Universidad Diego Portales
Facultad de Ingeniería y Ciencias
Escuela de Informática y Telecomunicaciones

udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA Y CIENCIAS

TAREA 1: Sistema de Cache Distribuido

Integrantes:	Cristóbal Martínez
	José Agustín Pérez
Sección:	02
Profesor Cátedra:	Nicolás Hidalgo
Fecha:	26/09/2023

Índice de Contenidos

Índice de Figuras

1. Introducción

Este documento tiene como objetivo principal mostrar una comparación con respecto a velocidad y eficiencia de tres sistemas distintos, el primero es un sistema de caché desarrollado de manera casera, este sistema ha sido entregado previamente con el objetivo de estudiar sus características y performance, luego lo que se pide es implementar un sistema basado en memcached, este sistema es uno de los más usados actualmente dentro de los sistemas distribuidos, el cual mantiene un desarrollo a gran escala y de un alto rendimiento, por otro lado también se pide implementar un sistema que no utilice ningún tipo de cache, es decir que las peticiones que se hagan, se irán directamente al backend del sistema.

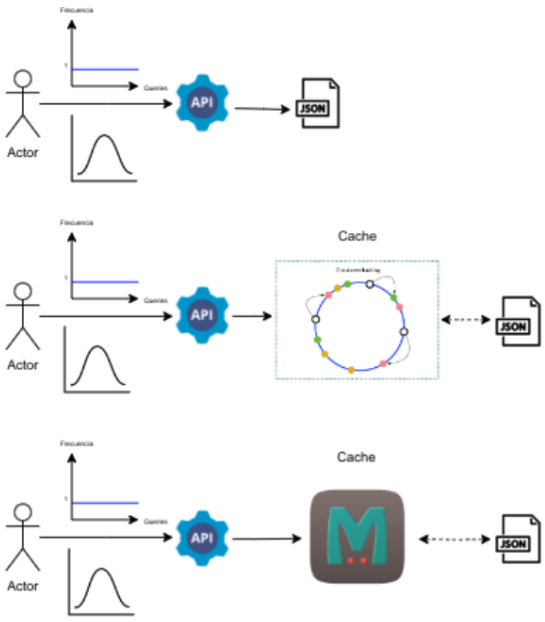

Figura 1: Escenarios a evaluar

La comparación entre estos tres sistemas se concreta por medio de distintas métricas como gráficos y conclusiones que se generen a partir de las mediciones que se realicen dentro del un ambiente similar, para realizar estas pruebas, se hizo entrega de un archivo JSON, que contiene un arreglo de 100 Millones de datos de autos, de esta manera ambos sistemas cuentan con un sistema que permite hacer búsquedas en base al id de un vehículo, si este id no se encuentra dentro del caché, entonces se busca dentro del archivo JSON, el cual simula correctamente el comportamiento de los sistemas cache, además se tendrá una implementación que permitirá simular una cantidad variable de búsquedas para poder comparar ambos sistemas en el caso de que se realicen consultas de manera sistemática.

2. Desarrollo

La primera actividad a realizar es analizar el sistema casero proporcionado, este sirve de ejemplo para observar cómo se comporta un sistema cache normalmente para luego compararlo con la implementación de memcached y el sistema sin cache.

2.1. Sistema sin Cache

El sistema de búsqueda de datos sin caché funciona como un sistema directo que accede y consulta el archivo cars.json en busca de información en tiempo real a través del programa findcarsbyid sin almacenar previamente datos en caché. A diferencia del sistema de caché casero, que optimiza el acceso a datos frecuentemente utilizados, este sistema se basa en la consulta directa al archivo principal de datos en cada solicitud. No utiliza almacenamiento en caché intermedio ni nodos maestro-esclavo. Cada solicitud se traduce en una búsqueda inmediata en el archivo JSON y se devuelve el resultado correspondiente, sin aprovechar la ventaja de la caché para reducir el tiempo de acceso a datos previamente consultados.

En resumen, este sistema realiza búsquedas en el archivo JSON en tiempo real sin utilizar una capa de caché intermedia ni distribución de nodos, lo que lo hace más fácil de implementar, pero potencialmente menos eficiente en términos de tiempos de respuesta para búsquedas frecuentes.

2.2. Sistema Casero

El sistema de caché casero funciona mediante una arquitectura maestro-esclavo, utiliza gRPC para la comunicación entre los nodos y consistent hashing. El nodo maestro actúa como el punto de entrada principal para las solicitudes y dirige cada solicitud al nodo esclavo correspondiente. Los nodos esclavos almacenan datos en caché y responden a solicitudes de inserción, recuperación y eliminación. Cada nodo esclavo gestiona su caché utilizando un algoritmo LRU para mantener los datos más relevantes y eliminar los menos utilizados. Cuando una solicitud llega, el nodo maestro determina si el dato solicitado está en caché; si lo está, se devuelve de inmediato, de lo contrario, se busca en una fuente de datos principal, como un archivo JSON, que en este caso es el archivo cars.json, y se almacena en caché para futuras solicitudes. Este sistema mejora la eficiencia al reducir el tiempo de acceso a datos frecuentemente utilizados y garantizar una distribución uniforme de claves entre nodos esclavos para un rendimiento óptimo.

La forma de ejecutar el sistema de caché casero fue a través de la herramienta Codespaces ofrecida por GitHub. Inicialmente, se configuró un entorno de desarrollo remoto utilizando Codespaces, lo que permitió establecer un ambiente aislado y personalizado para el proyecto. Una vez configurado el entorno, se clonó el repositorio del sistema de caché en el espacio de trabajo de Codespaces. Luego, se ejecutaron los comandos necesarios para iniciar tanto el servidor de caché como el buscador. El servidor de caché se inició con un argumento que especificaba si actuaría como maestro o esclavo. Además, se configuraron las direcciones IP y puertos necesarios para la comunicación entre nodos. Una vez que los nodos estaban en funcionamiento, se pudo interactuar con el sistema de caché utilizando el buscador a través de su interfaz de línea de comandos. Codespaces proporcionó un entorno de desarrollo virtualizado que facilitó la configuración y ejecución del sistema de caché, lo que simplificó el proceso de prueba y demostración de su funcionamiento.

2.3. Sistema Memcached

Después de analizar el comportamiento del caché casero, se procedió a modificarlo para que funcionara utilizando Memcached. Memcached es una herramienta de almacenamiento en caché de alto rendimiento que se emplea para guardar datos en la memoria RAM de forma distribuida. Su propósito principal consiste en acelerar el acceso a datos frecuentemente utilizados al minimizar la necesidad de acceder a una base de datos o fuentes de datos más lentas, lo que resulta en una mejora significativa en el rendimiento y la escalabilidad de las aplicaciones. En este contexto, Memcached se integra con el sistema de caché casero para ofrecer una solución más eficiente y escalable en la gestión de datos en caché.

Las modificaciones realizadas en el código del sistema de caché casero para que funcione con Memcached incluyeron la incorporación de la biblioteca "python-memcache" para gestionar la conexión con el servidor Memcached. Se ajustó la inicialización del cliente de caché para utilizar Memcached, estableciendo la conexión con el servidor Memcached y configurando su dirección y puerto. Además, se adaptaron las funciones "get" y "put" para que interactúen con Memcached en lugar de gestionar el caché internamente. Ahora, cuando se solicita un valor, se busca primero en Memcached, y si no se encuentra, se recupera de la fuente de datos principal y se almacena en Memcached para futuras consultas. Esta modificación permitió aprovechar las ventajas de Memcached, como el almacenamiento en memoria distribuida y el rápido acceso a los datos, mejorando así el rendimiento y la eficiencia del sistema de caché.

3. Resultados Experimentales

Luego de haber medido las métricas simulando las consultas en cada uno de los sistemas, se procede a graficar estos datos en utilizando Graphical Analisys, con el objetivo de encontrar diferencias gráficas entre los sistemas implementados.

	10	20	30	40	50	60	70	80	90	100
Sin	20.71	44.65	65.15	79.89	101.59	117.68	132.55	162.65	183.76	195.86
Cas	30	28	46.51	35.45	33.66	16.3	7.17	4.65	3.2	4.5
Mem	16.32	15.2	19.43	23.77	18.79	24.38	9.15	3.66	2.64	3.66

Figura 2: Tabla con respecto al tiempo

	10	20	30	40	50	60	70	80	90	100
Cas	3	8	10	25	35	52	67	78	88	99
Mem	3	14	21	31	40	56	67	78	89	99

Figura 3: Tabla con respecto a las colisiones

Posteriormente se solicita responder las siguientes preguntas en base a los resultados visualizados en los graficos:

1. **¿Para todos los sistemas es recomendable utilizar cache?**

   No todos los sistemas necesitan o se benefician de una capa de caché. La decisión de utilizar caché en un sistema debe basarse en las siguientes consideraciones:

   - **Rendimiento:** La caché se utiliza comúnmente para mejorar el rendimiento de un sistema al reducir la latencia y acelerar el acceso a los datos. Si el rendimiento es un factor crítico para tu sistema, entonces considerar la caché puede ser una opción.
   - **Patrones de acceso a datos:** Si el sistema tiene patrones de acceso a datos que muestran solicitudes repetitivas de los mismos datos o lecturas frecuentes de datos que rara vez cambian, la caché puede ser beneficiosa para almacenar copias de estos datos en memoria para un acceso más rápido.
   - **Escalabilidad:** En sistemas que deben escalar para atender a un gran número de usuarios o transacciones, la caché puede ayudar a reducir la carga en los recursos subyacentes y mejorar la capacidad de escalabilidad del sistema.
   - **Costos y recursos:** La implementación de una caché agrega costos en términos de recursos de hardware y desarrollo. Debes considerar si los beneficios en términos de rendimiento justifican estos costos.
   - **Consistencia de datos:** En sistemas donde la consistencia de datos es crítica, como aplicaciones financieras o de control de inventario, la caché debe implementarse cuidadosamente para garantizar que los datos en caché estén siempre actualizados y sean coherentes con los datos subyacentes.

- **Tamaño y capacidad de almacenamiento:** Se debe considerar cuántos datos necesitas cachear y cuánta memoria o espacio de almacenamiento tienes disponible para alojar la caché.

  En resumen, la decisión de utilizar caché en un sistema en general depende de los objetivos del sistema, los patrones de acceso a datos, la escalabilidad, los costos y otros factores específicos. No todos los sistemas requieren caché, y su implementación debe ser cuidadosamente evaluada en función de las necesidades y restricciones del sistema en particular.

2. **¿Que ventajas aporta memcached en comparación con nuestro sistema casero en términos de velocidad y eficiencia?**

   Memcached aporta varias ventajas en comparación con el sistema casero en términos de velocidad y eficiencia.

   En primer lugar, Memcached es una solución de almacenamiento en caché en memoria que está diseñada específicamente para acelerar el acceso a datos frecuentemente utilizados. Esto significa que Memcached almacena los datos en la RAM, lo que permite un acceso extremadamente rápido a los mismos. En contraste, un sistema casero puede no estar optimizado para el almacenamiento en memoria, lo que resulta en tiempos de acceso más lentos a los datos.

   En segundo lugar, Memcached es altamente eficiente en términos de uso de recursos. Está diseñado para ser ligero y consume poca memoria y CPU. Esto significa que puede ejecutarse de manera eficiente en sistemas con recursos limitados. Por otro lado, un sistema casero podría requerir más recursos para lograr un rendimiento similar.

   Además, Memcached ofrece una interfaz de almacenamiento en caché fácil de usar y es altamente escalable. Puede distribuirse en clústeres para manejar grandes volúmenes de datos y cargas de trabajo. En comparación, la implementación de un sistema casero podría requerir una cantidad significativa de esfuerzo en desarrollo y mantenimiento para lograr la misma escalabilidad y eficiencia.

   En resumen, Memcached proporciona ventajas en términos de velocidad y eficiencia en comparación con un sistema casero, gracias a su diseño optimizado para el almacenamiento en caché en memoria, su eficiencia en el uso de recursos y su escalabilidad.

3. **Enumera y describe tres caracterısticas avanzadas que memcached ofrece que nuestro sistema casero no posee.**

   Memcached ofrece varias características avanzadas que el sistema casero no posee, las tres principales son las siguientes:

   - **Almacenamiento en Memoria Distribuida:** Memcached está diseñado para almacenar datos en memoria distribuida, lo que significa que puede manejar grandes volúmenes de datos en la memoria RAM de varios servidores. Esto permite un acceso extremadamente rápido a los datos en comparación con sistemas que dependen del almacenamiento en disco o bases de datos tradicionales. Nuestro sistema casero, en contraste, no se beneficia de esta característica y opera principalmente en un solo nodo.

- **Escalabilidad Horizontal:** Memcached es altamente escalable de manera horizontal. Puede agregar fácilmente más servidores Memcached a medida que crece la carga de trabajo, distribuyendo la carga de manera uniforme entre los nodos. Esto asegura que el sistema pueda manejar un alto volumen de solicitudes y mantener un rendimiento constante. En nuestro sistema casero, la escalabilidad es más limitada y requeriría una implementación adicional para lograr un nivel similar de escalabilidad.
- **Protocolo de Red Eficiente:** Memcached utiliza un protocolo de red altamente eficiente que minimiza la sobrecarga de comunicación y el uso de ancho de banda. Esto significa que las solicitudes y respuestas entre clientes y servidores Memcached son rápidas y consumen menos recursos de red en comparación con otros sistemas. Nuestro sistema casero, si bien utiliza gRPC para la comunicación entre nodos, puede no estar optimizado de la misma manera que Memcached en términos de eficiencia de red.

4. **¿Como se podrıa mejorar nuestro sistema casero para que se acerque mas a la robustez y funcionalidad de memcached?**

   Para mejorar nuestro sistema casero y acercarlo más a la robustez y funcionalidad de Memcached, podríamos considerar las siguientes mejoras:

   - **Implementación de Almacenamiento en Memoria Distribuida:** Para mejorar significativamente nuestro sistema casero y acercarlo a la robustez de Memcached, podríamos implementar un sistema de almacenamiento en memoria distribuida. Esto implicaría dividir los datos en caché en múltiples servidores o nodos para aprovechar la capacidad de escalabilidad horizontal. Cada nodo podría ser responsable de un conjunto específico de claves, lo que permitiría manejar una mayor carga de trabajo y redundancia de datos. Además, podríamos incorporar técnicas de partición de datos y equilibrio de carga para garantizar una distribución uniforme de las claves entre los nodos y evitar cuellos de botella.
   - **Manejo de Fallas Automático y Tolerancia a Fallas:** Para lograr una mayor robustez, sería esencial implementar un mecanismo de manejo de fallas automático. Esto implicaría la detección automática de nodos caídos o inaccesibles y la redistribución de la carga a nodos activos. También podríamos incorporar la replicación de datos entre nodos para garantizar la disponibilidad incluso en caso de fallas de hardware. Además, podríamos establecer un sistema de tolerancia a fallas que permita que el sistema siga funcionando incluso si algunos nodos experimentan problemas. Esto aseguraría una alta disponibilidad y resistencia a interrupciones.
   - **Optimización del Protocolo de Comunicación y Compresión de Datos:** Para mejorar la eficiencia y el rendimiento del sistema, podríamos realizar mejoras en el protocolo de comunicación entre nodos. Esto podría incluir la implementación de un protocolo más eficiente y liviano que minimice la sobrecarga de red y reduzca la latencia. Además, podríamos incorporar técnicas de compresión de datos para reducir el tamaño de las transferencias de datos entre nodos, lo que resultaría en un uso más eficiente de los recursos de red y una mayor velocidad de comunicación.

5. **¿En que situaciones considerarıas apropiado utilizar nuestro sistema de cache casero en lugar de una solución como memcached?**

   El uso de nuestro sistema de caché casero en lugar de una solución como Memcached podría considerarse apropiado en las siguientes situaciones:

   - **Entorno de Desarrollo o Pruebas:**Nuestro sistema de caché casero podría ser una elección adecuada en entornos de desarrollo o pruebas donde se necesita una solución de caché simple para experimentar y realizar pruebas de concepto. Su simplicidad y facilidad de configuración serían ventajas en estos casos, ya que permitirían a los desarrolladores implementar rápidamente un sistema de caché para evaluar su comportamiento.
   - **Aplicaciones de Pequeña Escala:** En aplicaciones de pequeña escala con requisitos de caché modestos, donde la infraestructura y la escalabilidad no son una preocupación principal, nuestro sistema casero podría ser suficiente. Si la cantidad de datos a almacenar en caché es limitada y la carga de trabajo es manejable, no sería necesario implementar una solución más compleja como Memcached.
   - **Aprendizaje y Educación:** Nuestro sistema de caché casero podría ser útil en entornos educativos o de aprendizaje, donde los estudiantes están explorando conceptos de almacenamiento en caché y sistemas distribuidos. Proporciona una base sólida para comprender los principios básicos de cómo funcionan los sistemas de caché antes de aventurarse en soluciones más avanzadas como Memcached.
   - **Aplicaciones con Recursos Limitados:** En dispositivos o sistemas con recursos limitados, donde la sobrecarga de recursos por parte de Memcached podría ser un problema, nuestro sistema casero podría ser una opción viable. Su diseño más ligero y su menor uso de recursos podrían ser beneficiosos en tales escenarios.

   En resumen, nuestro sistema de caché casero podría ser apropiado en situaciones donde se valora la simplicidad, la facilidad de configuración y el bajo consumo de recursos, y donde los requisitos de escalabilidad y rendimiento no son críticos. Sin embargo, es importante tener en cuenta que Memcached ofrece una mayor robustez, escalabilidad y rendimiento, por lo que sigue siendo la elección preferida en entornos empresariales de alta demanda y aplicaciones que requieren una infraestructura de caché distribuida más sofisticada.

4. Comparación de resultados

Para realizar una comparación de resultados más gráfica, se colocan los datos obtenidos en las tablas dentro del graphical análisis

4.1. Comparación de tiempos

Se puede notar en la gráfica que el sistema sin cache (línea roja) se demora más tiempo en realizar las consultas con respecto a ambos sistemas cache. Además se puede observar que aunque ambos cache tienen tiempos similares, el sistema basado en memcache (línea amarilla) tiende a ser más rápido que el sistema de cache implementado de forma casera (línea azul)

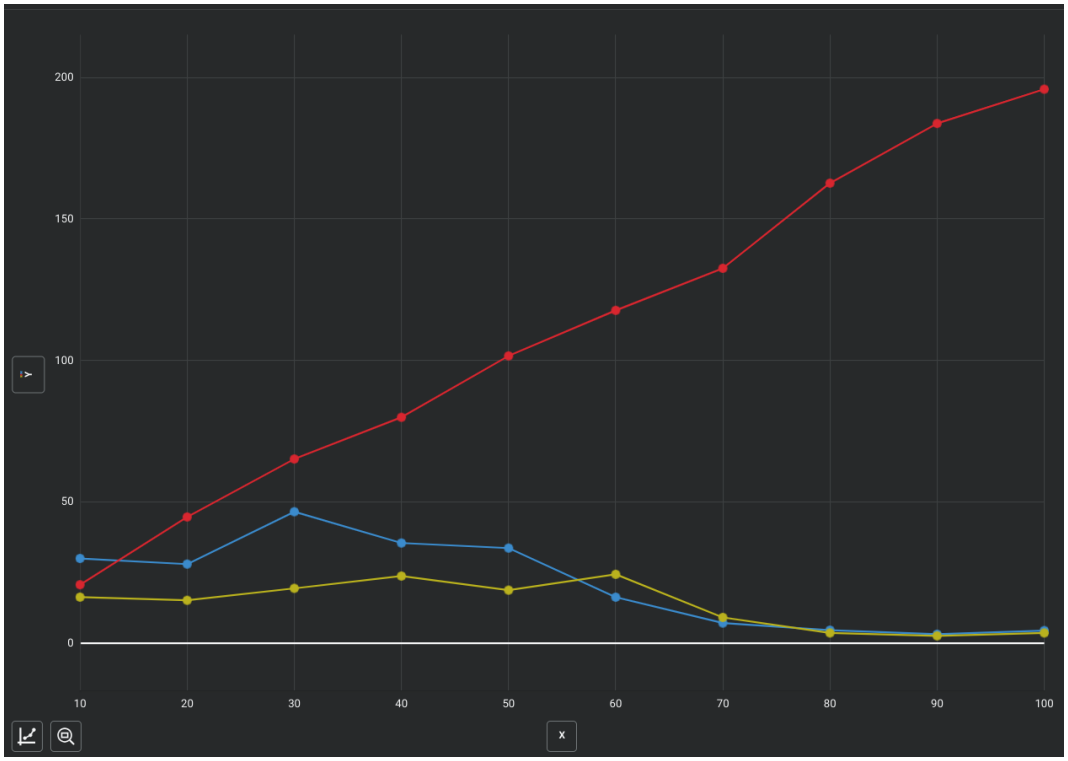

Figura 4: Gráfico de comparación de tiempos

4.2. Comparación de colisiones

En el contexto de la figura que se presenta, se observa claramente que el sistema basado en Memcached (representado por la línea amarilla) experimenta más colisiones de caché en comparación con el sistema sin caché (línea azul) durante las simulaciones realizadas.

Las ”colisiones de caché” se refieren a situaciones en las que dos o más solicitudes de acceso a datos intentan acceder a la misma ubicación o conjunto de datos en la caché al mismo tiempo. Cuando se produce una colisión de caché, el sistema debe tomar medidas para resolverla, lo que puede incluir la invalidación de datos almacenados en la caché o la espera de que se complete una operación antes de continuar. Estas colisiones pueden tener un impacto negativo en el rendimiento del sistema, ya que pueden resultar en un mayor tiempo de respuesta y una menor eficiencia.

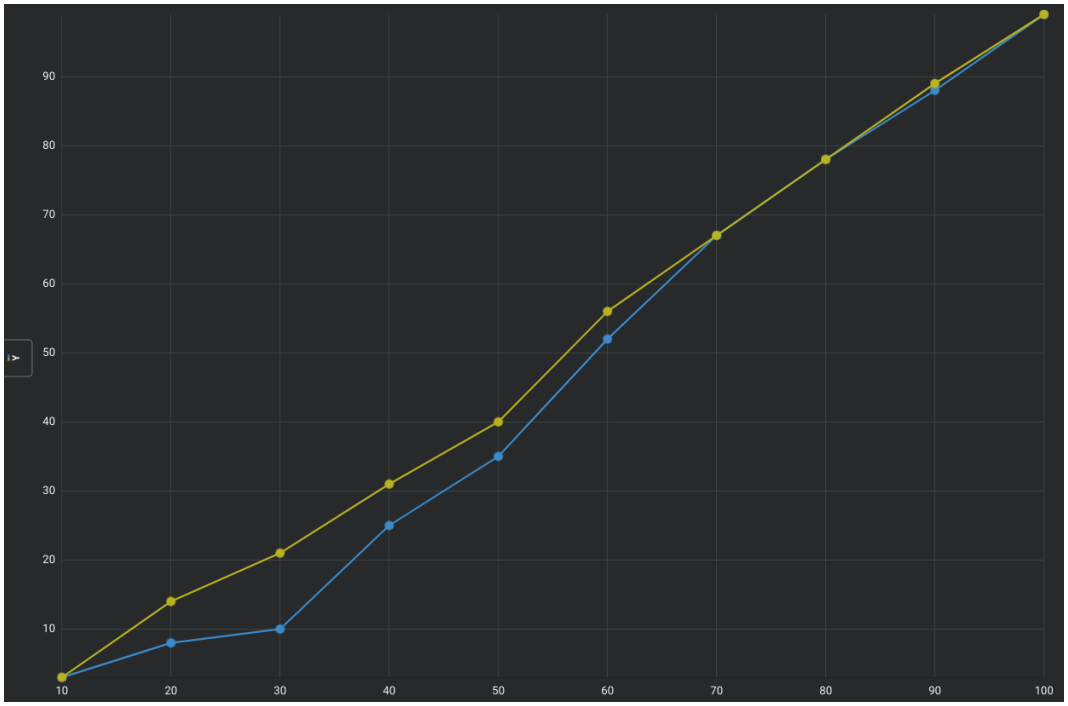

Figura 5: Gráfico de comparación de colisiones

5. Conclusiones

En esta comparación de sistemas de almacenamiento en caché, se evaluaron tres enfoques diferentes: el sistema casero, Memcached y un sistema sin caché. El sistema casero se destacó por su arquitectura maestro-esclavo y el uso de gRPC, lo que permitió una distribución eficiente de las solicitudes y el almacenamiento en caché de datos. Esto resultó en una mejora significativa en la eficiencia y el rendimiento al reducir el tiempo de acceso a datos frecuentemente utilizados.

La integración de Memcached en el sistema casero representó otra mejora sustancial. Memcached, con su almacenamiento en memoria distribuida, aceleró aún más el acceso a datos al minimizar la necesidad de acceder a fuentes de datos más lentas, como bases de datos. Esto hizo que el sistema fuera aún más eficiente y escalable.

Por otro lado, el sistema sin caché, que consultaba directamente el archivo JSON, era más sencillo de implementar pero menos eficiente en términos de tiempos de respuesta para búsquedas frecuentes. En resumen, la elección del sistema de caché dependerá de las necesidades específicas de rendimiento y escalabilidad de una aplicación o sistema distribuido, con el sistema casero y Memcached destacándose por su capacidad para mejorar significativamente la eficiencia y el rendimiento.

6. Referencias

- Contenedor de docker de Memcached: https://hub.docker.com/_/memcached
- Repositorio Github Tarea1: https://github.com/Cristobal1202/Tarea1_Martinez_Perez.git