Proyecto 3 - Servidor distribuido centralizado

Emanuel Esquivel López, Fabricio Elizondo Fernández, Roger Valderrama
ema11412@estudiantec.cr, faelizondo@estudiantec.cr roger.andres18@estudiantec.cr
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

I. Introducción

En la actualidad el procesamiento distribuido es una practica muy común utilizada para el diseño e implementación de algoritmos complejos, con la facilidad de que estos serán procesados de manera separada y en paralelo en distintos computadores. Un cluster en un sistema de este tipo esta formado por diversas computadores trabajando en conjunto, normalmente este tipo de comunicación se hace por medio de red, ya que estas están comunicadas en un mismo NAT llamado comúnmente LAN.[2]

Un método muy conocido para la comunicación entre computadoras es el uso de REST API, este consiste en servicio de comunicación en la red mediante http, lo que nos facilita el envió de datos mediante el uso de JSON, por lo que se facilita la comunicación entre varias computadores utilizando endpoint los cuales son rutas de acceso predeterminadas para cada REST API el cual esta ubicado en cada computador.[1]

Con lo visto anteriormente en este proyecto se trata el procesamiento de múltiples imágenes mediante el uso de un cluster de 2 nodos, el cual están conectados a un servidor centralizado, lo que nos genera un procesamiento distribuido de imágenes, el cual consiste en invertir o realizar la operación de XOR a las imágenes de entrada esto en los diferentes nodos y a su vez se guardaran en los mismos. A continuación en este documento, se presenta una explicación de las herramientas utilizadas en la elaboración de este proyecto, los atributos ingenieriles involucrados, diagramas que explican el diseño del sistema, instrucciones de uso para los usuarios, bitácora de los desarrolladores, y un análisis final con sus respectivas conclusiones y recomendaciones.

II. Ambiente de desarrollo

El proyecto fue desarrollado en linux, por lo que mucha de su implementación y elaboración fue con el uso de comandos en la terminal del mismo. El servidor del sistema fue elaborado en el lenguaje de programación C, utilizando CLion como ambiente de desarrollo integrado y Cmake como plataforma de generación o automatización del código, para efectuar las relaciones de dependencias en los archivos fuentes, durante el proceso respectivo de compilación. Este servidor tiene que establecer una comunicación con una aplicación web cliente, para lo cual se utilizó el framework HTTP llamado Ulfius, que nos permite desarrollar aplicaciones REST en el lenguaje de C. Para la creación de la página web, con la que tendrá interacción el usuario o máquina cliente, se utilizó el framework de Angular. Este framework nos permite el desarrollo de la aplicación web por medio de módulos, que integran en su interior componentes HTML, CSS y Typescript, simplificando de gran manera la creación de sistemas web complejos. En adición, se incluyen componentes para el manejo de peticiones HTTP, necesarias para establecer la comunicación con la capa del servidor centralizado. Todo lo anterior se llevó acabo en una computadora de arquitectura 64 bits, sistema operativo Linux Manjaro linux KDE, 16 Gb RAM, procesador Ryzen 7 2700x. Además de esto se utilizo una segunda computadora para virtualizar ubuntu y así poder probar los nodos clientes del servidor centralizado.

III. Atributos

III-A. Aprendizaje continuo

En el diseño e implementación del proyecto se aprendió a crear un sistema distribuido mediante el uso de REST API, mediante el framework $ulfius$, el cual es una biblioteca para el manejo de archivos y de peticiones GET, POST, SET, entre otras lo que facilito la comunicación entre los nodos. Se fortaleció el atributo de pensamiento critico para la elección de la arquitectura a implementar así como el trabajo en equipo. Y con el objetivo de alcanzar la funcionalidad completa del sistema, en el tiempo establecido de acuerdo al cronograma, fue necesario aplicar técnicas de manejo de tiempo eficiente entre los desarrolladores, realizando una correcta distribución de tareas, tanto para investigación como para implementación, sumado a una rápida comprensión del problema en cuestión, de manera que se adoptaron prácticas de metodologías ágiles, que permitieran ir comprendiendo y creando módulos del sistema de manera simultánea y durante la marcha.

III-B. Herramientas de ingeniería

Tal y como se contempló en la sección previa este documento,fue necesario la investigación e incorporación de múltiples herramientas, tanto para el desarrollo de interfaces gráficas en la web, como para la implementación de un servidor REST que distribuya las imágenes a ser procesadas. Se utilizaron herramientas de ingeniería nuevas para nosotros ya que se trabajó el servicio REST mediante $ulfius$ lo que facilitó la comunicación entre varios computadoras, con mensajes en formato JSON, además de que el cliente se realizó con el framework de Angular para mayor facilidad y mejor presentación. Finalmente con ayuda de dos computadoras se logro la paralelización del código a ejecutar, que correspondía con aplicar el filtro de XOR a una imagen determinada. Debido a la variedad de tecnologías (frameworks o bibliotecas), lenguajes

de programación y capas de desarrollo que se integraron en el proyecto, se tuvieron que aplicar técnicas adecuadas de diseño para establecer una correcta comunicación entre cada componente, lo cual acrecentó las habilidades en el uso de cada tecnología.

IV. Detalles de diseño

IV-A. *Arquitectura general*

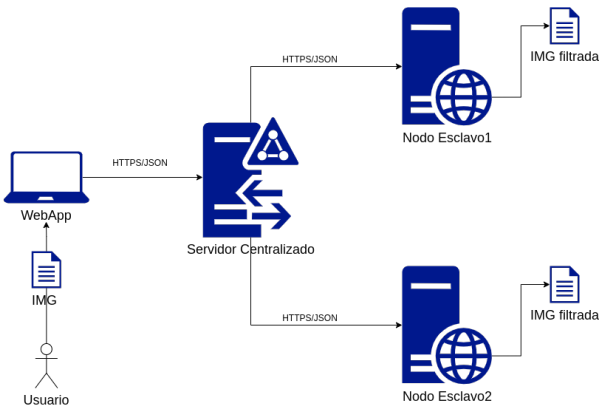

Figura 1. Diagrama de arquitectura

En la figura anterior se ven la interacción de todas las partes principales del proyecto, como lo son el servidor centralizado el cual recibe un JSON por parte de una aplicación web la cual es la que el cliente interactúa.

El cliente en la aplicación se encarga de tomar la imagen de entrada del usuario, junto con la llave para el algoritmo y las veces que el usuario quiere enviar la imagen.

Una vez estos datos son recolectados, son enviados en un JSON para su procesamiento, el servidor principal se encarga de repartir las imágenes en los nodos esclavos, así como desde estos mismos se verán los resultados.

IV-B. *Clases del sistema*

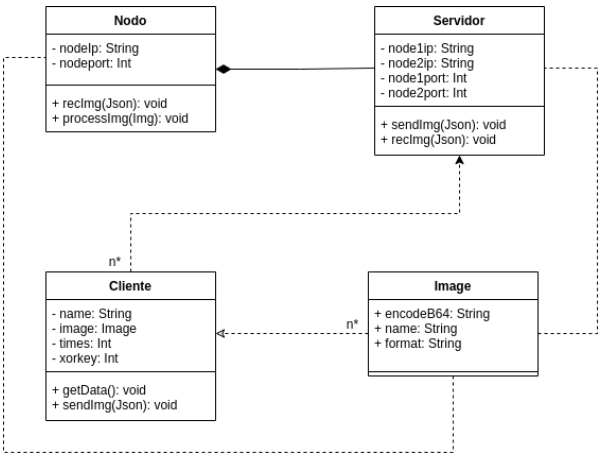

Figura 2. Diagrama de clases

EN este diagrama de clases se puede ver la interacción de las clases generales del sistema, como se observa tenemos4 clases predominantes, como lo son la imagen, la cual solo posee atributos los cuales son su enconde en base64 nombre y formato, así como también podemos ver el sistema de envió o cliente, el cual recopila la información solicitada, lo que es la imagen, xorkey el cual se usara para filtrar la imagen, además de esto también la cantidad de veces que este envía la imagen.

EL servidor tiene como método la recepción de la imagen y envió, el cual son JSON los cuales contienen la información primordial de la imagen, además de esto tiene como atributos la información de los nodos esclavos. Estos mismos *heredan* del servidor ya que poseen el método y atributos similares, así como tambien se le agrega un método nuevo de procesar la imagen de llegada la cual sera procesada y guardada en el mismo nodo.

V. Instrucciones de uso

V-A. *Instalación*

Para la instalación es necesario la disponibilidad de los siguientes paquetes y IDE.

- Clion
- Ulfius

Para esto debe seguir las instrucciones necesarias en cada caso, ya que se trabajo en la distribucion de linux Manjaro, la cual esta basada en Arch, lo que las instrucciones de instalación pueden variar bastante en otras distribuciones.

V-B. *Compilación*

SE debe contar con CMake ya que CLion indexa el CMakeLists.txt para su correcta compilación, en realidad la compilación se hace de manera automática desde CLion pero si es necesario tener CMake en el sistema.

V-C. *Ejecución*

Para el cliente una vez iniciada la pagina web tenemos el siguiente contenido.

Figura 3. Contenido del cliente

EN el cual se puede ver los recuadros principales del cliente, el cual nos pide una imagen, numero de veces de envió y además clave para el XOR.

Además de esto se necesita recolectar la información de envió, en la tuerca se setea los valores del puerto y dirección IP del servidor.

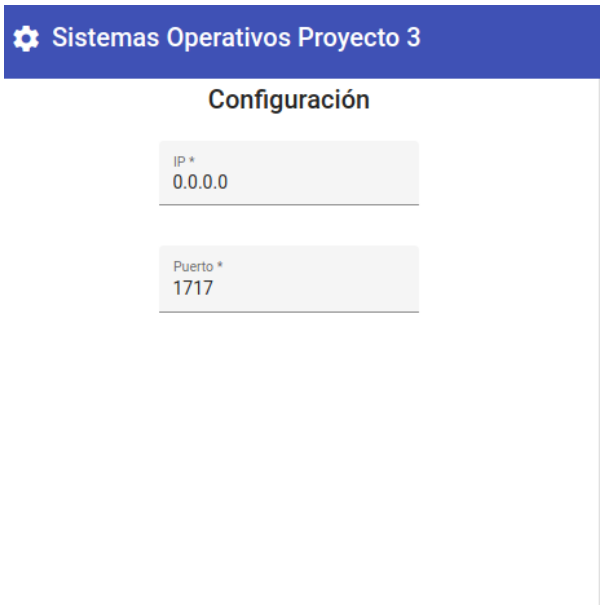

Figura 4. Ajuste de Ip y Puerto

Con esto listo se dispone a enviar al servidor central el cual envía a los nodos esclavos, en la siguiente figura se puede ver los valores deseados a enviar.

Figura 5. Envió de imagen a servidor

Una vez con esto en los nodos se pueden visualizar las imágenes y la consola del mismo imprime el proceso de la siguiente manera.

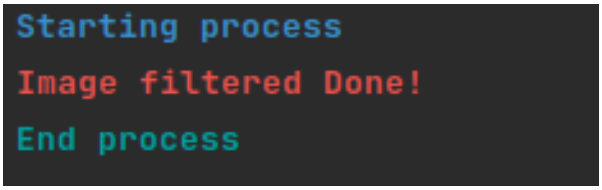

Figura 6. Proceso de imágenes

VI. Tabla de actividades

VI-A. *Emanuel Esquivel*

Actividad	Horas
Investigación sobre OpenMPI	3
Investigación ulfius	5
Investigación sobre PPM format	3
Implementación del filtro para la imagen	3
Trabajo en servidor centralizado	2
Implementación del cliente Web	4
Documentación	2
Total	22

VI-B. *Roger Valderrama Ordonez*

Actividad	Horas
Investigación sobre OpenMPI	3
Investigación ulfius	5
Trabajo en servidor centralizado	6
Implementación del cliente Web	5
Documentación	4
Total	22

VI-C. *Fabricio Elizondo Fernández*

Actividad	Horas
Investigación sobre OpenMPI	2
Investigación ULFIUS Framework	4
Investigación sobre Threads	3
Implementación sobre semáforos	3
Trabajo en servidor centralizado	4
Trabajo del cliente Web	5
Documentación	2
Total	23

VII. Conclusiones

Tal y como se contempló en el desarrollo de este proyecto, el procesamiento distribuido permite una mejor utilización de los equipos disponibles, y mejora el balanceo de procesamiento que se debe realizar dentro de una misma aplicación. Existen tareas u aplicaciones, cuyos requerimientos computacionales, pueden llegar a sobre pasar las capacidades de ejecución de un único equipo. Ante esta situación, podría parecer viable soluciones que busquen un incremento de las capacidades del hardware, sin embargo, no se trata solo de buscar más equipo que nos proporcione una ayuda en el procesamiento respectivo, sino que se deben contemplar los aspectos claves de la aplicación y del contexto en el que nos encontremos, para utilizar de manera inteligente y eficiente los equipos y recursos con los que se pueda disponer en el momento necesario. Esto es precisamente, lo que se busca con distribuir de manera adecuada las tareas por realizar en

un proyecto determinado.

Por otra parte, se puede destacar el hecho de que la distribución de procesos y de recursos, le brinda al usuario final, mejores tiempos de respuesta en las solicitudes que necesiten hacer al sistema, ya que se dividen las tareas, las cuales tendrán equipos computacionales destinados solo para eso, y de esta manera, se consigue una paralelización real de programas en ejecución, por lo que múltiples tareas se concretan de manera concurrente, sin necesidad de esperar la finalización de otras. Situación que se hubiera presentado en este proyecto, sino se distribuyen nodos encargados de filtrar y procesar múltiples imágenes al mismo tiempo.

VIII. Sugerencias y recomendaciones

Al trabajar con el protocolo de REST, para establecer la comunicación entre el cliente, el servidor centralizado, y los nodos procesadores, se optó por utilizar como cliente, una página web, que incluya dicho servicio de comunicación. Al trabajar con páginas web se recomienda utilizar algún tipo de framework, que facilite las labores de desarrollo, y que permita agilizar las pruebas con la parte del usuario. Evidentemente, el enfoque de este proyecto no es el diseño web ni su funcionamiento como tal, no obstante, corresponde con un código necesario para verificar las pruebas y los resultados de que el sistema general funciona como es debido.

Por otra parte, la forma en que se establezca la comunicación entre los distintos equipos, no es tan relevante siempre y cuando se cumpla con el propósito deseado, en nuestro caso optamos por utilizar REST como servicio de comunicación, pero también se evaluaron otras opciones como el uso de la biblioteca de MPI. Lo trascendental en este rubro, es que independientemente del protocolo seleccionado, los procesos se distribuyan de una manera eficiente y se puedan ejecutar tareas de manera paralela, sin necesidad de esperar por algún tipo de respuesta. Para ello se recomienda la implementación de semáforos que eviten el llamado ”busy waiting”de los nodos que esperan por una solicitud de procesamiento, y también se necesita utilizar threads, para ejecutar solicitudes de manera concurrente.

Referencias

[1] Masse, M. (2011). REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. .º’Reilly Media, Inc.”.
[2] Buyya, R. (1999). High performance cluster computing. New Jersey: F’rentice.