5 tweets enviándole el mensaje “PROCESAR\_TAREA” al Gestor Worker. Entonces obtiene cada tag y lo envía por separado y espera al mensaje “OK” de confirmación.

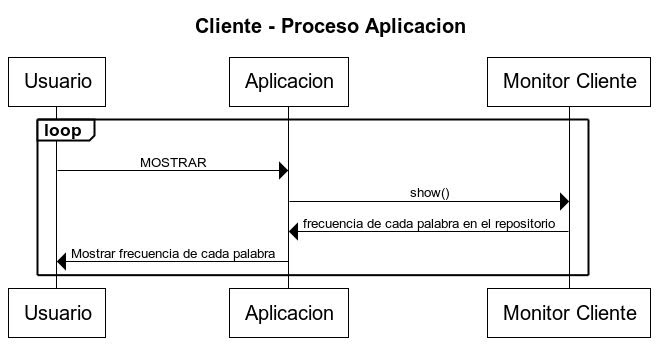
Diagrama, Tabla

Descripción generada automáticamente

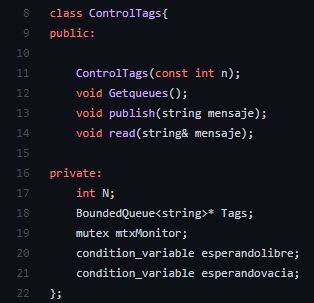
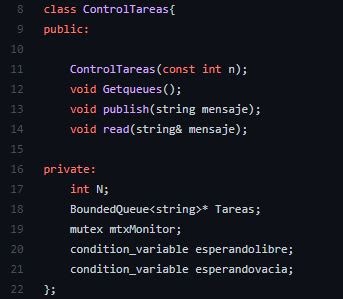
**Diseño del proceso Gestor Cliente**. Está esperando a recibir el mensaje “SACAR\_DATOS” del proceso Analizador para mandarle el siguiente tag de la cola de Tags.



**Diseño del proceso Analizador**. Está continuamente mandándole el mensaje “SACAR\_DATOS” al Gestor Cliente para introducir el tag recibido en la última línea del repositorio a través de la función append () del Monitor Cliente.



**Diseño del proceso Aplicación**. El objeto Usuario es el cliente que ha ejecutado la aplicación. Cada vez que Usuario introduzca el mensaje “MOSTRAR”, la aplicación ejecutará el método show () del Monitor Cliente, el cual consiste en abrir el repositorio y mostrar la frecuencia de cada palabra en su interior.



**Interfaces de los monitores responsables del control de las colas de Tareas y de Tags** respectivamente.

Las variables permanentes de los monitores son:

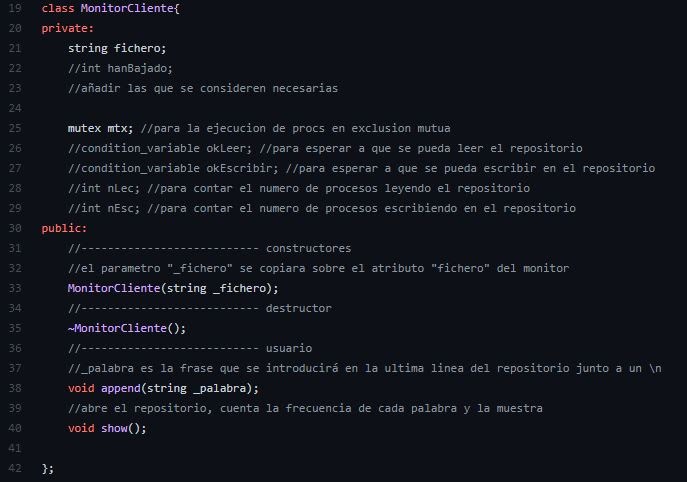
* N que hace referencia al número máximo de elementos que pueden tener las colas que tiene que ser 50.
* BoundedQueue<string>\* Cola es un puntero a cola que se usará para crear la respectiva cola.
* mutex mtxMonitor para hacer las operaciones del monitor

Las variables condición de los monitores son:

* Condición esperandolibre, cuando una operación espera a que haya algún elemento libre para añadir a la cola
* Condición esperandovacia, cuando la cola está vacía y esperamos que haya algún elemento para sacar

Operaciones de los monitores:

* ControlCola (int n) usa la variable de entrada n que usará para inicializar N y la cola usando el puntero a la cola
* Getqueues () muestra el contenido de la cola en caso de ser necesario
* publish (string mensaje) usa la variable de entrada mensaje que será el elemento que se publicará en la respectiva cola
* read (string mensaje) usa la variable de entrada/salida mensaje que se modifica su valor al primer elemento de la cola, borrándose dicho elemento de cola

Interfaz del Monitor Cliente. Este monitor se ha hecho debido a que varios procesos iban a acceder y modificar un fichero en común. En el constructor se indica el fichero a tratar como repositorio. El método append introduce en la última línea el string \_palabra junto a un salto de línea. El método show () abre el repositorio para leerlo desde el principio y contar la frecuencia de cada palabra gracias a una estructura de datos Map (string => int) y a continuación mostrarlo por la pantalla del cliente.

# 4.Evaluación del sistema final

Para desplegar el sistema se han utilizado las maquinas hendrix01 (Solaris), hendrix02 (Solaris), lab000 (Linux) y central (Linux), las cuales tienen 6 núcleos de CPU cada una, los cuales les permite ejecutar 6 procesos de forma concurrente.

Se ha empezado ejecutando el Servicio de Streaming y el Gestor de Colas en hendrix02 y hendrix01 respectivamente, ya que son los procesos servidores.

A continuación, se han ejecutado los procesos clientes: en ‘central’ el proceso Master y los N procesos Workers y en lab000 el cliente Analizador.

El número de procesos totales ha sido 6 + 2\*N, siendo N el número de procesos Workers, que se multiplica por 2 debido a que también habrá N procesos gestores de Workers.

La comunicación entre cada proceso ha sido a nivel de transporte mediante el protocolo TCP/IP.

Para poder ejecutar este sistema es necesario disponer de 4 puertos abiertos, uno para cada Gestor y otro más para el Servicio de Streaming.

# 5.Planificación y organización del trabajo

Para empezar, se ha repartido el trabajo de cada maquina como se ha explicado en la introducción de esta memoria:

Maquina 1: Saul

Maquina 2: Ernesto

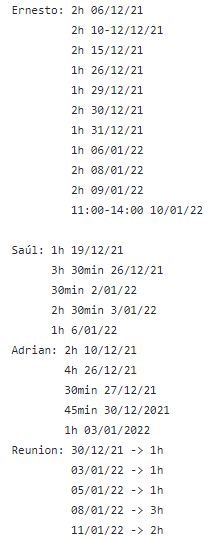
Maquina 3: Adrián

Maquina 4: Elshad

Una vez cada integrante del grupo tiene claro cuál es su parte, se puso una fecha límite para que cada uno hiciese un diseño de su parte y se hizo una reunión para acordar como iba a ser la comunicación entre distintas maquinas.

Entonces se marcó otra fecha límite para subir a un repositorio GitHub común todo el trabajo para que pudiésemos acceder todos a la parte de los demás y poder probarla. En esa reunión se probó todo el funcionamiento en conjunto y se dio por concluida la fase de creación de código.

A la siguiente reunión que fue 2 días después se dedicó a que Saul y Adrián creasen y probasen las pruebas del trabajo mientras que Ernesto se dedicaba a redactar la memoria y hacer todos los diagramas.



# 6.Conclusiones y posibles mejoras futuras

Este trabajo nos ha servido para comprender mejor cómo funciona la arquitectura Master-Worker y como se deben usar los monitores para asegurar un correcto acceso a recursos compartidos.

Lo que más ha costado de hacer ha sido tratar los tweets y los tags debido a que había que manipular mucho texto y era difícil asegurarse de que todos los procesos estaban cumpliendo correctamente con su comportamiento esperado.

Otra cosa que ha costado hacer ha sido ejecutar todo el sistema repartido en 4 máquinas ya que en cada maquina había que cambiar los Makefiles para poder compilar correctamente, e incluso se ha tenido que crear un Socket diferenciado para la compilación en Hendrix y la compilación en Linux. Además, se ha tenido que ejecutar el Master en la misma máquina que el Servicio de Streaming ya que si no se hacía así daba un error que no hemos se sabía solucionar.

A pesar de que el sistema actual funciona correctamente, creemos que podríamos haber mejorado el rendimiento en las comunicaciones si en lugar de enviar los tags de uno en uno se hubiesen mandado más a la vez, o si hubiésemos aprovechado mejor los 6 núcleos máximos de cada máquina para poner 6 workers aislados en la misma maquina junto a 6 procesos gestores de Workers en otra máquina para aprovechar al máximo la arquitectura Master-Worker.

Si se usase una maquina entera solamente para los gestores Workers se obtendría un +50% de rendimiento en el sistema ya que pasaríamos de 4 procesos Gestor Worker máximos que se pueden tener ahora (ya que en la misma maquina hay 2 procesos extra a la vez: el Gestor Master y el Gestor Cliente) a 6 procesos al mismo tiempo.