Memoria técnica del proyecto

Programación de Sistemas Concurrentes y Distribuidos

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

Número de equipo: 11

Integrantes:

Héctor Arcega Vela, 818058

Cecilia Monteagudo Redrado, 820417

Berta Olano Guillén, 815935

Carlos Palomera Oliva, 820949

Alonso Lucas Juberías, 815767

Simón Alonso Gutiérrez, 821038

.

Índice de contenidos

1.Introducción

Nuestro equipo está formado por:

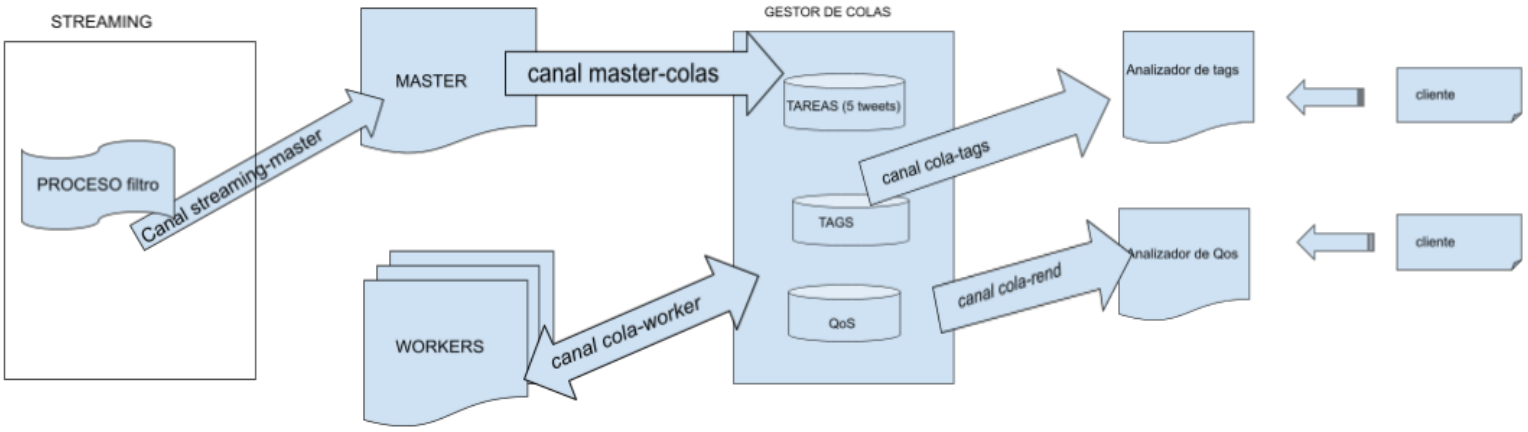
Héctor Arcega Vela, Cecilia Monteagudo Redrado, Berta Olano Guillén,

Carlos Palomera Oliva, Alonso Lucas Juberías, Simón Alonso Gutiérrez

En este sistema el objetivo ha sido que mediante 5 programas (Servicio de Streaming, Master, Workers, Analizadores y Gestor de Colas) ubicados en 4 máquinas diferentes se consiga procesar los tweets suministrados obteniendo de estos diferentes datos. Para obtener dichos datos, en primer lugar el servicio de streaming los procesa enviando al master la información necesaria para que este lo envíe al gestor de colas donde se almacenarán los bloques de tweets recibidos. El gestor de colas se encarga de pasar estos bloques a los workers, quienes le devolverán al gestor los tags y qos ya procesados y listos para almacenar. Una vez almacenados en el gestor, éste los envía a los analizadores para que extraigan la información de interés de ellos. Los analizadores crean estadísticas con los resultados del análisis de esta información y las escriben en dos ficheros de texto.

En cuanto a la estructura de este documento, el apartado de diseño a alto nivel de la solución incluye un esquema del sistema realizado con un pie de página explicativo, así como una presentación del diseño concreto del sistema. Consta además de un apartado de decisiones de diseño relevantes en el que, separando cada parte del proyecto en subsecciones, se describe de manera más detallada cada componente de este sistema. La evaluación final del sistema indica que programas concretos hay en cada máquina del sistema y las máquinas concretas que se utilizan. Posteriormente se describen las decisiones organizativas del proyecto y cómo se ha repartido respecto al tiempo el trabajo entre los integrantes del equipo en el apartado de planificación y organización. Por último, en conclusiones y posibles mejoras futuras hacemos un análisis de nuestro sistema reflexionando sobre lo que podría mejorarse.

2.Diseño de alto nivel de la solución



Representa las 4 máquinas en 4 columnas conectadas por canales. La primera máquina son el proceso filtro y el servicio streaming, que envía los tweets sin los caracteres no permitidos al proceso master mediante el canal streaming-master. Este último proceso ya pertenece a la segunda máquina junto a los workers. El master envía los tweets en el formato requerido a l gestor de colas mediante el canal master-colas. El gestor de colas forma la máquina 3 y está compuesto por las diferentes colas ilustradas en la imágen. Del gestor se envían a los workers los bloques de tweets por el canal cola-worker, de nuevo a la máquina 2. El worker extrae los tags y qos y los envía por el mismo canal de nuevo al gestor de colas, que los almacena en sus respectivas colas, de las que luego los manda a los analizadores, máquina 4, mediante los canales cola-tags y cola-rend respectivamente. Estos sacan la información correspondiente y los clientes la ordenan en un fichero.

La tarea del servicio streaming es acceder a los tweets en bruto, ver si cumplen con el formato correcto y filtrarlos para dejar solo los caracteres válidos antes de enviarlos por un canal al master.

El master se encarga de enviar una petición al servicio de streaming mediante un canal. Después, debe darle el formato adecuado a los strings de tweets que recibe del servidor streaming. De la misma manera envía al gestor de colas mediante otro canal distinto los tweets ya modificados.

Respecto al gestor de colas, este se encarga de comunicar el proceso master con los workers y a su vez estos con los analizadores. La información que recibe la guarda en una cola distinta dependiendo del proceso que le envíe los datos, pero en ningún momento el proceso que envía los datos le dice en qué cola tiene que guardar la información, de esto ya se encarga el propio gestor de colas. Esto es posible debido a que tenemos 3 monitores(uno para cada cola) los cuales ejecutan todas las operaciones en exclusión mutua, con esto aunque una cola se llene las otras dos pueden seguir haciendo sus operaciones con normalidad.

Los workers envían una petición al gestor de colas, del que reciben 5 tweets (por worker). Extrae los tags entendidos como los #, @, fecha y enlaces utilizados en cada tweet mediante la función extraertags explicada posteriormente. Lo mismo se hace con los qos que hemos decidido que sean el tiempo que se tarda en extraer los tags de cada tweet y la cantidad de estos que se cogen de cada uno. Recoge en un string los tags y en otro los qos y los envía por el mismo canal al gestor de colas después de haberle avisado mediante una petición de que se iban a enviar estos strings.

Los analizadores acceden a sus respectivas colas que les comunican por un lado los tags por tweet y por otro el número de tags por tweet y el tiempo que ha costado procesarlos. Esos datos son compartidos con el cliente ayudándose de dos monitores que aseguran la exclusión mutua (uno para los tags y otro para el Qos) y que son necesarios para el correcto acceso a los datos por parte del cliente. Este elabora unas estadísticas con el resultado del análisis de los datos compartidos por los analizadores y se encarga de escribirlas en dos ficheros de texto.

3.Decisiones de diseño relevantes

3.1 Diseño del gestor de colas

3.1.1 Monitor gestor de colas

En este monitor dedicado al gestor de colas, hemos definido una Bounded Queue(una cola con límite de tamaño) y dos variables condición, esperandoPublish y esperandoRead.

Las operaciones realizadas en exclusión mutua necesarias para el buen funcionamiento del monitor, han sido:

publish(string cadena)

Esta operación se encarga de encolar los elementos en la cola controlando que el tamaño no sobrepase al máximo definido y notificando a la operación read de que hay elementos disponibles para leer, en caso contrario, la inclusión de nuevos elementos se bloquea hasta que haya hueco libre. Todo esto en exclusión mutua.

read(string& cadena)

Esta operación se encarga de extraer y desencolar los elementos de la cola. Mientras el tamaño de esta sea igual a 0, se bloqueará esperando a que publish le de permiso para extraer datos, es decir, que haya algún elemento para extraer. Aquí también notificaremos a publish cuando el número de elementos de la cola sea menor al máximo, ya que puede que este esperando para añadir nuevos elementos cuando haya hueco libre.

3.1.2.Gestor

Para organizar la información, hemos creado 3 colas (Tareas, Tags y Qos), es decir, hemos creado 3 monitores para que cuando necesitemos bloquear la entrada o salida de datos de alguna de ellas, no se bloqueen el resto de colas y se pueda acceder a sus datos.

Para las conexiones con los diferentes procesos, existen 4 canales, por uno se reciben los datos del master, por otro se envían y reciben los datos de los workers y por los dos restantes se envían los datos para los analizadores de tags y qos. Por cada una de las conexiones se crea un nuevo proceso y en el caso de los workers, el número de procesos worker será igual al número de workers especificado al ejecutar el programa.

Un problema que nos surgió fue que el gestor de colas tenía que recibir un bloque de tweets en un solo recv, pero como era demasiado largo el mensaje nos daba error, esto lo solucionamos haciendo un recv por cada tweet en vez de por cada bloque.

3.2.Diseño del máster

3.2.1.Conexión con Streaming

Establece conexión mediante el canal chanstreaming con el servicio Streaming. Primero envía una petición a éste y recibe como respuesta 25 strings que corresponden a un tweet cada uno. Guarda cada tweet en una componente de un vector que vacía cada iteración de un bucle infinito.

3.2.2.Conexión con el gestor de colas

Establece conexión mediante el canal chancolas con el gestor de colas y envía los tweets recibidos con el formato adecuado para que el gestor pueda crear la tarea de 5 tweets.

3.3.Diseño de los workers

3.3.1.Función separar cadena

Esta función separa un string tantas veces como separadores haya y los guarda en un vector de strings. Se usa para separar los tweets por palabras y poder buscar así los tags.

3.3.2.Función extraer tags

Una vez tenemos el vector con las palabras del tweet separadas, busca los tags que hemos considerado hashtags(#), menciones(@), enlaces y fecha. Ya encontrados los guarda en una componente del vector de string tags separados por un delimitador. Cada componente corresponde a un tweet del bloque. Además lleva la cuenta del número de tags que se extraen de un tweet, lo que se utiliza luego para los qos.

3.3.3.Worker

Cada worker es un thread diferente que envía una petición al gestor de colas, del que recibe un bloque de 5 tweets, cada uno en un string. Extrae sus tags mediante la función anteriormente explicada y calcula el tiempo que tarda en realizar esta tarea para cada uno de los tweets. Este tiempo (en microsegundos) más la cuenta de tags los guarda en un vector de strings qos. Posteriormente envía una petición para poder mandar los strings que contienen los tags y los qos por separado. El número de procesos worker a utilizar se pasa como argumento en la ejecución y así se puede modificar fácilmente.

3.4 Diseño del streaming

3.4.1. Conexión con Streaming

Establece conexión con el master por medio de una canal denominado chanStreaming. Una vez establecida la conexión se mantendrá a la espera de un string, si este string no es el que llegá mostrará por pantalla un mensaje de error y se pondrá a la espera de un nuevo string. Si el string es el correcto le mandará 25 tweets ya filtrados como respuesta.

3.4.2. Tweets en bruto

El servicio streaming abrirá el fichero "tweets-sinProcesar.csv" para acceder a los tweets. Lee caracter por caracter todo su contenido. Si el servicio llega al final del fichero, lo cerrará para volverlo a abrir, y comenzar así de nuevo el ciclo.

3.4.3. Filtro

El servicio lee los tres primeros apartados del tweet que tendrán que tener el formato válido, si no será descartado y se pasará al siguiente. Cada vez que lee un carácter comprueba si es un carácter permitido, si lo es se añade a un string que conformará un tweet cuando alcance un fin de línea. Tras ello será comunicado con el master como se ha ya descrito.

3.5 Diseño de los analizadores

3.5.1. Conexión con gestor de colas

Establece conexión con el gestor de colas mediante dos canales, uno para cada cola. A través de ellos envía una petición trás la cual recibirá los tags extraídos por un canal y el tiempo que ha costado extraerlos por otro.

3.5.2. Gestión de los mensajes

Ya que lo que le llega por los canales no tienen el formato indicado para trabajar, lo cambiamos a un formato más cómodo para luego realizar el análisis y almacenarlo en un vector dinámico (para los tags), y en dos enteros (para los QoS), que compartirá con el cliente.

3.5.3. Monitores

Para garantizar la exclusión mutua de los datos compartidos entre las funciones de gestión de los mensajes y el cliente hemos usado dos monitores. Uno de ellos se encarga de los tags y otro de los QoS. Son dos monitores sencillos con dos instrucciones, una para entrar y otra para salir.

3.5.4. Cliente

El proceso cliente se encarga de acceder a los datos compartidos anteriormente citados, realiza el análisis de los datos y lo escribe en dos ficheros "AnalisisTAGs.txt" y "AnalisisQOS.txt" que tendrán que haber sido previamente creados.

4. Evaluación del sistema final

En este sistema hemos utilizado un total de cuatro máquinas distribuidas de la siguiente manera:

La máquina 1, en la que se ejecuta el servicio de streaming, la lanzamos en la máquina *hendrix02.cps.unizar.es* con sistema operativo SunOS.

La máquina 2 que contiene los procesos master y workers, se lanza en *central.cps.unizar.es* con sistema operativo Linux CentOS.

La máquina 3 engloba al gestor de colas compuesto por los procesos: servmaster, servworker, servqos,servtags. Se ejecuta en *lab000.cps.unizar.es* con sistema operativo Linux CentOS.

Por último la máquina 4 que contiene el proceso analizadores y cliente, se ejecuta en *hendrix01.cps.unizar.es* con sistema operativo SunOS.

En cuanto a las condiciones de ejecución, es necesario que se ejecuten primero los servidores Streaming y Gestor de colas para que después master y workers puedan conectarse adecuadamente, ya que hacen la función de “clientes”. En último lugar, una vez que los workers ya han sido ejecutados, se ejecutan los analizadores.

Algunas decisiones más tomadas por el equipo son las siguientes:

En la máquina 3 hay además 3 monitores para el gestor de colas, todos con las mismas operaciones ya descritas anteriormente (mismo fichero para los tres).

Únicamente hay 1 canal bidireccional para la conexión con los workers, envío de tweets y recibo de tags y qos. Entre la máquina 2 y la 3.

Finalmente decidimos envíar los tweets uno a uno pero almacenándolos por bloques para una mejor conexión con diferentes máquinas.

5. Planificación y organización del trabajo

En nuestra primera reunión decidimos repartir el trabajo de la siguiente manera:

-Gestor de colas: Simón y Héctor

-Master y workers: Berta y Cecilia

-Streaming y analizadores: Carlos y Alonso

Una vez leído todo el trabajo y repartidos los roles, nos centramos más cada uno en desarrollar un esquema de nuestras respectivas partes para una mayor comprensión del trabajo. Después de esos días de comprensión de cada parte, organizamos una reunión para crear el diseño general con las diferentes conexiones que habría entre máquinas y procesos.

Cuando ya sabíamos cual era el trabajo de cada sección, comenzamos a realizar las diferentes partes divididos en las parejas mencionadas anteriormente. Durante este proceso la mayor parte de las ideas estaban claras y las conexiones entre las diferentes partes eran buenas pero cuando nos juntamos de nuevo para comprobar todas las conexiones, surgieron algunos problemas de conexión ya que había aspectos que implementamos de diferentes maneras y tuvimos que llegar a una sola idea para todos los ficheros se conectasen como habíamos pensado.

Una vez realizadas todas las conexiones, comprobamos el paso de datos y hubo errores de mensajes no recibidos o enviados incorrectamente, después de modificar algunos aspectos como bucles, envíos de datos que no correspondían y establecer ciertas normas de conexión (como por ejemplo las peticiones de enviar o recibir datos por parte de los workers), conseguimos que todos los datos se transmitiesen correctamente en localhost.

Ya comprobados todos los aspectos de conexiones y envío de mensajes en localhost, pasamos a verificar que todo funcionaba correctamente en varias máquinas. En este punto, surgieron errores de envío de bloques de tweets por ser demasiado extensos y decidimos enviarlos individualmente y almacenarlos por bloques en las colas, para luego ser enviados de nuevo individualmente a los workers. A partir de este punto no nos llevó mucho tiempo ya que solo surgieron un par de errores que no supusieron un gran problema. Viendo que todo se enviaba y recibía correctamente y obteniendo los resultados esperados, dimos por finalizada la implementación del trabajo.

El reparto de tareas ha sido equitativo, tal y como se ha mostrado anteriormente. La cantidad de esfuerzo realizada dentro de las diferentes parejas también ha sido equitativa ya que el trabajo se ha realizado en todo momento con las decisiones de ambos miembros y trabajando al mismo tiempo el mismo número de horas, estas decisiones fueron puestas en común al reunirnos el grupo entero. La media de horas de trabajo por parejas son unas 20 horas y sumadas a las de trabajo grupal un total de unas 30 horas. (Para ver esta información de reparto de horas en más detalle, consultar las tablas de control de esfuerzos).

6.Conclusiones y posibles mejoras futuras.

En conjunto el proyecto funciona correctamente, las conexiones son eficaces y los procesos ejecutan su parte del trabajo adecuadamente coordinados y con una velocidad considerable.

En lo referente al Servicio de Streaming pensamos que tras informarnos sobre la codificación de archivos como el entregado de tweets sin procesar para poder usar también algunos caracteres especiales como la “ñ” o las vocales tildadas con la implementación que hemos realizado ha sido un acierto. Además cada pareja ha realizado su parte del trabajo de forma correcta y los problemas que hemos tenido han sido mayoritariamente de conexión. Consideramos que inicialmente quizás no nos comunicamos de la mejor manera y por ejemplo empezamos a desarrollar el master y el worker como servidores teniendo luego que cambiarlos. Por otro lado nos costó un poco al principio comenzar a trabajar, lo que podríamos considerar una mala decisión, aunque luego lo hayamos mejorado.

En cuanto a las mejoras que creemos que podrían llevarse a cabo en nuestro sistema, a la hora de extraer tags de cada tweet se podría obtener más información, como por ejemplo el número de usuarios diferentes que utilizan ciertos hashtags, lo que llevaría a un análisis más concreto en los analizadores y por tanto, a conseguir más información.

De la misma manera el control de calidad podría ser más amplio, teniendo en cuenta por ejemplo los recursos de memoria utilizados al procesar los tweets.

Respecto a la eficacia de los analizadores se podría haber usado un TAD creado por nosotros para aprovechar de mejor manera la memoria y el tiempo. También podríamos cambiar el método por el cual el cliente actualiza los datos analizados para que, en vez de depender del tiempo, dependiera de la cantidad de tweets/tags/qos procesados, y así cada actualización sería una estadística que variara menos (en cuanto a lo de variar, se hace referencia al tamaño de la muestra de cada actualización en cual varía según la velocidad del ordenador, por ende tiene diferentes ejecuciones en diferentes máquinas).

Para mejorar la velocidad de los procesos podríamos haber realizado las comunicaciones de los tweets en vez de uno por uno, en paquetes pequeños. Así no haría falta enviar tantos mensajes entre ellos, lo cual se traduce en un aumento de la velocidad general de todo el sistema. Pero comprobamos que si los paquetes eran muy grandes aparecían errores y la comunicación no era eficaz. Como consideramos más importante la eficacia de la comunicación, preferimos asegurarla a costa de ralentizar los procesos.