ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA
Universitat Rovira i Virgili

[DΣIM]

Sistemas Distribuidos

Práctica 2

curso 2023-24

Estudiantes: Aleix Carillo e Ivan Arenal

Profesor/a: Usama Benabdelkrim

Fecha de entrega: 16/06/24

Abstract

En esta práctica se muestran dos implementaciones, una centralizada y la otra descentralizada, para un sistema distribuido de almacenaje de parejas clave valor. En ambas implementaciones se dispone de 3 nodos que actúan como servidores. Por un lado, en la implementación centralizada, tenemos un nodo máster y los otros dos son “esclavos”. En esta, se implementa un protocolo “Two-Phase Commit” (2FC) para garantizar la consistencia del sistema, mediante comunicaciones gRPC entre los nodos. Por otro lado, en la implementación descentralizada los tres nodos tienen el mismo valor y realizan por igual las operaciones. No obstante, estas no se llevan a cabo hasta haber garantizado un “quorum”, que es un valor mínimo de votos para realizar la operación en concreto. Para garantizar esto, se ha implementado un protocolo basado en quórums (quórums protocol), mediante comunicaciones gRPC entre los nodos.

Decisiones de diseño

Implementación centralizada

Para esta implementación se han implementado dos clases, una para el servidor o nodo máster y otra para los nodos esclavos. En ambas encontramos el método “init()” en el cual se inicializan las variables empleadas para el servidor y el diccionario de clave valor para cada una de las clases. Este se inicializa mediante un fichero .json creado a modo de recuperación en caso de tener algún fallo en el sistema. Este se actualiza a cada petición de “put” que se realiza para actualizar los valores. En el caso del servidor máster, no se ha podido implementar la función de descubrimiento de puertos para asignarlos a los servidores y estos se han asignado manualmente como se observa en el código entregado. No obstante, pese a no haber podido realizar la implementación, se conoce la forma en la cual se hubiese hecho. Por tanto, el servidor máster corre en el puerto 32770 mientras que los otros dos servidores esclavos se tendrían que descubrir con el método nombrado anteriormente, que corren por los puertos 32771 y 32772.

Para la implementación de las operaciones get(), put() y canCommit() se ha decido utilizar semáforos, para garantizar que no haya múltiples accesos a las variables mediante otros hilos que no estén ejecutando esa operación en concreto. Como se observa en el código, se usan las funciones “acquire()” y “release()”, para bloquear y liberar este acceso nombrado anteriormente. Se ha implementado en estas, ya que, las hemos considerado críticas y que pueden llegar a afectar negativamente, en caso de múltiples accesos, al sistema. Por otro lado, para las funciones “restore()” y “slowDown()”, no se ha optado por el uso de semáforos, ya que, son funciones “no críticas” que reinician valores de variables o aplican valores determinados a estas.

Cabe destacar que, en la función put() de los servidores esclavos se almacena directamente el valor para la llave en cuestión. Sin embargo, en el servidor máster, en la función put() se gestiona el protocolo del Two-Phase Commit mediante el cual se pregunta a los nodos esclavos si pueden hacer commit y si todos están de acuerdo, los cambios se guardan. El proceso de modificación de valores está controlado por el uso de semáforos, ya que, como se ha comentado antes consideramos esta operación y, en concreto esta parte, crítica. Una vez finalizado todo el proceso, se modifican los cambios en el fichero .json que sirve como “backup”.

Finalmente, la ejecución de estos servidores está automatizada en el fichero “centralized.py”, en el cual, se asigna un thread para el máster y se inicia y se asignan otros dos threads para los esclavos y se inician también.

Implementación descentralizada

Para esta implementación se ha implementado una única clase, ya que, a diferencia del centralizado, en esta los tres servidores tienen el mismo peso, es decir no hay un servidor que controle por encima de otros. En esta también encontramos el método “init()” en el cual se inicializan las variables empleadas para el servidor y el diccionario de clave valor para cada una de las clases. Este se inicializa mediante un fichero .json creado a modo de recuperación en caso de tener algún fallo en el sistema. Este se actualiza a cada petición de “put” que se realiza para actualizar los valores. En esta implementación, no se ha podido implementar la función de descubrimiento de puertos para asignarlos a los servidores y estos se han asignado manualmente como se observa en el código entregado. No obstante, pese a no haber podido realizar la implementación, se conoce la forma en la cual se hubiese hecho. En este caso, los tres servidores deberían ser descubiertos mediante el método y almacenar los puertos en los que están corriendo.

En relación con la implementación de las operaciones get() y put(), como en la anterior implementación, se ha decidido usar semáforos para evitar posibles accesos concurrentes de diferentes hilos y así, garantizar una mejor consistencia. No obstante, la decisión de si realizar o no estas operaciones esta basada en los votos de los servidores, que se controlan mediante quórums, uno para la escritura y otro para la lectura. En ambos métodos se comprueba si los votos emitidos por los nodos son igual o mayor al valor que tienen fijado de quorum para cada operación. En caso afirmativo, se lleva a cabo esta y en caso negativo, no se realiza la escritura o la lectura. Para realizar el proceso de voto, tenemos la función askVote(), la cual devuelve el peso de cada nodo como respuesta, que servirá para ir acumulándolo en un total e ir realizando las comprobaciones. En el caso del método put(), una vez realizadas las comprobaciones y modificados los valores, los cambios se actualizan en el fichero .json que sirve de backup.

Para las funciones restore() y slowDown(), se ha empleado la misma lógica que en la implementación centralizada. No se han usado semáforos tampoco, ya que, al ser reinicios de valores de variables o asignaciones, no hemos considerado que sea un punto crítico en el cual haya que controlar los accesos.

Finalmente, la ejecución de estos servidores está automatizada en el fichero “decentralized.py”, en el cual, a diferencia de la implementación anterior, se crean tres hilos, uno para cada servidor, y se le asigna el puerto donde corre y el peso que tiene para poder comprobar los quórums.

Comparación de las implementaciones

Se ha decidido exportar los resultados obtenidos de los tests, para ambas implementaciones, a ficheros .txt para así facilitar la legibilidad de estos y evitar volver a ejecutar para visualizar los resultados.

En aspectos generales, nuestras dos implementaciones responden correctamente a las operaciones de escritura, lectura, tienen buena concurrencia y tienen una correcta tolerancia a los fallos. Sin embargo, se observa que en las dos implementaciones la parte de rendimiento no se supera correctamente pese a haber incrementado el tiempo de 10 a 60 segundos.

A continuación, se adjuntan los resultados obtenidos:

Resultados Implementación centralizada

```
Performed 400 operations in 81.31 seconds.
Performed 400 operations in 81.73 seconds (slowing down master).
Performed 400 operations in 92.14 seconds (slowing down master).

Centralized Test Results Summary:
+------------------------------------------------------------+--------+------------------------------------------------------------
|                         Test Case                          | Result |                          Details
+------------------------------------------------------------+--------+------------------------------------------------------------
|          test_system_scalability_and_performance           |  FAIL  |                    Traceback (most recent call last):
|                                                            |        |  File "C:\URV\SD\Distributed-Storage-Systems\SD-Task2-2024-base-code_V1.3\eval\centralized_system_t
|                                                            |        |               assert duration < 60, "The system took too long to per
|                                                            |        |                                     ^^^^^^^^^^^^^
|                                                            |        |               AssertionError: The system took too long to perform
| test_system_scalability_and_performance_with_slowdown_master |  FAIL  |                    Traceback (most recent call last):
|                                                            |        | File "C:\URV\SD\Distributed-Storage-Systems\SD-Task2-2024-base-code_V1.3\eval\centralized_system_tests.py",
|                                                            |        |               assert duration < 60, "The system took too long to per
|                                                            |        |                                     ^^^^^^^^^^^^^
|                                                            |        |               AssertionError: The system took too long to perform
| test_system_scalability_and_performance_with_slowdown_slave |  FAIL  |                    Traceback (most recent call last):
|                                                            |        | File "C:\URV\SD\Distributed-Storage-Systems\SD-Task2-2024-base-code_V1.3\eval\centralized_system_tests.py",
|                                                            |        |               assert duration < 60, "The system took too long to per
|                                                            |        |                                     ^^^^^^^^^^^^^
|                                                            |        |               AssertionError: The system took too long to perform
|                           Total                            | PASSED |                                     5/8
+------------------------------------------------------------+--------+------------------------------------------------------------
```

Resultados Implementación descentralizada

```
Performed 400 operations in 62.02 seconds.
Performed 400 operations in 64.75 seconds (slowing down master).

Decentralized Test Results Summary:
+-------------------------------------------------------+--------+------------------------------------------------------------
|                       Test Case                       | Result |                          Details
+-------------------------------------------------------+--------+------------------------------------------------------------
|        test_system_scalability_and_performance        |  FAIL  |                    Traceback (most recent call last):
|                                                       |        |  File "C:\URV\SD\Distributed-Storage-Systems\SD-Task2-2024-base-code_V1.3\eval\decentralized_system_tests.py"
|                                                       |        |               self.assertLess(duration, 60, "The system took too long to perform t
|                                                       |        |   AssertionError: 62.02127456665039 not less than 60 : The system took too long tc
| test_system_scalability_and_performance_with_slowdown |  FAIL  |                    Traceback (most recent call last):
|                                                       |        | File "C:\URV\SD\Distributed-Storage-Systems\SD-Task2-2024-base-code_V1.3\eval\decentralized_system_tests.py", line
|                                                       |        |               assert duration < 60, "The system took too long to perform the c
|                                                       |        |                                     ^^^^^^^^^^^^^
|                                                       |        |               AssertionError: The system took too long to perform the operat
|                         Total                         | PASSED |                                     5/7
```

La principal diferencia entre estos es el tiempo necesitado para ejecutar las tareas. Como se observa en las imágenes, la implementación descentralizada obtiene tiempos menores. Esto tiene sentido y es correcto, ya que, en el modelo centralizado al iniciar el protocolo Two-Phase Commit, el servidor máster tiene que esperar la respuesta, ya sea confirmación o no, de todos los otros nodos. Sin embargo, en el modelo descentralizado, el protocolo basado en quórums no requiere la aprobación de todos los nodos, simplemente con llegar al valor del quorum establecido ya se puede realizar la operación en cuestión.
Como conclusión general, se obtienen mejores resultados en el modelo descentralizado, ya que, el nombre de tests superados es mayor en comparación al otro modelo.

Questions

Q1 Justify the consistency level of your centralised implementation.

La consistencia en la implementación está garantizada por dos aspectos principales. El primero, es el protocolo Two-Phase Commit, mediante el cuál el servidor máster solo realizará los cambios si todos los otros nodos esclavos están de acuerdo con realizarlos. El otro aspecto es el uso de semáforos, para evitar accesos simultáneos de diferentes hilos a variables o partes del código críticas. Gracias a esta implementación e garantiza que un único hilo está ejecutando la parte de código pertinente o modificando el valor de una variable.

Q2 Characterize each of your implementations according to the CAP theorem.

El teorema del CAP establece que es imposible para un sistema distribuido garantizar simultáneamente las tres siguientes propiedades, Consistencia, Disponibilidad y Tolerancia a particiones.

En nuestro caso, en la implementación centralizada se garantiza la consistencia utilizando el protocolo Two-Phase Commit explicado anteriormente y empleando el uso de semáforos. También proporcionamos disponibilidad, ya que, siempre se proporciona una respuesta, ya sea True o False, a la hora de realizar operaciones de escritura o lectura. Sin embargo, no garantiza la tolerancia a fallos, ya que, si cae el servidor máster, no se podrían hacer operaciones de escritura, por ejemplo, o si cae algún nodo esclavo no se obtendrían todas las respuestas.

Por otro lado, en la implementación descentralizada, se garantiza disponibilidad, ya que, como en el caso anterior, siempre se proporciona una respuesta a una petición de operación a realizar. Además, también proporciona tolerancia a fallos, ya que, no es necesario obtener las respuestas de todos los nodos para realizar operaciones. Además, en caso de que un nodo caiga la información no se pierde ya que el resto de los nodos contiene la misma información (replicado).

Link al repositorio

https://github.com/IvanArFe/Distributed-Storage-Systems.git