UNIVERSIDAD DIEGO PORTALES

Facultad de Ingeniería y Ciencias

Tarea 1: Sistemas Distribuidos
Sistema de Caché

Integrantes: Ignacio Ibarra
Benjamin Muñoz

Sección: 01

Fecha de entrega: 15/04/2023

udp UNIVERSIDAD DIEGO PORTALES

Facultad de Ingeniería y Ciencias

Índice.

1.- Introducción
2.- Descripción del script utilizado
3.- Implementación del caché
3.- Pruebas de rendimiento
4.- Evaluación de escalabilidad
5.- Conclusión
6.- Bibliografía

Introducción

En esta Tarea se construirá un sistema de caché utilizando Redis basada en una política de LRU junto con un backend para poder hacer pruebas y observar su rendimiento, escalabilidad y como es afectado el sistema por diferentes restricciones de hardware.

Descripción del script utilizado

```
const cliente1 = redis.createClient({
  host: 'caching1',
  port: 6379
});

const cliente2 = redis.createClient({
  host: 'caching2',
  port: 6379
});

const cliente3 = redis.createClient({
  host: 'caching3',
  port: 6379
});
```

Imagen 1 = Creación de los 3 espacios de caché

Primero se inicializan los 3 espacios de caché de redis, nombrados “caching1, caching2, caching3”.

```
const rand = Math.floor(Math.random() * (387 - 2 + 1) + 2);
```

Imagen 2 = Generación del número random.

Después se genera el índice de manera random entre 1 y 389, para poder realizar la consulta.

```
if (0 < rand && rand <= 130) {
  cliente1.get(`monsters-${rand}`, async (err, reply) => {
    if (reply) {
      return res.json(JSON.parse(reply));
    } else {
      const response = await axios.get(`https://botw-compendium.herokuapp.com/api/v2/entry/${rand}`);
      cliente1.set(`monsters-${rand}`, JSON.stringify(response.data.data), (err, reply) => {
        if (err) {
          console.log(err);
        } else {
          console.log(reply);
          res.json(response.data.data);
        }
      });
    }
  });
```

Imagen 3 = Funcionamiento del caché

Por consiguiente se verifica si el número random pertenece a uno de los tres cachés.

```
(0 < rand && rand <= 130)

(130 < rand && rand <= 260)

(260 < rand && rand < 390)
```

Imagen 4 = Rangos para el número random

Si se encuentra en el interior del rango, procede a preguntar si este id se encuentra dentro del caché. Si es así, responderá inmediatamente, si no, buscará la información dentro de la API, reemplazando un valor antiguo, por el nuevo id ingresado y así terminando la consulta. Cabe mencionar que este proceso se realiza 3 veces, es decir por cada rango del número random existen códigos iguales.

Cabe mencionar lo siguiente:

```
cliente1.get(`monsters-${rand}`, async (err, reply) => {
```

Imagen 5 = Consulta GET al cliente para la obtención de la información.

```
if(reply) {
  return res.json(JSON.parse(reply));
}
```

Imagen 6 = Búsqueda resulta positiva dentro del caché y retorna la información del “reply”.

El caché pregunta si tiene el índice por el cual se está consultando.

Si lo tiene lo responde inmediatamente, si no lo tiene:

```
const response = await axios.get(`https://botw-compendium.herokuapp.com/api/v2/entry/${rand}`);
cliente1.set(`monsters-${rand}`, JSON.stringify(response.data.data), (err, reply) => {
```

Imagen 7 = Búsqueda resulta negativa dentro del caché y se busca la consulta directamente a la API.

Lo busca directamente en la api y lo ingresa al caché.

Implementación del caché

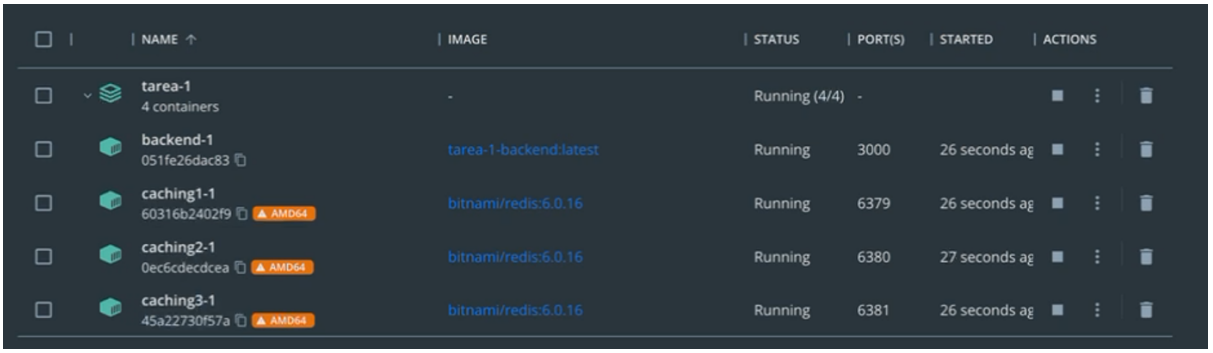

Imagen 8 = Contenedores de docker funcionando.

Como se puede observar en la imagen, se implementaron los tres Redis con los nombres de caching1, caching2 y caching3 junto con un contenedor de docker con el backend, ya que no era necesaria la implementación de un front o una interfaz gráfica.
Cada Caché está seteado con las siguientes características:

```
3   caching1:
4     image: bitnami/redis:6.0.16
5     restart: always
6     environment:
7       - ALLOW_EMPTY_PASSWORD=yes
8       - REDIS_MAXMEMORY=500b
9       - REDIS_MAXMEMORY_POLICY=allkeys-lru
10    volumes:
11      - ./data/redis:/bitnami/redis/data
12    ports:
13      - "6379:6379"
```

Imagen 9 = Características de los clientes redis implementados en la tarea

Pruebas de rendimiento

La primera prueba que se realizó fue la de nuestro sistema con ningún dato ingresado, lo que resultó en el siguiente gráfico:

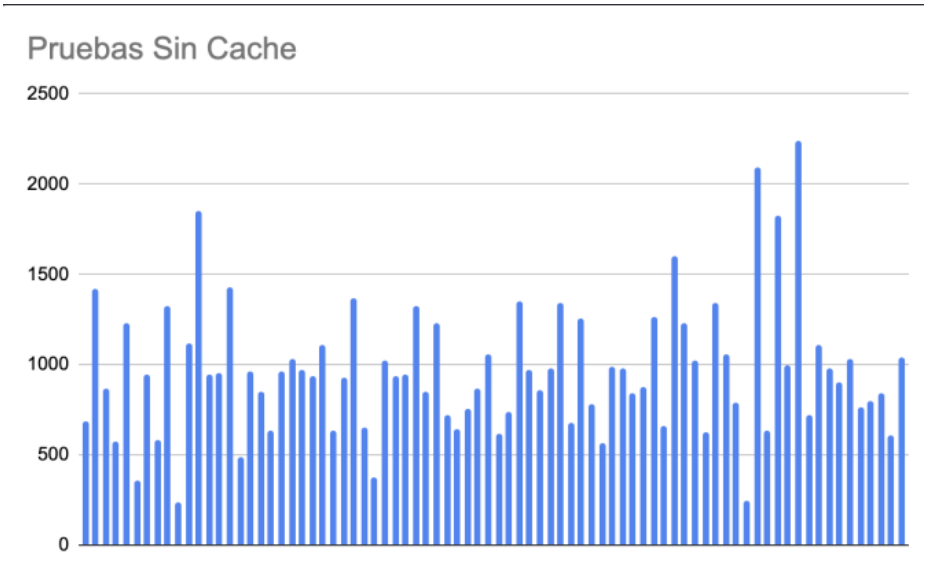

Imagen 10 = Gráfico resultante de la búsqueda sin caché.

Como se puede observar los valores tuvieron distintos tiempos de ejecución, lo que nos lleva a calcular su promedio, el cual se basa en 80, lo que resulta en un promedio de 963.025 (ms).

Después se realizó una segunda prueba donde se comprobó la ejecución de cerca de 50 consulta, pero ahora con el caché del sistema lleno:

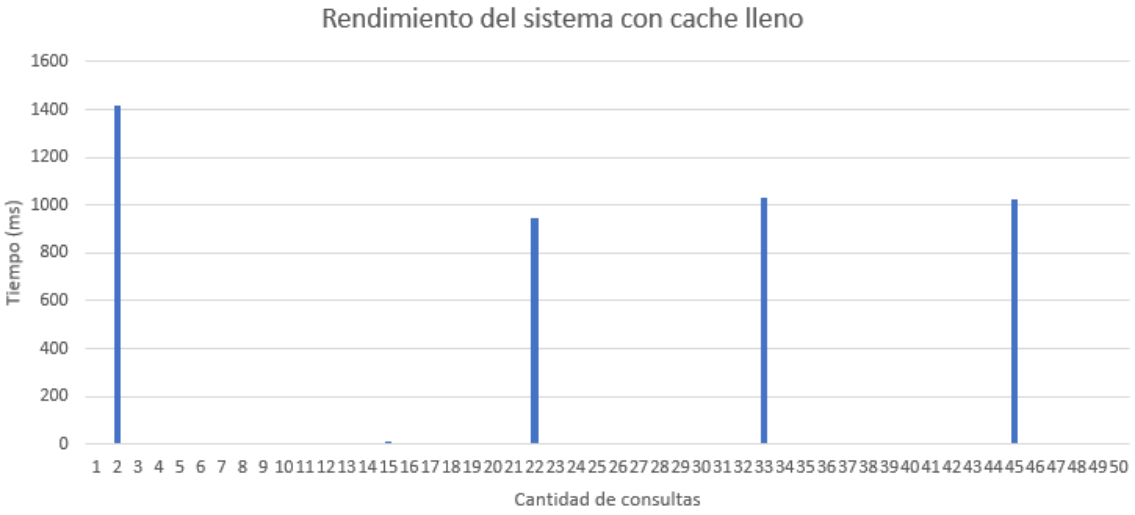

Imagen 11 = Gráfico resultante de la búsqueda con caché.

Como se observa, se pudieron obtener una baja cantidad de ejecuciones con un alto tiempo de respuesta que significan las consultas que se realizaron directamente a la API. De esto se puede concluir el tiempo promedio de consulta, sin tomar en cuenta aquellas que interactuaron con la API, nos da un tiempo de: 2.5(ms).

Evaluación de escalabilidad

Para poder hacer una correcta evaluación de escalabilidad se buscó definir los siguientes puntos, donde cada uno nos permite definir los parámetros que se ocuparon para el desarrollo de la tarea.

```
3   caching1:
4     image: bitnami/redis:6.0.16
5     restart: always
6     environment:
7       - ALLOW_EMPTY_PASSWORD=yes
8       - REDIS_MAXMEMORY=500b
9       - REDIS_MAXMEMORY_POLICY=allkeys-lru
10    volumes:
11      - ./data/redis:/bitnami/redis/data
12    ports:
13      - "6379:6379"
```

Imagen 12 = Características de los sistemas de caché redis.

(a) TTL.
La implementación del TTL en el caché se basa en que cada consulta que se encuentra al interior del caché, tiene un tiempo definido el cual, al cumplirse, produce la eliminación de la consulta liberando espacio para su uso dentro del caché. En nuestro código no se implementó esta cuadriláteros por la baja cantidad de datos que se manejan, haciendo inviable agregar esta característica.

(b) Técnica de particionamiento.
Ya que para el desarrollo de la tarea se nos pide explicitamente usar tres redis, se buscó utilizar cada uno de manera equitativa, de forma de repartir las consultas a los clientes.
De esta forma, para la solución se identificó el número mayor dentro de los identificadores y con este dividirlos en tres partes iguales.

(c) Política de remoción. (LRU)
Para esta sección, se utilizó la política de remoción de LRU, porque al ser una cantidad de índices muy baja, fácilmente al producir consultas es difícil que genere consultas que no se encuentran dentro del sistema de caché, así aligerando la carga que se le genera al sistema.

(d) Tamaño del cache. (500b)
Se eligieron 500b de información para cada uno de los redis dentro de nuestros sistema, porque la cantidad de información con la que se disponía era muy poca y para que pudiera ser válido el uso de particiones.

-Rangos de id para cada caché.

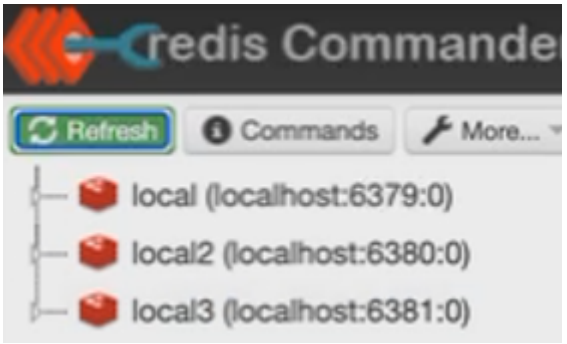

Imagen 13 y 14 = Redis Commander

Se ocupó, una extension de redis para poder visualizar como esta la informacion guardada en cada uno, con lo que se pudo identificar los rangos que tiene los diferentes sistemas de cache para guardar información, los cuales son:

-Cache redis 1(local) :1-129
-Cache redis 2(local 2):130-260
-Cache redis 3(local 3):261-389

Lo que da en promedio un almacenamiento de 128 entradas en cada caché.

Conclusión

En este trabajo se comprobó la importancia e implicancia que tienen los sistemas de caché dentro del funcionamiento de las distintas aplicaciones. Así como la diferencia entre las políticas de remoción y también cómo afecta al rendimiento de un servicio, en relación a la tardanza o consumo de memoria que puede ocurrir por un mal funcionamiento de este sistema.

Bibliografía

https://docs.docker.com/compose/
https://gadhagod.github.io/Hyrule-Compendium-API/#/
https://hub.docker.com/r/bitnami/redis/
https://github.com/gadhagod/Hyrule-Compendium-API
https://botw-compendium.herokuapp.com/api/v2

GitHub de la Tarea

https://github.com/Ignacio-ibarra05/Tarea-1_SD

Video demostración

https://drive.google.com/drive/folders/1RgjgsbxzTAGegSBl-1HH44N9Xz1G_jFN?usp=sharing