ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA
Universitat Rovira i Virgili

[DΣIM]

Sistemes Distribuïts

Pràctica 2: Sistema d'almacenament distribuït

Curs 23/24

Estudiants: Víctor Fosch Tena

Miguel Robledo Kusz

ÍNDEX

Abstract

En esta práctica se desarrollan dos modelos de sistemas de almacenamiento distribuido: el primero con un enfoque centralizado y el segundo con un enfoque descentralizado.

Sistema centralizado: El clúster del sistema centralizado consta de tres servidores, uno maestro y dos esclavos. Solo el maestro puede recibir solicitudes de tipo "put", mientras que las solicitudes de tipo "get" pueden ser procesadas por cualquier nodo, tanto el maestro como los esclavos. Para lograr consistencia en este sistema, se utiliza el protocolo de Commit en Dos Fases (2PC). Primero, el nodo maestro envía una solicitud de "canCommit" a los otros nodos, y luego, basado en las respuestas recibidas, envía una solicitud de "doCommit" en la cual todos los nodos confirman la información recibida.

Sistema descentralizado: En el sistema descentralizado, hay tres servidores de igual rango, por lo que cualquier nodo puede recibir y procesar tanto solicitudes "put" como "get". Para lograr consistencia en este sistema, se implementa un método de replicación basado en pesos y cuórum. Cada nodo en el clúster tiene asignado un peso, y para responder a las solicitudes "put" y "get", se debe alcanzar un cierto cuórum mediante la suma de los pesos de los nodos que responden afirmativamente.

System design and discussion

Centralized system

Podemos distinguir tres partes necesarias para que el sistema funcione correctamente. Primero, tenemos el archivo *'centralized.yaml'*, donde se almacenan los parámetros de configuración codificados para el clúster. Luego, está el directorio *'saves'*, donde se guarda un archivo de texto que es una réplica de la información de cada nodo para la persistencia. Finalmente, tenemos el archivo *'centralized.py'*, donde ocurren todos los procesos de inicio de los nodos, los reconocimientos de los nodos y la implementación de la API.

Obviamente, el archivo más importante es *'centralized.py'*, así que aquí están algunas decisiones de diseño que hemos tomado:

Para que el nodo master sepa a qué nodos esclavo debe incluir para el protocolo 2PC, cuando un nodo esclavo se inicia, envía una petición point-to-point al nodo maestro, y este se guarda la información en un set.

```
# Once the slave node is up, notifies the master node
channel = grpc.insecure_channel(f"{configuration['master']['ip']}:{configuration['master']['port']}")
notify_stub = store_pb2_grpc.KeyValueStoreStub(channel)
response = notify_stub.notifyMaster(store_pb2.SlaveConfiguration(config=config))
```

Para que el sistema sea tolerante a fallas de un nodo, o que pueda mantener su estado cuando se paran los nodos, hemos incluido dos métodos (read_file y persistent_save) que leen y guardan toda la información de los nodos a diferentes archivos de texto. Los archivos de texto tienen por nombre la ip y puerto del nodo al que corresponden.

Como ya se ha explicado anteriormente, las peticiones *'put'* sólo las recibe el nodo master, por lo que en la función put se incluye el proceso del 2PC con sus respectivas llamadas a las funciones *'canCommit'* y *'doCommit'*. En la función *'doCommit'* los nodos esclavos guardan la información, y por último lo hace el nodo maestro en la función *'put'*.

Decentralized system

Al igual que el sistema centralizado, el descentralizado consta de tres partes. El archivo *'decentralized.yaml'*, donde se almacenan los parámetros de configuración codificados para el clúster. Luego, está el directorio *'saves'*, donde se guarda un archivo de texto que es una réplica de la información de cada nodo para la persistencia. Finalmente, tenemos el archivo *'decentralized.py'*, donde ocurren todos los procesos de inicio de los nodos, los reconocimientos de los nodos y la implementación de la API.

Al igual que en el apartado anterior, vamos a comentar las decisiones de diseño tomadas para esta parte:

En este caso, todos los nodos del sistema tienen que tener conocimiento del resto de nodos, para ello se ha escogido, por simplicidad, el siguiente método:

Se ha guardado una parte de memoria compartida, en la que se encuentra una lista. Cuando un nodo se inicia, se inscribe en dicha lista. Luego cada nodo tiene un set propio donde tiene registrados todos los nodos del sistema.

En este caso todos los nodos pueden recibir peticiones *put*, por lo que en el método put, se recorre el set con todos los nodos para iniciar la votación. Luego si el quórum requerido se alcanza, se hace una llamada a *'doCommit'* para cada nodo.

El método para guardar la información en archivos de texto, es el mismo que se ha explicado en el apartado del sistema centralizado.

System comparison

Centralized results:

```
Centralized Test Results Summary:
+--------------------------------------------+--------+-------------------------------------------------------------------------------------------------------------------------+
|                 Test Case                  | Result |                                                        Details                                                          |
+--------------------------------------------+--------+-------------------------------------------------------------------------------------------------------------------------+
|   test_node_failure_during_transaction     |  FAIL  |                                           Traceback (most recent call last):                                            |
|                                            |        |   File "/home/miguelrk/Practica2_SD/SD-Task2-2024-base-code/eval/centralized_system_tests.py", line 366, in test_node_failure_during_transaction |
|                                            |        |          self.assertEqual(response.value, "recovery_value", "Data was not correctly recovered.")                         |
|                                            |        |                                        AssertionError: '' != 'recovery_value'                                           |
|                                            |        |                                    + recovery_value : Data was not correctly recovered.                                 |
| test_state_recovery_after_critical_failure |  FAIL  |                                           Traceback (most recent call last):                                            |
|                                            |        | File "/home/miguelrk/Practica2_SD/SD-Task2-2024-base-code/eval/centralized_system_tests.py", line 345, in test_state_recovery_after_critical_failure |
|                                            |        |      self.assertEqual(response.value, "stable_value", "Data did not recover correctly after critical failure.")         |
|                                            |        |                                         AssertionError: '' != 'stable_value'                                            |
|                                            |        |                          + stable_value : Data did not recover correctly after critical failure.                        |
|                   Total                    | PASSED |                                                          6/8                                                            |
+--------------------------------------------+--------+-------------------------------------------------------------------------------------------------------------------------+
```

Decentralized results:

```
Decentralized Test Results Summary:
+--------------------------------------------+--------+-------------------------------------------------------------------------------------------------------------------------+
|                 Test Case                  | Result |                                                        Details                                                          |
 |
+--------------------------------------------+--------+-------------------------------------------------------------------------------------------------------------------------+
|   test_node_failure_during_transaction     |  FAIL  |                                           Traceback (most recent call last):                                            |
 |
|                                            |        |   File "/home/miguelrk/Practica2_SD/SD-Task2-2024-base-code/eval/decentralized_system_tests.py", line 279, in test_node_failure_during_transaction    |
|                                            |        |          self.assertEqual(get_response.value, "recovery_value", "Data was not correctly recovered.")
 |
|                                            |        |                                        AssertionError: '' != 'recovery_value'
 |
|                                            |        |                                    + recovery_value : Data was not correctly recovered.
 |
| test_state_recovery_after_critical_failure |  FAIL  |                                           Traceback (most recent call last):
 |
|                                            |        | File "/home/miguelrk/Practica2_SD/SD-Task2-2024-base-code/eval/decentralized_system_tests.py", line 259, in test_state_recovery_after_critical_failure |
|                                            |        |      self.assertEqual(response_get.value, "stable_value", "Data did not recover correctly after critical failure.")
 |
|                                            |        |                                         AssertionError: '' != 'stable_value'
 |
|                                            |        |                          + stable_value : Data did not recover correctly after critical failure.
 |
|                                            |        |
(SD-env) miguelrk@LAPTOP-TDKU8579:~/Practica2_SD/SD-Task2-2024-base-code$ 
|                                            |        |                    During handling of the above exception, another exception occurred:
 |
|                                            |        |
 |
|                                            |        |                                           Traceback (most recent call last):
 |
|                                            |        | File "/home/miguelrk/Practica2_SD/SD-Task2-2024-base-code/eval/decentralized_system_tests.py", line 262, in test_state_recovery_after_critical_failure |
|                                            |        |                          self.fail("Error during state recovery test: %s" % str(e))
 |
|                                            |        |                     AssertionError: Error during state recovery test: '' != 'stable_value'
 |
|                                            |        |                          + stable_value : Data did not recover correctly after critical failure.
 |
|                   Total                    | PASSED |                                                          5/7
 |
+--------------------------------------------+--------+-------------------------------------------------------------------------------------------------------------------------+
```

Ambos sistemas presentan algunos errores en relación a la persistencia después de una fallada de nodo o después de parar el sistema.

Aunque ambos sistemas guardan el contenido de los nodos en ficheros, hay algunas veces que la información no se replica correctamente, y a la hora de intentar hacer un *get* de su valor no se encuentran.

Dicho esto cabe aclarar que tanto el guardado en ficheros como la replicación de contenido entre los nodos funciona en la mayoría de los casos, pero en ciertos casos debido al multithreading y las paradas repentinas es posible que la información no termine de replicarse o no se replique correctamente.

Questions

Q1 Justify the consistency level of your centralized implementation.

En el sistema centralizado, la consistencia viene dada tanto por parte de los ficheros de texto a la hora de recuperar el estado del sistema después de un parón, como por el Two-Phase Protocol.

Aunque en este sistema, sólo un nodo puede procesar las peticiones *put*, gracias al 2PC todos los nodos disponen de la información, y por tanto cuando el cliente quiera acceder a ella, no tendrá que preocuparse de encontrar qué nodo la contiene.

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

- Sistema centralizado

  Consistència: En un sistema centralizado con el protocolo 2PC (Two-Phase Commit), puede ser más factible lograr consistencia ya que el maestro coordina todas las operaciones y puede asegurar que todos los nodos confirmen antes de comprometer cambios.

  Disponibilidad: La disponibilidad es alta para las peticiones *get*, pero en las peticiones *put* puede verse afectada en caso de fallar el nodo maestro.

  Tolerancia a particiones: La tolerancia a particiones es un poco limitada, ya que el sistema puede verse afectado si hay una partición que afecta al maestro, pues dejarán de poderse procesar peticiones *put*.

- Sistema descentralizado

  Consistència: Aunque la consistencia en los sistemas descentralizados suele ser más complicada, este sistema es bastante consistente. Dicho esto es importante recalcar, que a diferencia del 2PC, pueden aceptarse peticiones *put* sin que todos los nodos estén de acuerdo, pues el quórum necesario es inferior al quórum total del sistema.

Disponibilidad: En este caso la disponibilidad es alta tanto para peticiones *put* como *get*, ya que mientras estén activos los nodos suficientes para alcanzar el quórum, podrán procesarse.

Tolerancia a particiones: En los sistemas descentralizados la tolerancia a particiones es más alta, igual en nuestro caso, pues aunque un nodo se encuentre afectado, el sistema puede seguir funcionando con cierta normalidad.