

UNIVERSIDAD DIEGO PORTALES
ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

Tarea 1: Sistema de cache distribuido

Profesor: Nicolas Hidalgo

Ayudantes: Nicolas Nuñez
Felipe Ulloa

Alumnos: Pablo Ahumada Villegas
Iman Jarufe Alcantar

Índice de contenidos

1. Introducción	2
2. Metodología	2
3. Resultados.	3
4. Análisis comparativo.	9
5. Preguntas de Investigación	11
6. Conclusiones	12
7. Anexos	13

1. Introducción

Se cuenta con el siguiente problema: los estudiantes del primer semestre del ramo de sistemas distribuidos implementaron un sistema de cache casero, pero se olvidaron de realizar comparaciones con un sistema de Memcached. Por lo tanto, el profesor les solicitó a los alumnos de segundo semestre que completaran el trabajo faltante utilizando las herramientas desarrolladas y entregadas por los alumnos anteriores. Para ello, se les entrega un sistema de cache básico que utiliza gRPC y un archivo JSON con 100 entradas, como base de datos.

Una vez mencionado el contexto general, este informe presenta los resultados obtenidos al realizar la Tarea 1 de Sistemas Distribuidos, que consiste en realizar tres tipos de búsquedas y compararlas. La primera consiste en realizar una búsqueda directamente desde el archivo JSON, la segunda en una búsqueda con cache casero y la ultima en una búsqueda con Memcached. El objetivo principal es aprender los beneficios y desafíos del uso y la generación de un sistema de cache, evaluando y comparando eficacia, robustez y características de un sistema de cache casero y una solución distribuida como Memcached.

2. Metodología

Para comenzar, se presenta el código en Python que se realiza la búsqueda directa, utilizada en las tres situaciones a comparar. Este código ha sido modificado, dado que el número de entradas en el cache es mayor o igual a el número de entradas de la base de datos (archivo JSON), y el tamaño de este ultimo es muy pequeño. Por ende, se ha decidido agregar un pequeño retraso de 0.01 segundos en el algoritmo de búsqueda para simular una base de datos mas grande que el cache casero.

Este ajuste se ha implementado porque no tiene mucho sentido realizar una comparación entre ambos en esta situación, ya que la búsqueda directa toma muy poco tiempo y es similar a la que se encuentran en el cache casero, con la diferencia de que este último tarda más al tener que ingresar la entrada al cache. Esto sin contar la diferencia de tiempo de comunicación por estar distribuido el sistema de cache, al contrario de la búsqueda directa, ya que los datos se encuentran en el mismo lugar o contenedor.

```
1 import json
2 import time
3 def find_car_by_id(target_id, file_path="./cars.json"):
4     with open(file_path, 'r') as f:
5         time.sleep(0.01) # Delay para simular archivo mas grande
6         data = json.load(f)
7
8         min, max = 0, len(data) - 1
9         while min <= max:
10             mid = (min + max) // 2
11             if data[mid]['id'] == target_id:
12                 return data[mid]
13             elif data[mid]['id'] < target_id:
14                 min = mid + 1
15             else:
16                 max = mid - 1
17         return None
```

Luego para realizar la búsqueda con memcached, se utiliza el siguiente código, el cual se describe a continuación.

```
1 from pymemcache.client import base
2 class MemClient:
3     def __init__(self, host="memcached", port=11211):
4         self.mclient = base.Client((host, port))
5
6     def get(self, key):
7         start_time = time.time() # Inicio del temporizador
8         response = self.mclient.get(key) # buscar en cache
9
10        if response: # Si encuentra la entrada en el cache
11            elapsed_time = time.time() - start_time # Fin del temporizador
12            return response
13
14        else: # Si no encuentra la entrada en el cache
15            value = find_car_by_id(int(key)) # buscar en base de datos
16            if value:
```

```
17         self.mclient.set(key, value) # ingresar al cache
18         elapsed_time = time.time() - start_time # Fin del temporizador
19         return value
```

Se inicia el cliente de memcached extraído de la librería pymemcache la cual también contiene el método get para buscar en el cache insertando una llave. En el caso de no ser encontrada no entrega ningún resultado y se procede a buscar directamente en la base de datos para posteriormente ingresar la entrada al cache.

El código completo se encuentra en el repositorio entregado.

Para los métricas a analizar se toma en cuenta el tiempo desde que comienza la búsqueda, hasta que encuentra el resultado; además de la efectividad del cache.

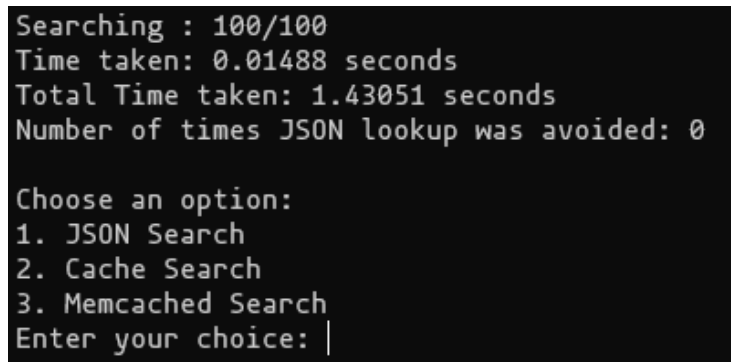
3. Resultados.

Realizados los pasos evidenciados en la metodología, se presentan los siguientes resultados, los cuales serán analizados en el apartado siguiente.

En las figuras presentes se evidencian las búsquedas mencionadas al comienzo del presente informe. Para cada tipo de búsqueda, realizamos 4 búsquedas con diferentes cantidades de datos (100, 300, 500 y 1000); una vez realizado esto se presentan los resultados obtenidos para cada caso en las siguientes figuras.

En primera instancia, se muestran las imágenes correspondientes a la realización de múltiples búsquedas mediante la búsqueda directa.

En la figura 1, se muestra el tiempo requerido para realizar una búsqueda directa, es decir, directamente al archivo JSON, sin considerar cache, en donde se realizan 100 búsquedas tomando un tiempo de 1,43051 segundos.



```
Searching : 100/100
Time taken: 0.01488 seconds
Total Time taken: 1.43051 seconds
Number of times JSON lookup was avoided: 0

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 1: Resultado de 100 búsquedas directas al archivo JSON.

En la figura 2, se muestra el tiempo requerido para realizar una búsqueda directa, es decir, directamente al archivo JSON, sin considerar cache, en donde se realizan 300 búsquedas tomando un tiempo de 4,97903 segundos.

```
Searching : 300/300
Time taken: 0.01711 seconds
Total Time taken: 4.97903 seconds
Number of times JSON lookup was avoided: 0

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 2: Resultado de 300 búsquedas directas al archivo JSON.

En la figura 3, se muestra el tiempo requerido para realizar una búsqueda directa, es decir, directamente al archivo JSON, sin considerar cache, en donde se realizan 500 búsquedas tomando un tiempo de 8,28584 segundos.

```
Searching : 500/500
Time taken: 0.01659 seconds
Total Time taken: 8.28584 seconds
Number of times JSON lookup was avoided: 0

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 3: Resultado de 500 búsquedas directas al archivo JSON.

En la figura 4, se muestra el tiempo requerido para realizar una búsqueda directa, es decir, directamente al archivo JSON, sin considerar cache, en donde se realizan 1000 búsquedas tomando un tiempo de 16,55267 segundos.

```
Searching : 1000/1000
Time taken: 0.01633 seconds
Total Time taken: 16.55267 seconds
Number of times JSON lookup was avoided: 0

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice:
```

Figura 4: Resultado de 1000 búsquedas directas al archivo JSON.

Luego, se muestran las imágenes correspondientes a realizar múltiples búsquedas mediante la búsqueda con el cache casero.

En la figura 5, se muestra el tiempo requerido para realizar una búsqueda con el cache casero, en donde se realizan 100 búsquedas tomando un tiempo de 1,97094 segundos.

```
Searching : 100/100
Time taken (cache): 0.00562 seconds
Total Time taken: 1.97094 seconds
Number of times JSON lookup was avoided: 41

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 5: Resultado de 100 búsquedas con el cache casero.

En la figura 6, se muestra el tiempo requerido para realizar una búsqueda con el cache casero, en donde se realizan 300 búsquedas tomando un tiempo de 3,92224 segundos.

```
Searching : 300/300
Time taken (cache): 0.00600 seconds
Total Time taken: 3.92224 seconds
Number of times JSON lookup was avoided: 205

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice:
```

Figura 6: Resultado de 300 búsquedas con el cache casero.

En la figura 7, se muestra el tiempo requerido para realizar una búsqueda con el cache casero, en donde se realizan 500 búsquedas tomando un tiempo de 5,25438 segundos.

```
Searching : 500/500
Time taken (cache): 0.00599 seconds
Total Time taken: 5.25438 seconds
Number of times JSON lookup was avoided: 400

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 7: Resultado de 500 búsquedas con el cache casero.

En la figura 8, se muestra el tiempo requerido para realizar una búsqueda con el cache casero, en donde se realizan 1000 búsquedas tomando un tiempo de 8,39922 segundos.

```
Searching : 1000/1000
Time taken (cache): 0.00575 seconds
Total Time taken: 8.39922 seconds
Number of times JSON lookup was avoided: 900

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 8: Resultado de 1000 búsquedas con el cache casero.

Para continuar, se muestran las imágenes correspondientes a realizar múltiples búsquedas mediante la búsqueda con memcached en donde no se encuentran todas las llaves en el cache.

En la figura 9, se muestra el tiempo requerido para realizar una búsqueda con memcached (en donde no se encuentran todas las llaves en él), realizando 100 búsquedas tomando un tiempo de 3,86350 segundos.

```
Searching : 100/100
Time taken: 0.01719 seconds
Total Time taken: 3.86350 seconds
Number of times JSON lookup was avoided: 35

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 9: Resultado de 100 búsquedas con memcached sin todas las llaves.

En la figura 10, se muestra el tiempo requerido para realizar una búsqueda con memcached (en donde no se encuentran todas las llaves en él), realizando 300 búsquedas tomando un tiempo de 5,77110 segundos.

```
Searching : 300/300
Time taken (cache): 0.00020 seconds
Total Time taken: 5.77110 seconds
Number of times JSON lookup was avoided: 205

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 10: Resultado de 300 búsquedas con memcached sin todas las llaves.

En la figura 11, se muestra el tiempo requerido para realizar una búsqueda con memcached (en donde no se encuentran todas las llaves en él), realizando 500 búsquedas tomando un tiempo de 6,16084 segundos.

```
Searching : 500/500
Time taken (cache): 0.00022 seconds
Total Time taken: 6.16084 seconds
Number of times JSON lookup was avoided: 400

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 11: Resultado de 500 búsquedas con memcached sin todas las llaves.

En la figura 12, se muestra el tiempo requerido para realizar una búsqueda con memcached (en donde no se encuentran todas las llaves en él), realizando 1000 búsquedas tomando un tiempo de 6,38178 segundos.

```
Searching : 1000/1000
Time taken (cache): 0.00028 seconds
Total Time taken: 6.38178 seconds
Number of times JSON lookup was avoided: 900

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 12: Resultado de 1000 búsquedas con memcached sin todas las llaves.

Finalmente, se muestran las imágenes correspondientes a realizar múltiples búsquedas mediante la búsqueda con memcached en donde si se encuentran todas las llaves en el cache.

En la figura 13, se muestra el tiempo requerido para realizar una búsqueda con memcached(en donde si se encuentran todas las llaves en él), realizando 100 búsquedas tomando un tiempo de 0,03955 segundos.

```
Searching : 100/100
Time taken (cache): 0.00023 seconds
Total Time taken: 0.03955 seconds
Number of times JSON lookup was avoided: 100

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 13: Resultado de 100 búsquedas con memcached con todas las llaves.

En la figura 14, se muestra el tiempo requerido para realizar una búsqueda con memcached(en donde si se encuentran todas las llaves en él), realizando 300 búsquedas tomando un tiempo de 0,11951 segundos.

```
Searching : 300/300
Time taken (cache): 0.00033 seconds
Total Time taken: 0.11951 seconds
Number of times JSON lookup was avoided: 300

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 14: Resultado de 300 búsquedas con memcached con todas las llaves.

En la figura 15, se muestra el tiempo requerido para realizar una búsqueda con memcached(en donde si se encuentran todas las llaves en él), realizando 500 búsquedas tomando un tiempo de 0,20992 segundos.

```
Searching : 500/500
Time taken (cache): 0.00025 seconds
Total Time taken: 0.20992 seconds
Number of times JSON lookup was avoided: 500

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 15: Resultado de 500 búsquedas con memcached con todas las llaves.

En la figura 16, se muestra el tiempo requerido para realizar una búsqueda con memcached(en donde si se encuentran todas las llaves en él), realizando 1000 búsquedas tomando un tiempo de 0,40829 segundos.

```
Searching : 1000/1000
Time taken (cache): 0.00028 seconds
Total Time taken: 0.40829 seconds
Number of times JSON lookup was avoided: 1000

Choose an option:
1. JSON Search
2. Cache Search
3. Memcached Search
Enter your choice: |
```

Figura 16: Resultado de 1000 búsquedas con memcached con todas las llaves.

4. Análisis comparativo.

Una vez obtenidos los resultados, se realizan análisis comparativos en base a las tres búsquedas realizadas.

Para realizar el análisis comparativo se crean tablas y gráficos, en donde se tabulan y grafican los tiempos obtenidos al momento de realizar múltiples búsquedas en los tres tipos de búsquedas requeridos.

En primera instancia se presentan las siguientes tres tablas, en donde se tabulan los tiempos de búsqueda asociados a 4 cantidades de búsquedas específicas en la búsqueda directa, la búsqueda con cache casero y con memcached.

Cantidad de búsquedas	Tiempo requerido
100 búsquedas	1,43051 segundos
300 búsquedas	4,97903 segundos
500 búsquedas	8,28584 segundos
1000 búsquedas	16,55267 segundos

Cuadro 1: Tabla comparativa de tiempo de búsquedas con búsqueda directa.

Cantidad de búsquedas	Tiempo requerido
100 búsquedas	1,97094 segundos
300 búsquedas	3,92224 segundos
500 búsquedas	5,25438 segundos
1000 búsquedas	8,39922 segundos

Cuadro 2: Tabla comparativa de tiempo de búsquedas con cache casero.

Cantidad de búsquedas	Tiempo requerido
100 búsquedas	3,86350 segundos
300 búsquedas	5,77110 segundos
500 búsquedas	6,16084 segundos
1000 búsquedas	6,38178 segundos

Cuadro 3: Tabla comparativa de tiempo de búsquedas con memcached sin todas las llaves en él.

Cantidad de búsquedas	Tiempo requerido
100 búsquedas	0,03955 segundos
300 búsquedas	0,11951 segundos
500 búsquedas	0,20992 segundos
1000 búsquedas	0,40829 segundos

Cuadro 4: Tabla comparativa de tiempo de búsquedas con memcached con todas las llaves en él.

Una vez evidenciadas las tablas, se procede a mostrar los respectivos gráficos generados a partir de cada una de las tablas.



Figura 17: Gráfico generado para los resultados de la búsqueda directa.

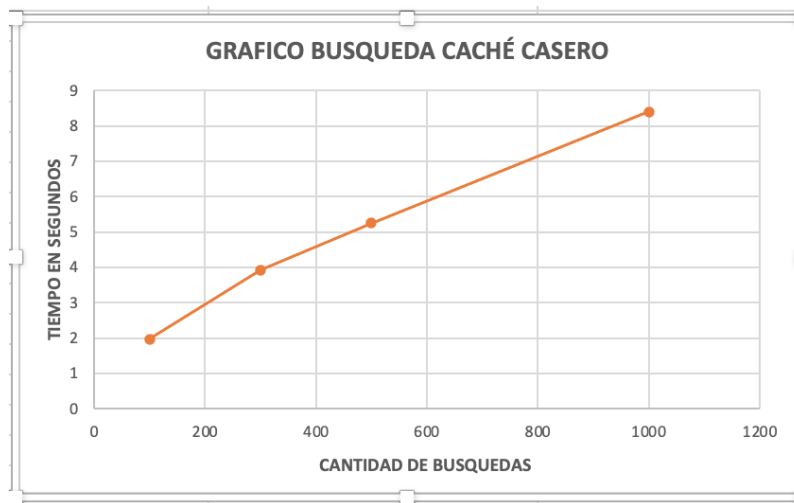


Figura 18: Gráfico generado para los resultados de la búsqueda con cache casero.

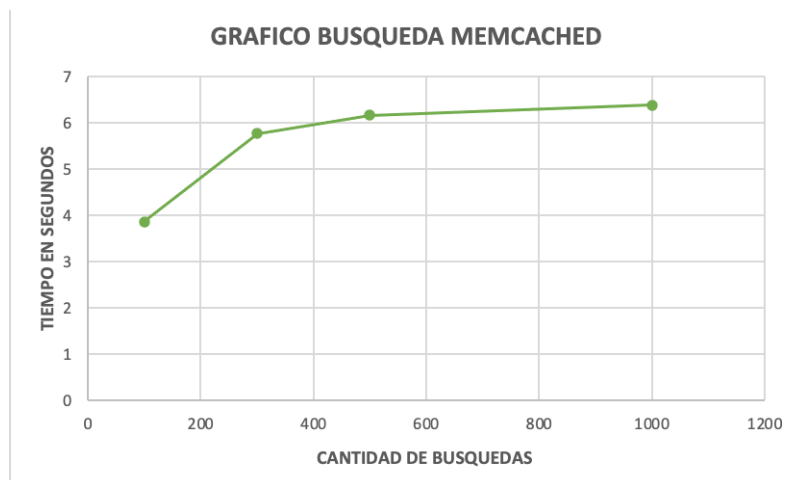


Figura 19: Gráfico generado para los resultados de la búsqueda con memcached sin todas las llaves en él.

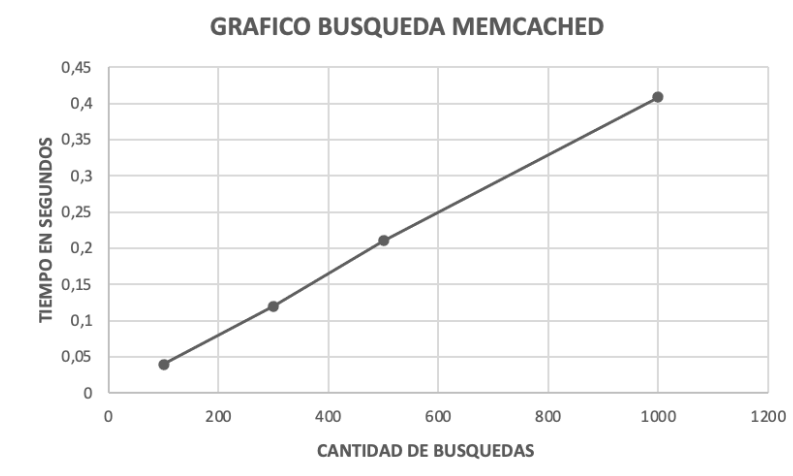


Figura 20: Gráfico generado para los resultados de la búsqueda con memcached con todas las llaves en él.

Mostrados los resultados obtenidos al momento de realizar las búsquedas solicitadas, podemos ver que existen 4 tablas y 4 gráficos, los cuales se implementan en base a los datos tabulados. Cada una de la tablas conforma el tiempo de ejecución de realizar la cantidad de búsquedas requeridas, para este caso se implementaron cuatro (100, 300, 500, 1000). Como podemos ver a medida que se utiliza tanto cache casero como memcached los tiempos entregados para cada cantidad de búsquedas va disminuyendo (considerablemente cuando se utiliza memcached). Además podemos ver que la primera tabla de memcached se demora un poco mas en algunas búsquedas, ya que esto sucede cuando no se encuentran todas las llaves dentro del memcached, tomando mas tiempo en buscarlas, ya que debe primero almacenarlas, pero luego en la tabla que le sigue se aprecia que el tiempo disminuye considerablemente al tener todas las llaves en él.

Estos tiempos disminuyen a medida que se utiliza cache, ya que proporciona un acceso rápido y eficiente a las llaves que ya se se han buscado previamente, lo que reduce la necesidad de buscar esos datos en ubicaciones de almacenamiento más lentas.

En cuanto a los gráficos, se aprecia una curva creciente a medida que se realizan mas cantidades de búsquedas, lo cual tiene sentido, ya que obviamente se demora mas tiempo en buscar pero aun así el tiempo es menor al utilizar cache y aun mas al utilizar memcached como se puede apreciar en el eje y, el cual hace referencia a los tiempos requeridos.

5. Preguntas de Investigación

En el presente apartado se nos solicita realizar un análisis detallado de las características de memcached en comparación con el cache casero respondiendo las siguientes preguntas de investigación.

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

R: Generalmente, no es recomendable utilizar cache en todos los sistemas de manera indiscriminada, ya que no es una solución universal y debe implementarse de manera estratégica cuando este sea un beneficio tanto en rendimiento como eficiencia.

Por ende, la decisión de utilizar cache en un sistema debe basarse en un análisis cuidadoso de los requisitos y las características específicas de tu aplicación, tomando en cuenta alguno de los siguientes factores, escalabilidad, costos y recursos, complejidad, etc.

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

R: Para responder a esta pregunta, hablaremos en base a los resultados obtenidos anteriormente. Por ende podemos decir que memcached es muchísimo mas rápido en cuanto a velocidad de búsqueda que nuestro cache casero, lo cual por consecuencia tiende a ser mas eficiente al momento de evaluar los tiempos de respuesta evidenciados por ambos, ya que memcached es diseñado específicamente para el almacenamiento en cache en memoria RAM y su capacidad para manejar grandes volúmenes de datos y solicitudes simultáneas lo convierten en una elección sólida para optimizar el rendimiento de las aplicaciones web y sistemas distribuidos.

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

R: Memcached ofrece muchas características avanzadas en comparación con un cache casero, pero para responder a esta pregunta se nombran los siguientes: escalabilidad horizontal, replicación y soporte para protocolos avanzados. Estas características pueden ser beneficiosas en aplicaciones con requisitos de cache estándar.

- a) **Escalabilidad horizontal:** Memcached es altamente escalable horizontalmente, lo que significa que se pueden agregar nodos adicionales para aumentar la capacidad de almacenamiento en cache y manejar cargas de trabajo crecientes.
- b) **Replicación:** Memcached admite replicación, permitiendo tener múltiples copias de los datos en diferentes nodos, lo cual mejora la disponibilidad y la tolerancia a fallos.
- c) **Soporte para protocolos avanzados:** Memcached admite protocolos avanzados, como ASCII y binary protocol, lo que puede mejorar la interoperabilidad y el rendimiento en ciertos casos de uso.

4. **¿Cómo se podría mejorar nuestro sistema casero para que se acerque más a la robustez y funcionalidad de memcached?**

R: Para mejorar nuestro cache casero Se podría utilizar memoria RAM de alta velocidad y baja latencia para almacenar datos en cache y sistemas de almacenamiento de alto rendimiento para almacenar datos en cache si es necesario. También se puede diseñar un plan de respaldo y recuperación en caso de fallos en el cache casero, incluyendo la replicación de datos o la implementación de un mecanismo de recuperación en caso de pérdida de datos en cache.

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

R: Una situación en la cual se podría utilizar cache casero, en vez de memcached seria cuando se este desarrollando una aplicación o un programa local(el cual se ejecuta en una sola maquina) como un programa de procesamiento de imágenes o un videojuego ya que este es fundamental para acelerar la ejecución del código y mejorar la eficiencia del procesador. Otra situación podría ser al desarrollar una aplicación de alto rendimiento que se ejecuta en una sola máquina y requiere un acceso rápido a datos e instrucciones locales el cache casero es el ideal, ya que memcached se utiliza en escenarios distribuidos en los que varios servidores comparten datos en cache.

6. Conclusiones

Para concluir el presente informe se puede decir en primera instancia que las tres búsquedas realizadas son diferentes y cada una de ella tiene ventajas y desventajas, lo que significa que la elección de cada una de estas depende de una serie de factores claves y la necesidades específicas de la aplicación, los recursos y las prioridades de rendimiento.

Podemos decir que una búsqueda directa es útil en situaciones donde el dominio de búsqueda es pequeño, con búsquedas puntuales y se requiere economizar recursos. En cambio una búsqueda con cache casero se usa para aplicaciones de menor escala o donde la configuración de un sistema de cache personalizado es preferible por razones específicas, y finalmente una búsqueda con memcached se usa para mejorar el rendimiento y la escalabilidad en aplicaciones y escenarios donde se requiere un acceso rápido a datos previamente almacenados en cache.

Si hablamos en temas de tiempo y eficiencia se puede concluir que los tiempos de búsqueda son menores al momento de utilizar un cache casero y aun mas diminutos al utilizar memcached en comparación con una búsqueda

directa, por ende, utilizar cache cuando sea necesario es mucho mas eficiente que realizar una búsqueda directa sobre todo al momento de tener una base de datos aun mayor a la cual se nos entregó en el presente trabajo.

Esos resultados se obtienen debido a que un cache casero es más eficiente que una búsqueda directa en contextos como acceso mas rápido a datos frecuentemente utilizados, políticas de almacenamiento, menos dependencia de recursos externos, etc debido a cómo este almacena y recupera datos previamente accedidos. Además, memcached es más eficiente que un cache casero debido a su diseño específico para el almacenamiento en memoria RAM, su protocolo de comunicación eficiente, la gestión automática de memoria, la escalabilidad y el soporte comunitario. Memcached es una solución probada y confiable que se utiliza ampliamente para mejorar el rendimiento de aplicaciones web y sistemas distribuidos.

Finalmente, una búsqueda con algún sistema de cache es mas eficiente y rápida que una búsqueda directa a un archivo en especifico, y mientras mas datos tenga el archivo, es mas recomendable y útil generar búsquedas con memcached.

7. Anexos

- Repositorio Guía: <https://github.com/Naikelin/sd/tree/main/t1>
- Repositorio Usado: <https://github.com/pixal-lab/SD-T1>
- Vídeo de instalación, uso y explicación: <https://youtu.be/0eZUTX8JQXg>